복셀 기반 3D 공간 연산자 실행 모듈 및 가시화 도구 제작

3D Voxel Operator API Specification



Java 버전: 17.0.20

문서 버전: 1.0.16

발행일: 2026 년 10 월

본 문서는 3D Voxel 공간 연산 처리를 위한 API 명세서입니다.

목 차

1. BasicDataCube 6

1.1. BasicDataCube.DataType 6

1.2. BasicDataCube.Resolution 8

1.3. BasicDataCube.Vector2 9

1.4. BasicDataCube.Vector3 11

1.5. BasicDataCube.Vector4 13

1.6. BasicDataCube 15

2. Neighbor 31

3. VoxelOpClipping [벡터] 33

3.1. clipping API 34

4. VoxelOpIntersection [벡터] 38

4.1. intersection API 38

5. VoxelOpDifference [벡터] 42

5.1. difference API 42

6. VoxelOpUnion [벡터] 46

6.1. union API 46

7. VoxelOpBuffer [벡터] 50

7.1. buffer API 51

8. VoxelOpSurface [벡터] 54

8.1. surface API 55

9. VoxelOpFill [벡터] 58

9.1. fillNeighborFlood API 61

9.2. fillRayCasting API 63

9.3. fillDEM API 66

9.4. fillCircle API 68

10. VoxelOpArithmetic [레스터] 72

10.1. add API 77
10.2. sub API 80
10.3. mul API 84
10.4. div API 87
10.5. mod API 91
10.6. mean API 95
10.7. min API 97
10.8. max API 99
10.9. variance API 101
10.10. standardDeviation API 103
10.11. pow API 105
10.12. sqrt API 107
10.13. log API 110
10.14. sin API 112
10.15. cos API 115
10.16. tan API 117
11. VoxelOpComparison [레스터] 121
11.1. compare API 124
11.2. compareRange API 130
12. VoxelOpLogical [레스터] 133
12.1. mask API 136
12.2. and API 143
12.3. or API 146
12.4. xor API 150
12.5. not API 154
13. VoxelOpFocal [레스터] 157
13.1. mean API 161
13.2. sum API 163

13.3. min API 166
13.4. max API 168
13.5. standardDeviation API 170
13.6. variance API 172
13.7. majority API 175
13.8. median API 177
13.9. count API 180
14. VoxelOpLocal [레스터] 184
14.1. mean API 193
14.2. sum API 196
14.3. min API 198
14.4. max API 201
14.5. standardDeviation API 203
14.6. variance API 206
14.7. majority API 208
14.8. median API 211
14.9. pow API 214
14.10. sqrt API 216
14.11. log API 219
14.12. sin API 222
14.13. cos API 225
14.14. tan API 228
14.15. countCompare API 231
14.16. classifyCompare API 234
14.17. classifyRange API 238
14.18. classifyScore API 241
15. VoxelOpDistance [레스터] 246
15.1. distanceEuclidean API 249

15.2. distanceManhattan API 253

15.3. distanceChebyshev API 256

16. VoxelOpConnectedComponent [레스터] 261

16.1. connectedComponent API 263

17. VoxelOpEarthwork [벡터] 270

17.1. earthworkGround API 271

17.2. earthworkRoad API 275

1. BasicDataCube

1.1. BasicDataCube.DataType

클래스 설명

```
public static enum BasicDataCube.DataType
    extends Enum<BasicDataCube.DataType>
```

BDC 바이너리 포맷에서 사용하는 데이터 타입 정의 Enum.

BDC 파일 헤더에는 1 바이트 코드(1~9)로 저장되며, 내부에서는 자바 데이터 타입 클래스, 바이트 크기, 비트 패킹 여부 등을 관리.

Enum Constant Summary

Enum Constants

Enum Constant	Description
BIT	BIT 타입 (1-bit, Boolean)
BYTE	BYTE 타입 (1-byte, Byte)
DOUBLE	배정밀도 DOUBLE 타입 (8-byte, Double)
FLOAT	단정밀도 FLOAT 타입 (4-byte, Float)
INT	INT 타입 (4-byte, Integer)
SHORT	SHORT 타입 (2-byte, Short)
VECTOR2	2 차원 벡터 타입 (16-byte, Double x 2)
VECTOR3	3 차원 벡터 타입 (24-byte, Double x 3)
VECTOR4	4 차원 벡터 타입 (32-byte, Double x 4)

Enum Constant Details

BIT

```
public static final BasicDataCube.DataType BIT
```

BIT 타입 (1-bit, Boolean)

BYTE

```
public static final BasicDataCube.DataType BYTE
```

BYTE 타입 (1-byte, Byte)

SHORT

```
public static final BasicDataCube.DataType SHORT
```

SHORT 타입 (2-byte, Short)

INT

```
public static final BasicDataCube.DataType INT
```

INT 타입 (4-byte, Integer)

FLOAT

```
public static final BasicDataCube.DataType FLOAT
```

단정밀도 FLOAT 타입 (4-byte, Float)

DOUBLE

```
public static final BasicDataCube.DataType DOUBLE
```

배정밀도 DOUBLE 타입 (8-byte, Double)

VECTOR2

```
public static final BasicDataCube.DataType VECTOR2
```

2 차원 벡터 타입 (16-byte, Double x 2)

VECTOR3

```
public static final BasicDataCube.DataType VECTOR3
```

3 차원 벡터 타입 (24-byte, Double x 3)

VECTOR4

```
public static final BasicDataCube.DataType VECTOR4
```

4 차원 벡터 타입 (32-byte, Double x 4)

1.2. BasicDataCube.Resolution

클래스 설명

```
public static enum BasicDataCube.Resolution
    extends Enum<BasicDataCube.Resolution>
```

BDC 공간 해상도 정의 Enum.

BDC 바이너리 포맷에서는 resolution 을 1 바이트 코드(1~2)로 저장. 한 축당 복셀 눈금 개수(64x64x64 또는 128x128x128)를 정의.

Enum Constant Summary

Enum Constants

Enum Constant	Description
R128	128x128x128 해상도 (코드: 2)
R64	64x64x64 해상도 (코드: 1)

Enum Constant Details

R64

```
public static final BasicDataCube.Resolution R64
```

64x64x64 해상도 (코드: 1)

R128

```
public static final BasicDataCube.Resolution R128
```

128x128x128 해상도 (코드: 2)

1.3. BasicDataCube.Vector2

클래스 설명

```
public static final class BasicDataCube.Vector2
    extends Object
```

2 차원 벡터 클래스.

Field Summary

Fields

Modifier and Type	Field	Description
double	x	X 좌표
double	y	Y 좌표

Constructor Summary

Constructors

Constructor	Description
Vector2(double x, double y)	Vector2 생성자.

Method Summary

Modifier and Type	Method	Description
String	toString()	Vector2 객체의 문자열 표현을 반환.

Field Details

x

```
public double x
```

X 좌표

y

```
public double y
```

Y 좌표

Constructor Details

Vector2

```
public Vector2(double x,
    double y)
```

Vector2 생성자.

Parameters:

x - X 좌표

`y` - Y 좌표

Method Details

toString

```
public String toString()
```

Vector2 객체의 문자열 표현을 반환.

Overrides:

`toString` in class `Object`

Returns:

"Vector2[x, y]" 형식의 문자열

1.4. BasicDataCube.Vector3

클래스 설명

```
public static final class BasicDataCube.Vector3
    extends Object
```

3 차원 벡터 클래스.

Field Summary

Fields

Modifier and Type	Field	Description
double	x	X 좌표
double	y	Y 좌표
double	z	Z 좌표

Constructor Summary

Constructors

Constructor	Description
Vector3(double x, double y, double z)	Vector3 생성자.

Method Summary

Modifier and Type	Method	Description
String	toString()	Vector3 객체의 문자열 표현을 반환.

Field Details

x

```
public double x
```

X 좌표

y

```
public double y
```

Y 좌표

z

```
public double z
```

Z 좌표

Constructor Details

Vector3

```
public Vector3(double x,
    double y,
    double z)
```

Vector3 생성자.

Parameters:

`x` - X 좌표

`y` - Y 좌표

`z` - Z 좌표

Method Details

toString

```
public String toString()
```

Vector3 객체의 문자열 표현을 반환.

Overrides:

`toString` in class `Object`

Returns:

"Vector3[x, y, z]" 형식의 문자열

1.5. BasicDataCube.Vector4

클래스 설명

```
public static final class BasicDataCube.Vector4
    extends Object
```

4 차원 벡터 클래스.

Field Summary

Fields

Modifier and Type	Field	Description
double	w	W 속성
double	x	X 좌표
double	y	Y 좌표
double	z	Z 좌표

Constructor Summary

Constructors

Constructor	Description
Vector4(double x, double y, double z, double w)	Vector4 생성자.

Method Summary

Modifier and Type	Method	Description
String	toString()	Vector4 객체의 문자열 표현을 반환.

Field Details

x

```
public double x
```

X 좌표

y

```
public double y
```

Y 좌표

z

```
public double z
```

Z 좌표

w

```
public double w
```

W 속성

Constructor Details

Vector4

```
public Vector4(double x,
    double y,
    double z,
    double w)
```

Vector4 생성자.

Parameters:

`x` - X 좌표

`y` - Y 좌표

`z` - Z 좌표

`w` - W 속성

Method Details

toString

```
public String toString()
```

Vector4 객체의 문자열 표현을 반환.

Overrides:

`toString` in class `Object`

Returns:

"Vector4[x, y, z, w]" 형식의 문자열

1.6. BasicDataCube

클래스 설명

```
public final class BasicDataCube
    extends Object
```

베이직 데이터 큐브 (Basic Data Cube, BDC) 클래스.

구분	Name	Type	Length (byte)	내용	필수
Header	MAGIC	byte	5	"bcube", Datacube 포맷 식별자	Y
	version	byte	2	버전 정보, 현재 버전: 1	Y
	level	byte	1	0레벨부터 시작	Y
	col	int32	4	본초자오선 (0도)를 0으로 하는 격자의 번호	Y
	row	int32	4	-90도를 0으로 하는 격자의 번호	Y
	floor	int32	4	해수면을 0으로 하는 격자의 번호	Y
	data_block_count	int16	2	공간데이터큐브에 포함된 data block의 수. Basic Datacube: 1로 고정, Application Datacube: 포함된 레이어 개수와 동일	Y
	data_block_offset	array[int32]	4*data_block_count	data block offset 배열. [0, block1 Length….]	N
Data Blocks	structured_voxeldata			data_block_count만큼 반복	N

구분	Name	Type	Length (byte)	내용	필수
Structured_VoxelData	data_block_code	byte	1	"S": Structured voxel data code, "V": all values Valid, sentinel 미사용	Y
	datatype	byte	1	1: bit 2: byte(1), 3: Int16(2), 4: Int32(4), 5: float(4), 6: double(8), 7: Vector2(16), 8: Vector3(24), 9: Vector4(32)	Y
	resolution	byte	1	1: 64*64*64배열, 2: 128*128*128배열	Y
	sentinel	byte	datatype size	NULL을 표현하는 임의의 값. bit: 0, byte/Int16/Int32: 사용자 임의 지정(데이터 유효범위 침범 방지), float/double: NaN Vector2: [NaN, NaN] Vector3: [NaN, NaN, NaN] Vector4: [NaN, NaN, NaN, NaN]	Y
	voxel_values	bytea	datatype* resolution^3	정규화된 배열로 저장되는 데이터. (Array[x][y][z]를 c-order로 평탄화)	N

3 차원 공간 복셀(Voxel) 데이터를 관리하며, 바이너리 포맷(BDC) 파싱 및 인코딩, 좌표-인덱스 변환, 센티넬(Sentinel, null 표현) 값 처리 등을 지원.

Nested Class Summary

Nested Classes

Modifier and Type	Class	Description
`static enum`	`BasicDataCube.DataType`	BDC 바이너리 포맷에서 사용하는 데이터 타입 정의 Enum.

static enum	BasicDataCube.Resolution	BDC 공간 해상도 정의 Enum.
static final class	BasicDataCube.Vector2	2 차원 벡터 클래스.
static final class	BasicDataCube.Vector3	3 차원 벡터 클래스.
static final class	BasicDataCube.Vector4	4 차원 벡터 클래스.

Constructor Summary

Constructors

Constructor	Description
BasicDataCube(byte[] bdcBinary)	BDC 바이너리 바이트 배열로부터 BasicDataCube 객체를 생성.
BasicDataCube(byte level, int col, int row, int floor, BasicDataCube.DataType datatype, BasicDataCube.Resolution resolution, Object sentinel, Object initValue)	level,col,row,floor,datatype,resolution,sentinel,initValue 를 사용해서 초기화된 객체를 생성.
BasicDataCube(String bdcFilePath)	지정한 BDC 파일 경로의 파일 데이터를 읽어 객체를 생성.

Method Summary

Modifier and Type	Method	Description
int	col()	컬럼 인덱스를 반환.
byte	dataBlockCode()	데이터 블록 코드를 반환.
String	dataBlockCodeStr()	데이터 블록 코드 문자열을 반환.
short	dataBlockCount()	데이터 블록 수를 반환.
int	dataBlockOffset()	데이터 블록 오프셋을 반환.
BasicDataCube.DataType	datatype()	데이터 타입을 반환.
byte	datatypeByte()	데이터 타입 코드를 반환.
int	floor()	높이 인덱스를 반환.
byte[]	getBdcBinary()	BasicDataCube 객체의 전체 바이너리 데이터를 byte 배열로 반환.
boolean	getBit(int x, int y, int z)	복셀좌표로 BIT 타입 복셀 데이터 획득.
byte	getByte(int x, int y, int z)	복셀좌표로 BYTE 타입 복셀 데이터 획득.
double	getDouble(int x, int y, int z)	복셀좌표로 DOUBLE 타입 복셀 데이터 획득.
float	getFloat(int x, int y, int z)	복셀좌표로 FLOAT 타입 복셀 데이터 획득.

int	getInt(int x, int y, int z)	복셀좌표로 INT 타입 복셀 데이터 획득.
short	getShort(int x, int y, int z)	복셀좌표로 SHORT 타입 복셀 데이터 획득.
BasicDataCube.Vector2	getVector2(int x, int y, int z)	복셀좌표로 VECTOR2 타입 복셀 데이터 획득.
BasicDataCube.Vector3	getVector3(int x, int y, int z)	복셀좌표로 VECTOR3 타입 복셀 데이터 획득.
BasicDataCube.Vector4	getVector4(int x, int y, int z)	복셀좌표로 VECTOR4 타입 복셀 데이터 획득.
boolean	isSentinel(int x, int y, int z)	복셀좌표로 복셀 데이터가 널을 표현하는 값(센티넬)인지 확인.
byte	level()	레벨 인덱스를 반환.
byte[]	magic()	큐브 포맷 식별자 복사본을 반환.
String	magicStr()	큐브 포맷 식별자 문자열을 반환.
BasicDataCube.Resolution	resolution()	데이터 해상도를 반환.
byte	resolutionByte()	데이터 해상도 코드를 반환.
int	row()	로우 인덱스를 반환.
byte[]	sentinel()	센티넬 (널표현) 바이트 배열 반환.
Object	sentinelObj()	센티넬(Sentinel, 널표현값) 객체를 반환.
void	setBit(int x, int y, int z, boolean value)	복셀좌표로 BIT 타입 복셀 데이터 설정.
void	setByte(int x, int y, int z, byte value)	복셀좌표로 BYTE 타입 복셀 데이터 설정.
void	setDouble(int x, int y, int z, double value)	복셀좌표로 DOUBLE 타입 복셀 데이터 설정.
void	setFloat(int x, int y, int z, float value)	복셀좌표로 FLOAT 타입 복셀 데이터 설정.
void	setInt(int x, int y, int z, int value)	복셀좌표로 INT 타입 복셀 데이터 설정.
void	setShort(int x, int y, int z, short value)	복셀좌표로 SHORT 타입 복셀 데이터 설정.
void	setVector2(int x, int y, int z, BasicDataCube.Vector2 value)	복셀좌표로 VECTOR2 타입 복셀 데이터 설정.
void	setVector3(int x, int y, int z, BasicDataCube.Vector3 value)	복셀좌표로 VECTOR3 타입 복셀 데이터 설정.
void	setVector4(int x, int y, int z, BasicDataCube.Vector4 value)	복셀좌표로 VECTOR4 타입 복셀 데이터 설정.
String	toString()	BasicDataCube 객체의 주요 속성 정보를 문자열로 반환.

byte[]	version()	버전 정보 복사본을 반환.
String	versionStr()	버전 정보 문자열을 반환.
void	writeToFile(String bdcFilePath)	큐브 바이너리 데이터를 BDC 파일 형태로 디스크에 저장.

Constructor Details

BasicDataCube

```
public BasicDataCube(byte[] bdcBinary)
```

BDC 바이너리 바이트 배열로부터 BasicDataCube 객체를 생성. BDC 파일을 모두 읽어들인 전체 바이트 배열을 사용.

Parameters:

`bdcBinary` - BDC 포맷 규격의 전체 바이트 배열

Throws:

`NullPointerException` - binary 가 null 일 경우 발생

`IllegalArgumentException` - 바이너리 헤더 및 데이터 파싱 실패 시 발생

BasicDataCube

```
public BasicDataCube(String bdcFilePath)
    throws IOException
```

지정한 BDC 파일 경로의 파일 데이터를 읽어 객체를 생성.

Parameters:

`bdcFilePath` - 읽어올 BDC 파일 경로

Throws:

`IOException` - 파일 입출력 예외 발생 시

BasicDataCube

```
public BasicDataCube(byte level,
    int col,
    int row,
    int floor,
    BasicDataCube.DataType datatype,
    BasicDataCube.Resolution resolution,
    Object sentinel,
    Object initValue)
```

level,col,row,floor,datatype,resolution,sentinel,initValue 를 사용해서 초기화된 객체를 생성.

sentinel 과 initValue 는 datatype 에 해당하는 객체로 입력.

- bit: Boolean, byte: Byte, short: Short
- int: Integer, float: Float, double: Double

- Vector2: `BasicDataCube.Vector2`, Vector3: `BasicDataCube.Vector3`, Vector4: `BasicDataCube.Vector4`

Parameters:

`level` - 레벨 인덱스 (0 부터시작)

`col` - 컬럼 인덱스 (0 부터시작)

`row` - 로우 인덱스 (0 부터시작)

`floor` - 높이 인덱스 (0 또는 ±값)

`datatype` - 데이터 타입

`resolution` - 데이터 해상도

`sentinel` - null 을 입력하면 데이터블록코드(dataBlockCode)는 'V'가 되며 sentinel 미사용모드로 생성 (내부 sentinel 값은 모두 0 으로 초기화). 객체로 입력하면 데이터블록코드(dataBlockCode)는 'S'가 되며 sentinel 사용모드로 생성

`initValue` - 복셀값배열(voxelValues)을 어떤값으로 초기화 할건지에 대한 초기화 값. null 을 입력하면 모두 0 으로 초기화, 객체로 입력하면 모두 입력값으로 초기화

Throws:

`NullPointerException` - datatype 또는 resolution 이 null 일 경우 발생

`IllegalArgumentException` - level,col,row 가 음수이거나 sentinel,initValue 가 datatype 에 맞지 않을 경우 발생

Method Details

getBit

```
public boolean getBit(int x,
    int y,
    int z)
```

복셀좌표로 BIT 타입 복셀 데이터 획득. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

Returns:

복셀값

getByte

```
public byte getByte(int x,
    int y,
    int z)
```

복셀좌표로 BYTE 타입 복셀 데이터 획득. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

x - 복셀 x 좌표

y - 복셀 y 좌표

z - 복셀 z 좌표

Returns:

복셀값

getShort

```
public short getShort(int x,
    int y,
    int z)
```

복셀좌표로 SHORT 타입 복셀 데이터 획득. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

x - 복셀 x 좌표

y - 복셀 y 좌표

z - 복셀 z 좌표

Returns:

복셀값

getInt

```
public int getInt(int x,
    int y,
    int z)
```

복셀좌표로 INT 타입 복셀 데이터 획득. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

x - 복셀 x 좌표

y - 복셀 y 좌표

z - 복셀 z 좌표

Returns:

복셀값

getFloat

```
public float getFloat(int x,
    int y,
    int z)
```

복셀좌표로 FLOAT 타입 복셀 데이터 획득. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

Returns:

복셀값

getDouble

```
public double getDouble(int x,
    int y,
    int z)
```

복셀좌표로 DOUBLE 타입 복셀 데이터 획득. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

Returns:

복셀값

getVector2

```
public BasicDataCube.Vector2 getVector2(int x,
    int y,
    int z)
```

복셀좌표로 VECTOR2 타입 복셀 데이터 획득. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

Returns:

복셀값 (새로 생성한 `BasicDataCube.Vector2` 객체)

getVector3

```
public BasicDataCube.Vector3 getVector3(int x,
    int y,
    int z)
```

복셀좌표로 VECTOR3 타입 복셀 데이터 획득. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

Returns:

복셀값 (새로 생성한 `BasicDataCube.Vector3` 객체)

getVector4

```
public BasicDataCube.Vector4 getVector4(int x,
    int y,
    int z)
```

복셀좌표로 VECTOR4 타입 복셀 데이터 획득. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

Returns:

복셀값 (새로 생성한 `BasicDataCube.Vector4` 객체)

setBit

```
public void setBit(int x,
    int y,
    int z,
    boolean value)
```

복셀좌표로 BIT 타입 복셀 데이터 설정. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

`value` - 설정할 복셀값

setByte

```
public void setByte(int x,
    int y,
    int z,
    byte value)
```

복셀좌표로 BYTE 타입 복셀 데이터 설정. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

`value` - 설정할 복셀값

setShort

```
public void setShort(int x,
    int y,
    int z,
    short value)
```

복셀좌표로 SHORT 타입 복셀 데이터 설정. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

`value` - 설정할 복셀값

setInt

```
public void setInt(int x,
    int y,
    int z,
    int value)
```

복셀좌표로 INT 타입 복셀 데이터 설정. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

`value` - 설정할 복셀값

setFloat

```
public void setFloat(int x,
    int y,
    int z,
    float value)
```

복셀좌표로 FLOAT 타입 복셀 데이터 설정. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

`value` - 설정할 복셀값

setDouble

```
public void setDouble(int x,
    int y,
    int z,
    double value)
```

복셀좌표로 DOUBLE 타입 복셀 데이터 설정. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

`value` - 설정할 복셀값

setVector2

```
public void setVector2(int x,
    int y,
    int z,
    BasicDataCube.Vector2 value)
```

복셀좌표로 VECTOR2 타입 복셀 데이터 설정. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

`value` - 설정할 복셀값

Throws:

`NullPointerException` - value 가 null 일 경우 발생

setVector3

```
public void setVector3(int x,
    int y,
    int z,
    BasicDataCube.Vector3 value)
```

복셀좌표로 VECTOR3 타입 복셀 데이터 설정. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

`value` - 설정할 복셀값

Throws:

`NullPointerException` - value 가 null 일 경우 발생

setVector4

```
public void setVector4(int x,
    int y,
    int z,
    BasicDataCube.Vector4 value)
```

복셀좌표로 VECTOR4 타입 복셀 데이터 설정. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

`value` - 설정할 복셀값

Throws:

`NullPointerException` - value 가 null 일 경우 발생

isSentinel

```
public boolean isSentinel(int x,
    int y,
    int z)
```

복셀좌표로 복셀 데이터가 널을 표현하는 값(센티넬)인지 확인. 센티넬

미사용모드(dataBlockCode='V')이면 항상 false. 성능을 위해 x,y,z 범위를 체크하지 않음.

Parameters:

`x` - 복셀 x 좌표

`y` - 복셀 y 좌표

`z` - 복셀 z 좌표

Returns:

센티넬값이면 true

magic

```
public byte[] magic()
```

큐브 포맷 식별자 복사본을 반환.

Returns:

큐브 포맷 식별자 복사본 (기본: "bcube"를 나타내는 바이트배열)

version

```
public byte[] version()
```

버전 정보 복사본을 반환.

Returns:

버전 정보 복사본 (기본: [1, 0])

level

```
public byte level()
```

레벨 인덱스를 반환.

Returns:

레벨 인덱스 (0 부터시작)

col

```
public int col()
```

컬럼 인덱스를 반환.

Returns:

컬럼 인덱스 (0 부터시작)

row

```
public int row()
```

로우 인덱스를 반환.

Returns:

로우 인덱스 (0 부터시작)

floor

```
public int floor()
```

높이 인덱스를 반환.

Returns:

높이 인덱스 (0 또는 ±값)

dataBlockCount

```
public short dataBlockCount()
```

데이터 블록 수를 반환.

Returns:

데이터 블록 수 (1)

dataBlockOffset

```
public int dataBlockOffset()
```

데이터 블록 오프셋을 반환.

Returns:

데이터 블록 오프셋 (0)

dataBlockCode

```
public byte dataBlockCode()
```

데이터 블록 코드를 반환.

Returns:

데이터 블록 코드 ('S' 또는 'V' 를 나타내는 바이트)

datatype

```
public BasicDataCube.DataType datatype()
```

데이터 타입을 반환.

Returns:

데이터 타입 (BIT,BYTE,SHORT,INT,FLOAT,DOUBLE,VECTOR2,VECTOR3,VECTOR4)

resolution

```
public BasicDataCube.Resolution resolution()
```

데이터 해상도를 반환.

Returns:

데이터 해상도 (R64 또는 R128)

sentinel

```
public byte[] sentinel()
```

센티넬 (널표현) 바이트 배열 반환. 데이터블록코드(S 또는 V)와 상관없이 BDC 에서 읽은 그대로 반환.

센티넬 원본 바이트 배열을 복사하여 반환. 반환되는 배열은 LITTLE_ENDIAN 임.

Returns:

센티넬 바이트 배열

magicStr

```
public String magicStr()
```

큐브 포맷 식별자 문자열을 반환.

Returns:

큐브 포맷 식별자 문자열 ("bcube")

versionStr

```
public String versionStr()
```

버전 정보 문자열을 반환.

Returns:

버전 정보 문자열 ("1.0")

dataBlockCodeStr

```
public String dataBlockCodeStr()
```

데이터 블록 코드 문자열을 반환.

Returns:

데이터 블록 코드 문자열 ("S" 또는 "V")

datatypeByte

```
public byte datatypeByte()
```

데이터 타입 코드를 반환.

Returns:

데이터 타입 코드 (1~9)

resolutionByte

```
public byte resolutionByte()
```

데이터 해상도 코드를 반환.

Returns:

데이터 해상도 코드 (1 또는 2)

sentinelObj

```
public Object sentinelObj()
```

센티넬(Sentinel, 널표현값) 객체를 반환. 데이터 블록 코드의 상태에 따라 동작이 달라짐:

- 데이터 블록 코드가 `'V'`인 경우: `null` 을 반환.
- 데이터 블록 코드가 `'S'`인 경우: 정의된 센티넬 값을 `Object` 타입으로 반환.

사용 예시: (반드시 `null` 여부를 먼저 확인한 후 적절한 타입으로 캐스팅하여 사용해야 함.)

```
Object obj=cube.sentinelObj();  if(obj!=null) { boolean bit=(Boolean)obj; }
Object obj=cube.sentinelObj();  if(obj!=null) { byte    b  =(Byte)   obj; }
Object obj=cube.sentinelObj();  if(obj!=null) { short   s  =(Short)  obj; }
Object obj=cube.sentinelObj();  if(obj!=null) { int     i  =(Integer)obj; }
Object obj=cube.sentinelObj();  if(obj!=null) { float   f  =(Float)  obj; }
Object obj=cube.sentinelObj();  if(obj!=null) { double  d  =(Double) obj; }
Object obj=cube.sentinelObj();  if(obj!=null) { Vector2 v2 =(Vector2)obj; }
Object obj=cube.sentinelObj();  if(obj!=null) { Vector3 v3 =(Vector3)obj; }
Object obj=cube.sentinelObj();  if(obj!=null) { Vector4 v4 =(Vector4)obj; }
```

Returns:

센티넬 객체 (블록 코드가 `'V'`이면 `null`, `'S'`이면 지정된 센티넬 객체)

getBdcBinary

```
public byte[] getBdcBinary()
```

BasicDataCube 객체의 전체 바이너리 데이터를 byte 배열로 반환.

반환되는 배열은 BDC 파일을 구성하는 전체 데이터이며, 다음 순서로 저장.

1. Magic
2. Version
3. Level
4. Column
5. Row
6. Floor
7. Data Block Count
8. Data Block Offset
9. Data Block Code
10. Data Type
11. Resolution
12. Sentinel
13. Voxel Values

반환되는 byte 배열은 기존 내부 데이터를 직접 반환하지 않고 새로운 배열을 생성하여 복사한 결과임. 따라서 호출할 때마다 메모리 할당 및 데이터 복사 비용이 발생. 이 메서드는 빈번하게 호출하지 않는 것이 좋음.

반환되는 바이너리 데이터는 LITTLE_ENDIAN 바이트 순서로 구성.

Returns:

BDC 파일 형식으로 구성된 전체 바이너리 데이터의 복사본

writeToFile

```
public void writeToFile(String bdcFilePath)
    throws IOException
```

큐브 바이너리 데이터를 BDC 파일 형태로 디스크에 저장. 파일이 이미 존재하면 덮어 씀.

Parameters:

`bdcFilePath` - 저장할 경로명 (상대경로/절대경로)

Throws:

`IOException` - 파일 쓰기 예외 발생 시

toString

```
public String toString()
```

BasicDataCube 객체의 주요 속성 정보를 문자열로 반환.

반환되는 문자열에는 BDC 파일의 기본 정보와 데이터 블록 정보, 데이터 타입 및 해상도, sentinel 값, voxel 데이터 크기 등의 주요 속성이 포함됨.

sentinel 값은 각 바이트를 16 진수 형식으로 변환하고 공백으로 구분하여 표시.

Overrides:

`toString` in class `Object`

Returns:

BasicDataCube 객체의 주요 속성 정보를 포함하는 문자열

2. Neighbor

클래스 설명

```
public enum Neighbor
    extends Enum<Neighbor>
```

복셀 이웃(neighbor) 종류 정의 Enum.

복셀 간 인접 관계를 정의하며, 복셀 공간 연산에서 6 방향, 18 방향, 26 방향 이웃을 나타냄.

- N6 : 면(face)을 공유하는 6 개 이웃
- N18 : 면(face) 또는 모서리(edge)를 공유하는 18 개 이웃
- N26 : 면(face), 모서리(edge), 꼭짓점(vertex)을 공유하는 26 개 이웃

Enum Constant Summary

Enum Constants

Enum Constant	Description
N18	18-neighbor: 면 또는 모서리를 공유하는 18 개 이웃
N26	26-neighbor: 면, 모서리 또는 꼭짓점을 공유하는 26 개 이웃
N6	6-neighbor: 면을 공유하는 6 개 이웃

Method Summary

Modifier and Type	Method	Description
int	code()	복셀 이웃 종류의 코드값을 반환.
static Neighbor	fromCode(int code)	정수 코드값 으로부터 Neighbor Enum 을 가져옴.

Enum Constant Details

N6

```
public static final Neighbor N6
```

6-neighbor: 면을 공유하는 6 개 이웃

N18

```
public static final Neighbor N18
```

18-neighbor: 면 또는 모서리를 공유하는 18 개 이웃

N26

```
public static final Neighbor N26
```

26-neighbor: 면, 모서리 또는 꼭짓점을 공유하는 26 개 이웃

Method Details

code

```
public int code()
```

복셀 이웃 종류의 코드값을 반환.

Returns:

이웃 복셀 코드 (6, 18 또는 26)

fromCode

```
public static Neighbor fromCode(int code)
```

정수 코드값 으로부터 Neighbor Enum 을 가져옴.

Parameters:

`code` - 코드값 (6, 18 또는 26)

Returns:

해당하는 `Neighbor`

Throws:

`IllegalArgumentException` - 유효하지 않은 코드값인 경우

3. VoxelOpClipping [벡터]

클래스 설명

```
public final class VoxelOpClipping
    extends Object
```

[벡터] 클리핑 연산자.

마스크 영역을 기준으로 객체를 잘라내거나 남김.

진입점은 정적 메서드 `clipping(BasicDataCube[], BasicDataCube[], int)` 하나이며 인스턴스를 만들 수 없음.

상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

`cubes` 와 `masks` 를 같은 `(col, row, floor)` 좌표끼리 짝지어 복셀 단위로 계산함.

대응하는 마스크 큐브가 없는 좌표는 마스크가 비어 있는 것으로 봄.

입력 큐브는 변경하지 않음.

option 값

option 값별 동작

option	동작	반환 배열
`CLIPPING_INTERSECTION` (1)	마스크와 겹치는 복셀만 남김	마스크가 없거나 한쪽 복셀이 모두 0 인 좌표는 빠지므로 `cubes` 보다 짧을 수 있음. 큐브는 모두 새로 만듦
`CLIPPING_DIFFERENCE` (2)	마스크와 겹치는 복셀을 지움	`cubes` 와 길이·좌표가 같음. 마스크와 겹치지 않은 좌표는 입력 큐브 객체를 그대로 넣음

그 밖의 값은 `IllegalArgumentException` 을 던짐.

결과 큐브는 대응하는 `cubes` 큐브의 헤더를 승계하고, 데이터 블록 코드는 언제나 `'V'`, 센티넬 0 임.

`'S'` 입력의 센티넬 자리는 0 으로 보고 계산함.

지원 타입

입력과 반환 모두 bit(`BasicDataCube.DataType.BIT`).

다른 타입이 섞이면 `IllegalArgumentException` 을 던짐.

해상도는 `BasicDataCube.Resolution.R64`, `BasicDataCube.Resolution.R128` 둘 다 지원하며, 두 리스트의 모든 큐브가 같은 level 과 resolution 을 가져야 함.

교집합 모드에서 나는 예외의 메시지에는 파라미터 이름이 `cubes1`(`cubes`), `cubes2`(`masks`)로 나옴.

사용 예:

```
   BasicDataCube[] kept = VoxelOpClipping.clipping(
           cubes, masks, VoxelOpClipping.CLIPPING_INTERSECTION);   // 마스크 안쪽만 남김

   BasicDataCube[] cut  = VoxelOpClipping.clipping(
           cubes, masks, VoxelOpClipping.CLIPPING_DIFFERENCE);     // 마스크 부분을 잘라냄
```

Field Summary

Fields

Modifier and Type	Field	Description
`static final int`	`CLIPPING_DIFFERENCE`	차집합 모드 (option = 2).
`static final int`	`CLIPPING_INTERSECTION`	교집합 모드 (option = 1).

Method Summary

Modifier and Type	Method	Description
`static BasicDataCube[]`	`clipping(BasicDataCube[] cubes, BasicDataCube[] masks, int option)`	클리핑 연산.

Field Details

CLIPPING_INTERSECTION

```
public static final int CLIPPING_INTERSECTION
```

교집합 모드 (option = 1).

마스크와 겹치는 복셀만 남김.

See Also:

- Constant Field Values

CLIPPING_DIFFERENCE

```
public static final int CLIPPING_DIFFERENCE
```

차집합 모드 (option = 2).

마스크와 겹치는 복셀을 지움.

See Also:

- Constant Field Values

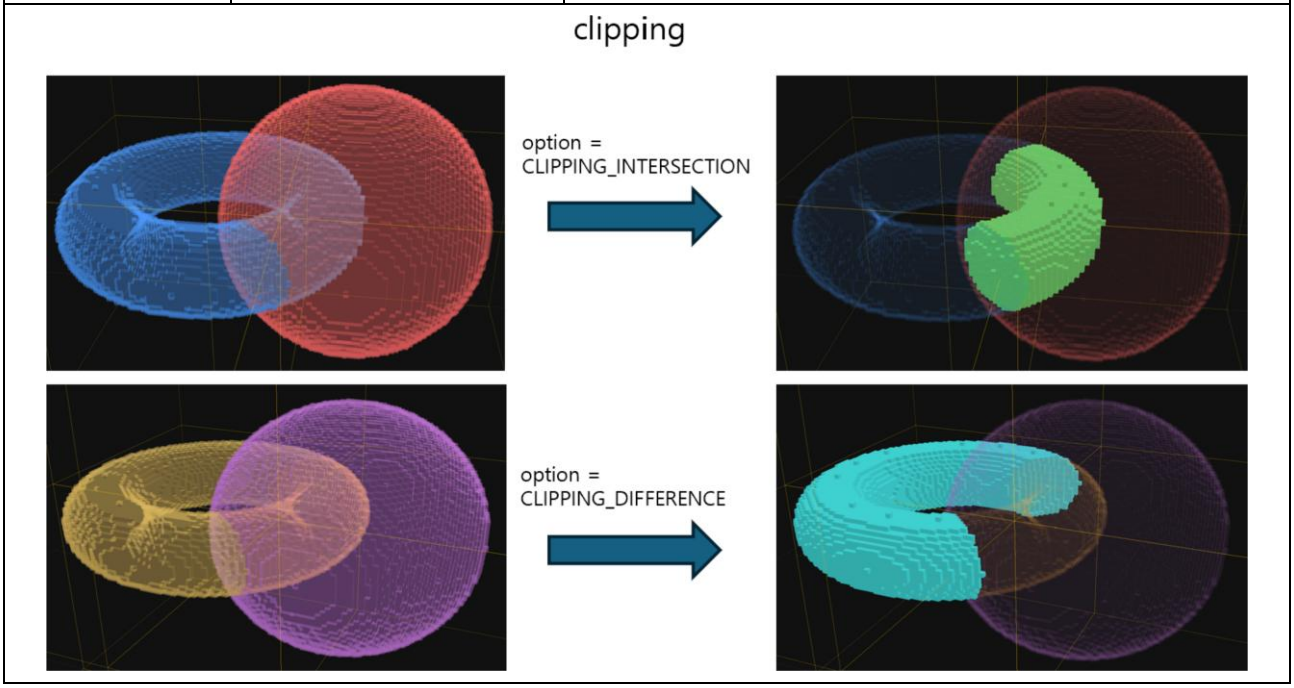
3.1. clipping API

```
public static BasicDataCube[] clipping(BasicDataCube[] cubes,
    BasicDataCube[] masks,
    int option)
```

클리핑 연산.

clipping

메서드	파라미터 / 반환	설명
clipping	(	큐브내 객체 기준으로 마스크영역을 잘라내거나, 마스크영역 부분만 추출하는 연산. (cubes, masks 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes	기준이되는 객체를 포함하는 큐브 리스트. 지원타입: bit
	BasicDataCube[] masks	마스크영역에 해당하는 큐브 리스트. 지원타입: bit
	int option)	CLIPPING_INTERSECTION: 특정영역에 해당하는 부분만 추출. (intersection 호출) CLIPPING_DIFFERENCE: 특정영역에 해당하는 부분을 잘라냄. (difference 호출)
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. (dataBlockCode='V', sentinel=0) 반환타입: bit



예제

시험 시나리오 TS-003-CLP(나선관을 아래층 마스크로 추출)의 입력 파일이 있는 폴더에서 실행.

```
TS-003-CLP>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-003-CLP>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
```

```
 import java.util.zip.ZipInputStream;
 import java.util.zip.ZipOutputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpClipping;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes = readZip("cubes.zip");
         BasicDataCube[] masks = readZip("masks.zip");
         BasicDataCube[] result = VoxelOpClipping.clipping(cubes, masks,
VoxelOpClipping.CLIPPING_INTERSECTION);
         System.out.println("result cube count: " + result.length);
         writeZip("result.zip", result);
         System.out.println("saved: result.zip");
     }


     // zip 안의 .bdc 를 저장 순서대로 읽음
     static BasicDataCube[] readZip(String path) throws Exception {
         List<BasicDataCube> list = new ArrayList<>();
         try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
             for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                 list.add(new BasicDataCube(zip.readAllBytes()));
             }
         }
         return list.toArray(new BasicDataCube[0]);
     }


     // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
     static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
         try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
             for (int i = 0; i < cubes.length; i++) {
                 BasicDataCube c = cubes[i];
                 zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                         i, c.level(), c.col(), c.row(), c.floor())));
                 zip.write(c.getBdcBinary());
                 zip.closeEntry();
             }
         }
     }


 }
```

Parameters:

`cubes` - 기준이 되는 객체를 담은 큐브 리스트.

bit 타입이어야 하고, 원소가 하나 이상이어야 하며, 모든 큐브의 level 과 resolution 이 같아야 하고, `(col, row, floor)` 좌표가 서로 달라야 함

`masks` - 마스크 영역을 담은 큐브 리스트.

조건은 `cubes` 와 같음.

`null` 또는 빈 배열을 넘기면 교집합 모드는 길이 0 인 배열을, 차집합 모드는 `cubes` 를 그대로 돌려줌

`option` - `CLIPPING_INTERSECTION` 또는 `CLIPPING_DIFFERENCE`

Returns:

연산 결과 큐브 리스트 (bit).

길이는 `option` 에 따라 다름 (클래스 설명의 표 참고)

Throws:

`IllegalArgumentException` - `option` 이 1 도 2 도 아닐 때, `cubes` 가 `null` 이거나 비었을 때, 어느 한쪽 리스트에 `null` 원소가 있을 때, bit 가 아닌 큐브가 섞였을 때, 한 리스트 안에서 또는 두 리스트 사이에서 level 이나 resolution 이 서로 다를 때, 한 리스트 안에 `(col, row, floor)`가 중복될 때

4. VoxelOpIntersection [벡터]

클래스 설명

```
public final class VoxelOpIntersection
    extends Object
```

[벡터] 교집합 연산자.

두 큐브 리스트의 객체가 겹치는 부분만 돌려줌.

진입점은 정적 메서드 `intersection(BasicDataCube[], BasicDataCube[])` 하나이며 인스턴스를 만들 수 없음.

상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

같은 `(col, row, floor)` 좌표끼리 짝지어 복셀 단위 AND 연산을 함.

입력 큐브는 변경하지 않음.

지원 타입

입력과 반환 모두 bit(`BasicDataCube.DataType.BIT`).

다른 타입이 섞이면 `IllegalArgumentException` 을 던짐.

해상도는 `BasicDataCube.Resolution.R64`, `BasicDataCube.Resolution.R128` 둘 다 지원하며, 두 리스트의 모든 큐브가 같은 level 과 resolution 을 가져야 함.

센티넬

반환 큐브는 입력과 상관없이 언제나 데이터 블록 코드 `'V'`, 센티넬 0 임.

`'S'` 입력의 센티넬 자리는 0 으로 보고 계산함.

Method Summary

Modifier and Type	Method	Description
`static BasicDataCube[]`	`intersection(BasicDataCube[] cubes1, BasicDataCube[] cubes2)`	cubes1 ∩ cubes2.

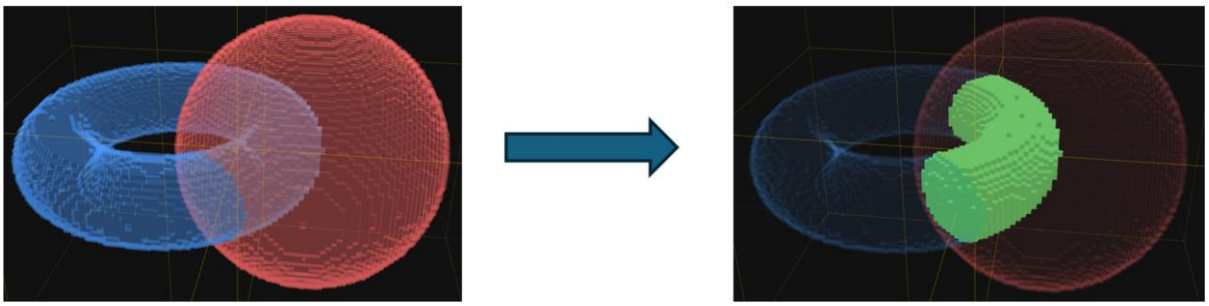
4.1. intersection API

```
public static BasicDataCube[] intersection(BasicDataCube[] cubes1,
    BasicDataCube[] cubes2)
```

cubes1 ∩ cubes2.

intersection

메서드	파라미터 / 반환	설명
intersection	(	첫번째 큐브내 객체와 두번째 큐브내 객체의 교집합 부분만 추출하는 연산.

		(cubes1, cubes2 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes1	첫번째 객체를 포함하는 큐브 리스트. 지원타입: bit
	BasicDataCube[] cubes2)	두번째 객체를 포함하는 큐브 리스트. 지원타입: bit
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. (dataBlockCode='V', sentinel=0) 반환타입: bit
intersection 		

반환 배열의 길이와 순서

`cubes1` 의 순서를 따르지만 **더 짧을 수 있음.**

다음 좌표는 결과가 반드시 공집합이므로 배열에서 아예 빠짐.

- 대응하는 `cubes2` 큐브가 없는 좌표
- 어느 한쪽 큐브의 복셀이 모두 0 인 좌표

반면 양쪽 다 복셀이 있는데 겹치는 복셀이 없어서 결과가 빈 큐브가 된 좌표는 **남아 있음.**

따라서 반환 배열의 인덱스를 `cubes1` 의 인덱스와 그대로 대응시킬 수 없으며, 좌표로 다시 찾아야 함.

겹치는 좌표가 하나도 없으면 **길이 0 인 배열**을 돌려줌(`null` 이 아님).

모든 원소는 새로 만든 큐브이며, 대응하는 `cubes1` 큐브의 헤더를 승계함.

예제

시험 시나리오 TS-004-INT(토러스와 3D 십자)의 입력 파일이 있는 폴더에서 실행.

```
TS-004-INT>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-004-INT>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
```

```
import com.voxel.operator.VoxelOpIntersection;

public class Example {

    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes1 = readZip("cubes1.zip");
        BasicDataCube[] cubes2 = readZip("cubes2.zip");
        BasicDataCube[] result = VoxelOpIntersection.intersection(cubes1, cubes2);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }

    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }

    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }

}
```

Parameters:

`cubes1` - 첫 번째 객체를 담은 큐브 리스트.

bit 타입이어야 하고, 원소가 하나 이상이어야 하며, 모든 큐브의 level 과 resolution 이 같아야 하고, `(col, row, floor)` 좌표가 서로 달라야 함

`cubes2` - 두 번째 객체를 담은 큐브 리스트.

조건은 `cubes1` 과 같음.

`null` 또는 빈 배열을 넘기면 길이 0 인 배열을 돌려줌

Returns:

연산 결과 큐브 리스트 (bit).

`cubes1` 과 같거나 더 짧음

Throws:

`IllegalArgumentException` - `cubes1` 이 `null` 이거나 비었을 때, 어느 한쪽 리스트에 `null` 원소가 있을 때, bit 가 아닌 큐브가 섞였을 때, 한 리스트 안에서 또는 두 리스트 사이에서 level 이나 resolution 이 서로 다를 때, 한 리스트 안에 `(col, row, floor)`가 중복될 때

5. VoxelOpDifference [벡터]

클래스 설명

```
public final class VoxelOpDifference
    extends Object
```

[벡터] 차집합 연산자.

큐브 리스트의 객체에서 마스크와 겹치는 부분을 잘라냄.

진입점은 정적 메서드 `difference(BasicDataCube[], BasicDataCube[])` 하나이며 인스턴스를 만들 수 없음.

상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

같은 `(col, row, floor)` 좌표끼리 짝지어 복셀 단위로 `cubes AND NOT masks` 를 계산함.

입력 큐브는 변경하지 않음.

지원 타입

입력과 반환 모두 bit(`BasicDataCube.DataType.BIT`).

다른 타입이 섞이면 `IllegalArgumentException` 을 던짐.

해상도는 `BasicDataCube.Resolution.R64`, `BasicDataCube.Resolution.R128` 둘 다 지원하며, 두 리스트의 모든 큐브가 같은 level 과 resolution 을 가져야 함.

센티넬

반환 큐브는 입력과 상관없이 언제나 데이터 블록 코드 `'V'`, 센티넬 0 임.

`'S'` 입력의 센티넬 자리는 0 으로 보고 계산함.

Method Summary

Modifier and Type	Method	Description
`static BasicDataCube[]`	`difference(BasicDataCube[] cubes, BasicDataCube[] masks)`	cubes − masks.

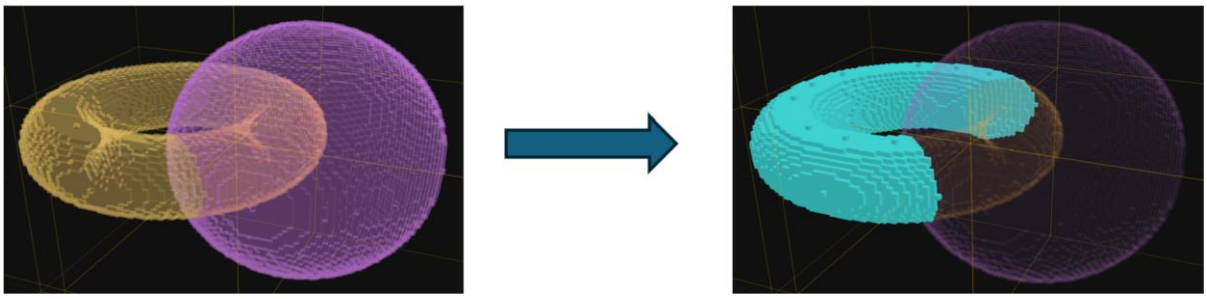
5.1. difference API

```
public static BasicDataCube[] difference(BasicDataCube[] cubes,
    BasicDataCube[] masks)
```

cubes − masks.

difference

메서드	파라미터 / 반환	설명
difference	(	큐브내 객체 기준으로 마스크영역을 잘라내어 차집합 부분을 추출하는 연산. (cubes, masks 는 같은 공간크기(level), 같은 해상도(resolution)를

		가짐)
	BasicDataCube[] cubes	기준이되는 객체를 포함하는 큐브 리스트. 지원타입: bit
	BasicDataCube[] masks)	마스크영역에 해당하는 큐브 리스트. 지원타입: bit
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. (dataBlockCode='V', sentinel=0) 반환타입: bit
difference 		

반환 배열의 길이와 순서

`cubes` 와 길이·순서·좌표가 모두 같음.

결과가 빈 큐브가 되어도 배열에서 빠지지 않으므로, 반환 배열의 인덱스를 `cubes` 의 인덱스에 그대로 대응시킬 수 있음.

각 원소는 대응하는 `cubes` 큐브의 헤더를 승계함.

반환 배열에 입력 큐브가 그대로 섞임

대응하는 `masks` 큐브가 없거나 그 복셀이 모두 0 인 좌표는 잘라낼 것이 없으므로 계산 없이 **입력 큐브 객체를 그대로** 결과에 넣음 — 그 입력이 `'V'`일 때만.

`'S'` 입력은 센티넬을 0 으로 바꾼 새 `'V'` 큐브가 들어감.

마스크가 실제로 겹친 좌표는 새로 만든 큐브임.

예제

시험 시나리오 TS-010-DIF(십자에서 가운데 구 빼기)의 입력 파일이 있는 폴더에서 실행.

```
TS-010-DIF>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-010-DIF>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
```

```
import com.voxel.operator.VoxelOpDifference;

public class Example {

    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] masks = readZip("masks.zip");
        BasicDataCube[] result = VoxelOpDifference.difference(cubes, masks);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }

    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }

    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }

}
```

Parameters:

`cubes` - 기준이 되는 객체를 담은 큐브 리스트.

bit 타입이어야 하고, 원소가 하나 이상이어야 하며, 모든 큐브의 level 과 resolution 이 같아야 하고, `(col, row, floor)` 좌표가 서로 달라야 함

`masks` - 잘라낼 영역을 담은 큐브 리스트.

조건은 `cubes` 와 같음.

`null` 또는 빈 배열을 넘기면 `cubes` 를 그대로 돌려줌

Returns:

연산 결과 큐브 리스트 (bit).

`cubes` 와 길이가 같음

Throws:

`IllegalArgumentException` - `cubes` 가 `null` 이거나 비었을 때, 어느 한쪽 리스트에 `null` 원소가 있을 때, bit 가 아닌 큐브가 섞였을 때, 한 리스트 안에서 또는 두 리스트 사이에서 level 이나 resolution 이 서로 다를 때, 한 리스트 안에 `(col, row, floor)`가 중복될 때

6. VoxelOpUnion [벡터]

클래스 설명

```
public final class VoxelOpUnion
    extends Object
```

[벡터] 합집합 연산자.

두 큐브 리스트의 객체를 합친 결과를 돌려줌.

진입점은 정적 메서드 `union(BasicDataCube[], BasicDataCube[])` 하나이며 인스턴스를 만들 수 없음.

상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

같은 `(col, row, floor)` 좌표끼리 짝지어 복셀 단위 OR 연산을 함.

좌표가 다른 큐브끼리는 서로 영향을 주지 않음.

입력 큐브는 변경하지 않음.

지원 타입

입력과 반환 모두 bit(`BasicDataCube.DataType.BIT`).

다른 타입이 섞이면 `IllegalArgumentException` 을 던짐.

해상도는 `BasicDataCube.Resolution.R64`, `BasicDataCube.Resolution.R128` 둘 다 지원하며, 두 리스트의 모든 큐브가 같은 level 과 resolution 을 가져야 함.

센티넬

반환 큐브는 입력과 상관없이 언제나 데이터 블록 코드 `'V'`, 센티넬 0 임.

입력이 `'S'`이면 센티넬 자리를 0(`false`)으로 보고 계산함 — 센티넬이 `false` 이면 비트값 그대로이고, `true` 이면 켜진 자리가 모두 센티넬이라 전부 0 으로 봄.

Method Summary

Modifier and Type	Method	Description
`static BasicDataCube[]`	`union(BasicDataCube[] cubes1, BasicDataCube[] cubes2)`	cubes1 ∪ cubes2.

6.1. union API

```
public static BasicDataCube[] union(BasicDataCube[] cubes1,
    BasicDataCube[] cubes2)
```

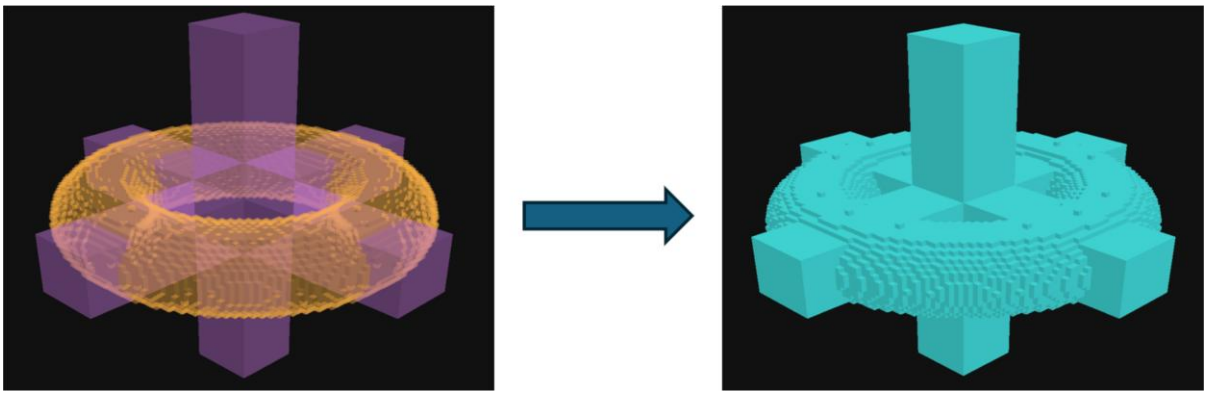
cubes1 ∪ cubes2.

union

메서드	파라미터 / 반환	설명
union	(	첫번째 큐브내 객체와 두번째 큐브내 객체의 합집합 부분을

		추출하는 연산. (cubes1, cubes2 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes1	첫번째 객체를 포함하는 큐브 리스트. 지원타입: bit
	BasicDataCube[] cubes2)	두번째 객체를 포함하는 큐브 리스트. 지원타입: bit
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. (dataBlockCode='V', sentinel=0) 반환타입: bit

union



반환 배열의 길이와 순서

`cubes1` 보다 길어질 수 있음.

`cubes1` 의 순서를 그대로 둔 뒤, `cubes2` 에만 있는 좌표의 큐브를 `cubes2` 의 순서대로 이어 붙임.

```
cubes1 = [A, B]      (좌표 A, B)
cubes2 = [B, C]      (좌표 B, C)
결과   = [A, B∪B, C]  (길이 3)
```

양쪽에 다 있는 좌표의 결과 큐브는 `cubes1` 큐브의 헤더(level, col, row, floor, datatype, resolution)를 승계하고, `cubes2` 에만 있는 좌표는 `cubes2` 큐브의 것을 따름.

데이터 블록 코드는 언제나 `'V'`, 센티넬은 0 임 (클래스 설명의 <센티넬>).

반환 배열에 입력 큐브가 그대로 섞임

합칠 상대가 없거나 상대 큐브의 복셀이 모두 0 인 좌표는 계산 없이 **입력 큐브 객체를 그대로** 결과에 넣음 — 그 입력이 `'V'`일 때만.

`'S'` 입력은 센티넬을 0 으로 바꾼 새 `'V'` 큐브가 들어감.

즉 반환 배열의 일부 원소는 인자로 넘긴 객체와 동일할 수 있음 (배열 자체는 새로 만든 것이라 배열을

고쳐도 입력 배열에는 영향이 없음).

그 외의 좌표는 새로 만든 큐브임.

예제

시험 시나리오 TS-008-UNI(위아래로 나뉜 구 합치기)의 입력 파일이 있는 폴더에서 실행.

```
TS-008-UNI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-008-UNI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpUnion;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes1 = readZip("cubes1.zip");
        BasicDataCube[] cubes2 = readZip("cubes2.zip");
        BasicDataCube[] result = VoxelOpUnion.union(cubes1, cubes2);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes1` - 첫 번째 객체를 담은 큐브 리스트.

bit 타입이어야 하고, 원소가 하나 이상이어야 하며, 모든 큐브의 level 과 resolution 이 같아야 하고, `(col, row, floor)` 좌표가 서로 달라야 함

`cubes2` - 두 번째 객체를 담은 큐브 리스트.

조건은 `cubes1` 과 같음.

`null` 또는 빈 배열을 넘기면 `cubes1` 을 그대로 돌려줌

Returns:

연산 결과 큐브 리스트 (bit).

`cubes1` 과 같거나 더 깊

Throws:

`IllegalArgumentException` - `cubes1` 이 `null` 이거나 비었을 때, 어느 한쪽 리스트에 `null` 원소가 있을 때, bit 가 아닌 큐브가 섞였을 때, 한 리스트 안에서 또는 두 리스트 사이에서 level 이나 resolution 이 서로 다를 때, 한 리스트 안에 `(col, row, floor)`가 중복될 때

7. VoxelOpBuffer [벡터]

클래스 설명

```
public final class VoxelOpBuffer
    extends Object
```

[벡터] 버퍼 연산자.

큐브 리스트 안의 객체를 이웃 정의와 거리만큼 확장하거나 축소함.

진입점은 정적 메서드 `buffer(BasicDataCube[], Neighbor, int)` 하나이며 인스턴스를 만들 수 없음.

상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

복셀 단위 연산이며 큐브 경계를 넘어 계산함.

하나의 객체가 여러 큐브에 걸쳐 잘려 있어도 이어진 것으로 보고 처리하므로, 서로 인접한 큐브는 **한 번의 호출에 함께 넘겨야** 경계에서 결과가 끊기지 않음.

연산 정의

`distance > 0`	확장. 자기 자신 또는 거리 안의 이웃 중 하나라도 `1` 이면 결과가 `1`
`distance = 0`	그대로
`distance < 0`	축소. 자기가 `1` 이고 거리 안의 이웃이 모두 `1` 이면 `1`, 하나라도 `0` 이면 `0`

거리 판정에 쓰는 이웃 집합은 `d = |distance|`에 대해 다음과 같음 (자기 자신 제외).

`Neighbor.N6`	`|dx| + |dy| + |dz| <= d` (맨해튼)
`Neighbor.N18`	`|dx| + |dy| + |dz| <= 2d` 이고 각 축이 `d` 이내
`Neighbor.N26`	`max(|dx|, |dy|, |dz|) <= d` (체비셰프)

반환 배열

확장에서는 입력보다 길어질 수 있음.

큐브 경계에 붙은 복셀의 버퍼가 옆 큐브로 넘어가는데 그 좌표의 큐브가 입력에 없으면, 해당 좌표에 빈 큐브를 새로 만들어 결과에 이어 붙임.

순서는 입력 좌표가 입력 순서대로 먼저 오고, 새로 생긴 좌표가 뒤에 붙음.

새 큐브는 level, datatype, resolution 을 입력 큐브에서 승계하며 col, row, floor 만 넘어간 위치의 값을 가짐.

축소에서는 입력과 길이·순서·좌표가 모두 같음.

결과가 빈 큐브가 되어도 그 자리는 남음.

좌표계 바깥의 취급

입력 리스트에 없는 좌표는 **복셀이 모두 0 인 것으로 봄.**

따라서 축소에서 리스트 가장자리 큐브의 바깥면에 닿은 복셀은 제거됨.

지원 타입

입력과 반환 모두 bit(`BasicDataCube.DataType.BIT`).

해상도는 `BasicDataCube.Resolution.R64`, `BasicDataCube.Resolution.R128` 둘 다 지원하며, 한 번의 호출에 넘기는 큐브는 level 과 resolution 이 모두 같아야 함.

센티넬

반환 큐브는 입력과 상관없이 언제나 데이터 블록 코드 `'V'`, 센티넬 0 임.

`'S'` 입력의 센티넬 자리는 0 으로 보고 계산함.

입력 큐브는 변경하지 않음.

Method Summary

Modifier and Type	Method	Description
`static BasicDataCube[]`	`buffer(BasicDataCube[] cubes, Neighbor neighbor, int distance)`	버퍼 연산.

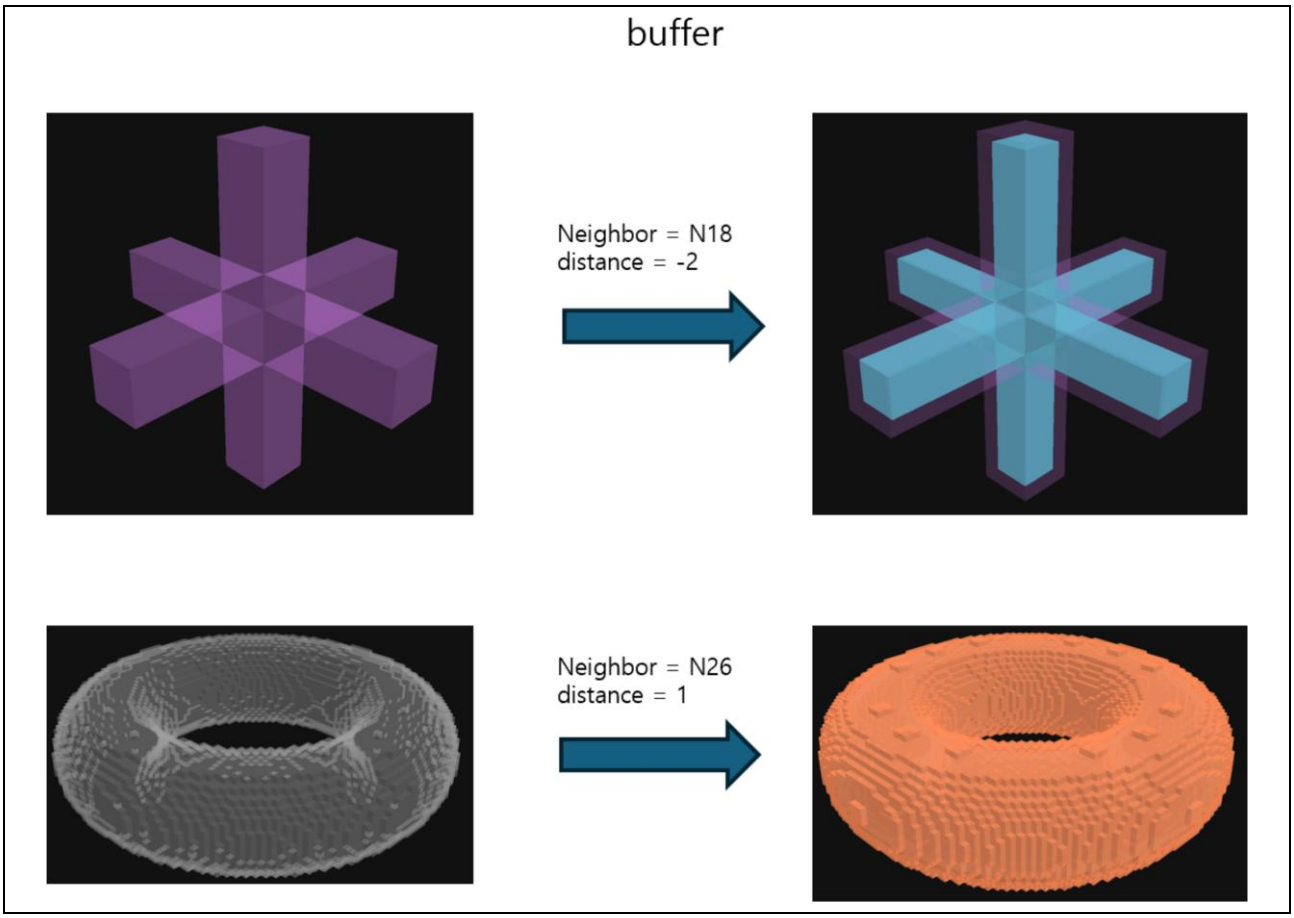
7.1. buffer API

```
public static BasicDataCube[] buffer(BasicDataCube[] cubes,
    Neighbor neighbor,
    int distance)
```

버퍼 연산.

buffer

메서드	파라미터 / 반환	설명
buffer	(	큐브내 객체 기준으로 거리만큼 이웃선택하여 버퍼링하는 연산. (cubes 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes	기준이되는 객체를 포함하는 큐브 리스트. 지원타입: bit
	Neighbor neighbor	이웃선택 방법: N6, N18, N26 중 하나.
	int distance)	이웃선택 거리: ...,-3,-2,-1,1,2,3,... 중 하나. (복셀단위) 양의거리: 확장 버퍼링 음의거리: 축소 버퍼링 (해당복셀에서 검색된 이웃복셀들이 모두 1 이면 남김) (해당복셀에서 검색된 이웃복셀들중 하나라도 0 이면 제거)
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. (dataBlockCode='V', sentinel=0) 반환타입: bit



`distance` 가 0 이면 계산 없이 입력 큐브 객체를 그대로 담은 새 배열을 돌려줌.

`distance` 가 0 이 아니면 결과 큐브를 새로 만듦.

단, 축소에서 입력이 이미 빈 큐브인 자리는 입력 인스턴스를 그대로 돌려줌.

예제

시험 시나리오 TS-013-BUF(나선관 면 이웃 5 칸 확장)의 입력 파일이 있는 폴더에서 실행.

```
TS-013-BUF>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-013-BUF>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.Neighbor;
import com.voxel.operator.VoxelOpBuffer;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpBuffer.buffer(cubes, Neighbor.N6, 5);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
```

```
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - 객체를 담은 큐브 리스트.

bit 타입이어야 하고, 원소가 하나 이상이어야 하며, 모든 큐브의 level 과 resolution 이 같아야 하고, `(col, row, floor)` 좌표가 서로 달라야 함

`neighbor` - 이웃복셀 정의.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

`distance` - 복셀이동거리.

양수는 확장, `0` 은 그대로, 음수는 축소

Returns:

연산 결과 큐브 리스트 (bit).

확장이면 입력보다 길어질 수 있고, 그 외에는 입력과 길이가 같음

Throws:

`IllegalArgumentException` - `cubes` 가 `null` 이거나 비었을 때, `cubes` 에 `null` 원소가 있을 때, bit 가 아닌 큐브가 섞였을 때, 한 리스트 안에서 level 또는 resolution 이 서로 다를 때, `(col, row, floor)`가 중복될 때

`NullPointerException` - `neighbor` 가 `null` 일 때

8. VoxelOpSurface [벡터]

클래스 설명

```
public final class VoxelOpSurface
    extends Object
```

[벡터] 경계추출 연산자.

객체의 표면에 해당하는 복셀만 남김.

진입점은 정적 메서드 `surface(BasicDataCube[], Neighbor)` 하나이며 인스턴스를 만들 수 없음.

상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

복셀 단위 연산이며 큐브 경계를 넘어 계산함.

하나의 객체가 여러 큐브에 걸쳐 잘려 있어도 이어진 것으로 보고 처리하므로, 서로 인접한 큐브는 **한 번의 호출에 함께 넘겨야** 큐브 경계면이 경계로 잘못 잡히지 않음.

연산 정의

값이 1 인 복셀 중 **이웃 중 하나라도 0 인 것**만 결과에 남김.

이웃이 모두 1 인 복셀은 객체 내부이므로 0 이 됨.

이웃 판정에 쓰는 집합은 거리 1 고정이며 이웃 정의에 따라 다음과 같음 (자기 자신 제외).

```
Neighbor.N6    면으로 맞닿은 6 개        |dx| + |dy| + |dz| <= 1
Neighbor.N18   면 + 모서리 18 개         |dx| + |dy| + |dz| <= 2
Neighbor.N26   면 + 모서리 + 꼭짓점 26 개
```

반환 배열

입력과 길이·순서·좌표가 모두 같음.

결과가 빈 큐브가 되어도 그 자리는 남으므로, 반환 배열의 인덱스를 입력 인덱스에 그대로 대응시킬 수 있음.

각 원소는 대응하는 입력 큐브의 헤더를 승계함.

복셀이 모두 0 인 큐브는 경계도 없으므로 계산 없이 **입력 큐브 객체를 그대로** 돌려줌 — 그 입력이 `'V'`일 때만.

`'S'` 입력은 센티넬을 0 으로 바꾼 새 `'V'` 큐브가 들어감.

좌표계 바깥의 취급

입력 리스트에 없는 좌표는 **복셀이 모두 0 인 것으로 봄.**

따라서 리스트 가장자리 큐브의 바깥면에 닿은 복셀은 경계로 판정됨.

지원 타입

입력과 반환 모두 bit(`BasicDataCube.DataType.BIT`).

해상도는 `BasicDataCube.Resolution.R64`, `BasicDataCube.Resolution.R128` 둘 다 지원하며, 한 번의 호출에 넘기는 큐브는 level 과 resolution 이 모두 같아야 함.

센티넬

반환 큐브는 입력과 상관없이 언제나 데이터 블록 코드 `'V'`, 센티넬 0 임.

`'S'` 입력의 센티넬 자리는 0 으로 보고 계산함.

입력 큐브는 변경하지 않음.

Method Summary

Modifier and Type	Method	Description
`static BasicDataCube[]`	`surface(BasicDataCube[] cubes, Neighbor neighbor)`	경계추출 연산.

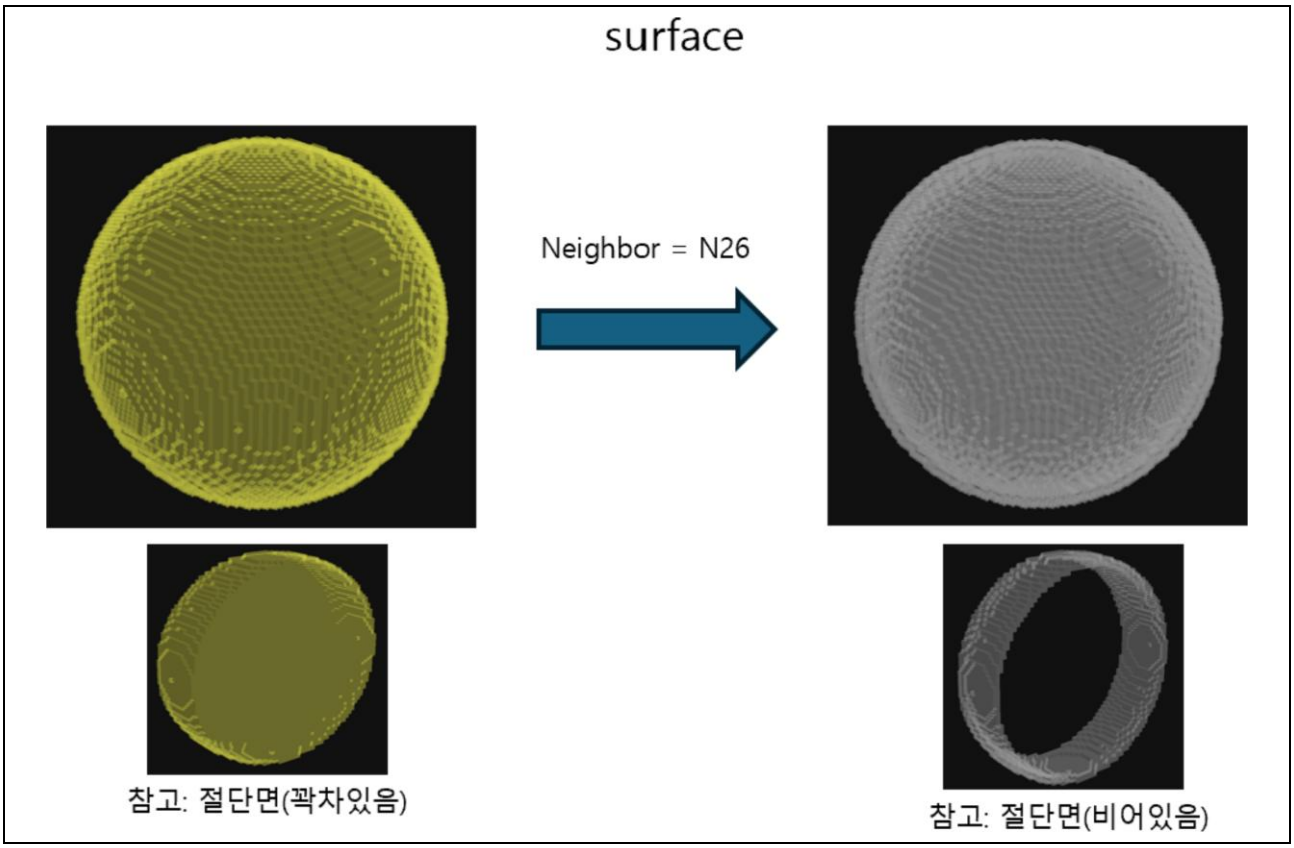
8.1. surface API

```
public static BasicDataCube[] surface(BasicDataCube[] cubes,
    Neighbor neighbor)
```

경계추출 연산.

surface

메서드	파라미터 / 반환	설명
surface	(	큐브내 객체 기준으로 이웃선택 기반 경계(표면)추출하는 연산. (cubes 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes	기준이되는 객체를 포함하는 큐브 리스트. 지원타입: bit
	Neighbor neighbor)	이웃선택 방법: N6, N18, N26 중 하나. (해당복셀에서 검색된 이웃복셀들중 하나라도 0 이면 경계)
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. (dataBlockCode='V', sentinel=0) 반환타입: bit



예제

시험 시나리오 TS-016-SUR(토러스 껍질)의 입력 파일이 있는 폴더에서 실행.

```
TS-016-SUR>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-016-SUR>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.Neighbor;
import com.voxel.operator.VoxelOpSurface;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpSurface.surface(cubes, Neighbor.N6);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
```

```
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }

}
```

Parameters:

`cubes` - 객체를 담은 큐브 리스트.

bit 타입이어야 하고, 원소가 하나 이상이어야 하며, 모든 큐브의 level 과 resolution 이 같아야 하고, `(col, row, floor)` 좌표가 서로 달라야 함

`neighbor` - 이웃복셀 정의.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

Returns:

경계 복셀만 남은 큐브 리스트 (bit).

`cubes` 와 길이·순서·좌표가 같음

Throws:

`IllegalArgumentException` - `cubes` 가 `null` 이거나 비었을 때, `cubes` 에 `null` 원소가 있을 때, bit 가 아닌 큐브가 섞였을 때, 한 리스트 안에서 level 또는 resolution 이 서로 다를 때, `(col, row, floor)`가 중복될 때

`NullPointerException` - `neighbor` 가 `null` 일 때

9. VoxelOpFill [벡터]

클래스 설명

```
public final class VoxelOpFill
    extends Object
```

[벡터] 내부채움 연산자.

객체 내부의 빈 공간을 1 로 채움.

인스턴스를 만들 수 없으며 채우는 방법에 따라 정적 메서드 네 개를 제공함.

- `fillNeighborFlood(BasicDataCube[], Neighbor)` — 바깥에서 홍수처럼 번져 닿지 않는 곳을 내부로 판정
- `fillRayCasting(BasicDataCube[], int[])` — 레이가 지나온 벽 개수의 홀짝으로 판정
- `fillDEM(BasicDataCube[])` — 지형 표면 아래를 모두 채움
- `fillCircle(BasicDataCube[], int[])` — 평면 단면마다 닫힌 영역을 내부로 판정

모두 상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

복셀 단위 연산이며 큐브 경계를 넘어 계산함.
하나의 객체가 여러 큐브에 걸쳐 잘려 있어도 이어진 것으로 보고 처리하므로, 서로 인접한 큐브는 **한 번의 호출에 함께 넘겨야** 내부 판정이 어긋나지 않음.

반환 배열

네 메서드 모두 **입력과 길이·순서·좌표가 같음.**
각 원소는 대응하는 입력 큐브의 헤더를 승계함.

채울 것이 하나도 없는 큐브는 **입력 큐브 객체를 그대로** 돌려줌 — 그 입력이 `'V'`일 때만.
`'S'` 입력은 센티넬을 0 으로 바꾼 새 `'V'` 큐브가 들어감.

좌표계 바깥의 취급

입력 리스트에 없는 좌표는 **복셀이 모두 0 인 바깥**으로 봄.
리스트 가장자리뿐 아니라 리스트 중간에 빠진 좌표도 마찬가지라, 그 틈으로 바깥이 이어짐.
채움은 입력에 있는 좌표에만 일어남.

지원 타입

입력과 반환 모두 bit(`BasicDataCube.DataType.BIT`).
해상도는 `BasicDataCube.Resolution.R64`, `BasicDataCube.Resolution.R128` 둘 다 지원하며, 한 번의 호출에 넘기는 큐브는 level 과 resolution 이 모두 같아야 함.

센티넬

반환 큐브는 입력과 상관없이 언제나 데이터 블록 코드 `'V'`, 센티넬 0 임.

`'S'` 입력의 센티넬 자리는 0 으로 보고 계산함.

입력 큐브는 변경하지 않음.

Field Summary

Fields

Modifier and Type	Field	Description
`static final int`	`CIRCLE_XY`	`fillCircle(com.voxel.operator.BasicDataCube[], int[])`의 기준면 — `XY` 평면을 `+Z`(위쪽)로 옮겨 가며 닫힌 영역을 채움.
`static final int`	`CIRCLE_YZ`	`fillCircle(com.voxel.operator.BasicDataCube[], int[])`의 기준면 — `YZ` 평면을 `+X`(동쪽)로 옮겨 가며 닫힌 영역을 채움.
`static final int`	`CIRCLE_ZX`	`fillCircle(com.voxel.operator.BasicDataCube[], int[])`의 기준면 — `ZX` 평면을 `+Y`(북쪽)로 옮겨 가며 닫힌 영역을 채움.
`static final int`	`RAY_X`	`fillRayCasting(com.voxel.operator.BasicDataCube[], int[])`의 방향값 — `+X`(동쪽)로 레이캐스팅.
`static final int`	`RAY_Y`	`fillRayCasting(com.voxel.operator.BasicDataCube[], int[])`의 방향값 — `+Y`(북쪽)로 레이캐스팅.
`static final int`	`RAY_Z`	`fillRayCasting(com.voxel.operator.BasicDataCube[], int[])`의 방향값 — `+Z`(위쪽)로 레이캐스팅.

Method Summary

Modifier and Type	Method	Description
`static BasicDataCube[]`	`fillCircle(BasicDataCube[] cubes, int[] circleFace)`	평면 단면마다 닫힌 영역을 찾아 내부를 채움.
`static BasicDataCube[]`	`fillDEM(BasicDataCube[] cubes)`	DEM 표면 아래를 채움.
`static BasicDataCube[]`	`fillNeighborFlood(BasicDataCube[] cubes, Neighbor neighbor)`	홍수채우기로 내부를 채움.
`static BasicDataCube[]`	`fillRayCasting(BasicDataCube[] cubes, int[] rayDirection)`	레이캐스팅으로 내부를 채움.

Field Details

RAY_X

```
public static final int RAY_X
```

`fillRayCasting(com.voxel.operator.BasicDataCube[], int[])`의 방향값 — `+X`(동쪽)로 레이캐스팅.

값은 `1`.

See Also:

- Constant Field Values

RAY_Y

```
public static final int RAY_Y
```

`fillRayCasting(com.voxel.operator.BasicDataCube[], int[])`의 방향값 — `+Y`(북쪽)로 레이캐스팅.

값은 `2`.

See Also:

- Constant Field Values

RAY_Z

```
public static final int RAY_Z
```

`fillRayCasting(com.voxel.operator.BasicDataCube[], int[])`의 방향값 — `+Z`(위쪽)로 레이캐스팅.

값은 `3`.

See Also:

- Constant Field Values

CIRCLE_XY

```
public static final int CIRCLE_XY
```

`fillCircle(com.voxel.operator.BasicDataCube[], int[])`의 기준면 — `XY` 평면을 `+Z`(위쪽)로 옮겨 가며 닫힌 영역을 채움.

값은 `1`.

z 좌표가 같은 복셀끼리 한 단면을 이룸.

수평 단면이므로 원기둥 모양 객체의 원 단면이 채워짐.

See Also:

- Constant Field Values

CIRCLE_YZ

```
public static final int CIRCLE_YZ
```

`fillCircle(com.voxel.operator.BasicDataCube[], int[])`의 기준면 — `YZ` 평면을 `+X`(동쪽)로 옮겨 가며 닫힌 영역을 채움.

값은 `2`.

x 좌표가 같은 복셀끼리 한 단면을 이룸.

See Also:

- Constant Field Values

CIRCLE_ZX

```
public static final int CIRCLE_ZX
```

`fillCircle(com.voxel.operator.BasicDataCube[], int[])`의 기준면 — ZX 평면을 +Y(북쪽)로 옮겨 가며 닫힌 영역을 채움.

값은 3.

y 좌표가 같은 복셀끼리 한 단면을 이룸.

See Also:

- Constant Field Values

9.1. fillNeighborFlood API

```
public static BasicDataCube[] fillNeighborFlood(BasicDataCube[] cubes,
    Neighbor neighbor)
```

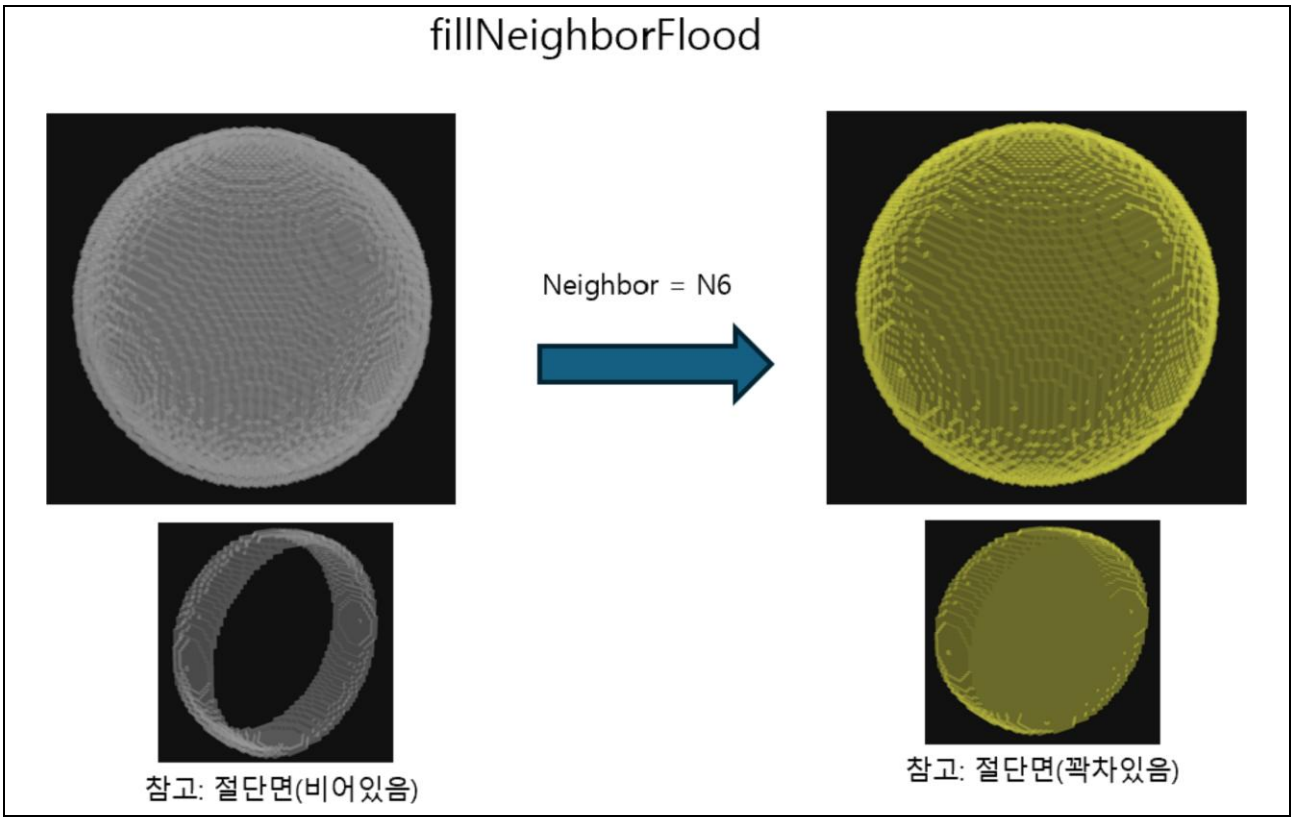
홍수채우기로 내부를 채움.

좌표계 바깥에 닿아 있는 0 복셀에서 출발해 0 복셀만 타고 번지며, 끝까지 닿지 않은 0 복셀을 내부로 판정해 1 로 바꿈.

`neighbor` 는 **0 복셀끼리 이어져 있다고 볼 기준**임.
넓게 잡을수록 바깥이 더 멀리 번져 내부로 남는 곳이 줄어듦 (대각선으로만 벌어진 틈은 `Neighbor.N6` 에서는 막혀 있고 `Neighbor.N26` 에서는 뚫림).

fillNeighborFlood

메서드	파라미터 / 반환	설명
fillNeighborFlood	(	큐브내 객체 기준으로 이웃선택 기반 Flood(홍수채우기)방법으로 내부 채움 연산. (cubes 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes	기준이되는 객체를 포함하는 큐브 리스트. 지원타입: bit
	Neighbor neighbor)	이웃선택 방법: N6, N18, N26 중 하나. (여러큐브에 분할되어 있는 객체는 연산불가 -> 하나의 객체로 병합) (큐브 최외곽 0 인셀로부터 외부와의 연결성 추적) (외부와 연결되지 않은 나머지 셀들을 내부로 판단)
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. (dataBlockCode='V', sentinel=0) 반환타입: bit



예제

시험 시나리오 TS-019-FIL(구 껍질 채움)의 입력 파일이 있는 폴더에서 실행.

```
TS-019-FIL>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-019-FIL>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.Neighbor;
import com.voxel.operator.VoxelOpFill;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpFill.fillNeighborFlood(cubes, Neighbor.N6);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
```

```
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - 객체를 담은 큐브 리스트.

bit 타입이어야 하고, 원소가 하나 이상이어야 하며, 모든 큐브의 level 과 resolution 이 같아야 하고, `(col, row, floor)` 좌표가 서로 달라야 함

`neighbor` - 0 복셀의 연결 판정 기준.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

Returns:

내부가 채워진 큐브 리스트 (bit).

`cubes` 와 길이·순서·좌표가 같음

Throws:

`IllegalArgumentException` - `cubes` 가 `null` 이거나 비었을 때, `cubes` 에 `null` 원소가 있을 때, bit 가 아닌 큐브가 섞였을 때, 한 리스트 안에서 level 또는 resolution 이 서로 다를 때, `(col, row, floor)`가 중복될 때

`NullPointerException` - `neighbor` 가 `null` 일 때

9.2. fillRayCasting API

```
public static BasicDataCube[] fillRayCasting(BasicDataCube[] cubes,
    int[] rayDirection)
```

레이캐스팅으로 내부를 채움.

축 하나를 따라 좌표계를 가로지르는 직선마다, 지나온 벽체(1 이 이어진 덩어리) 개수를 셈.

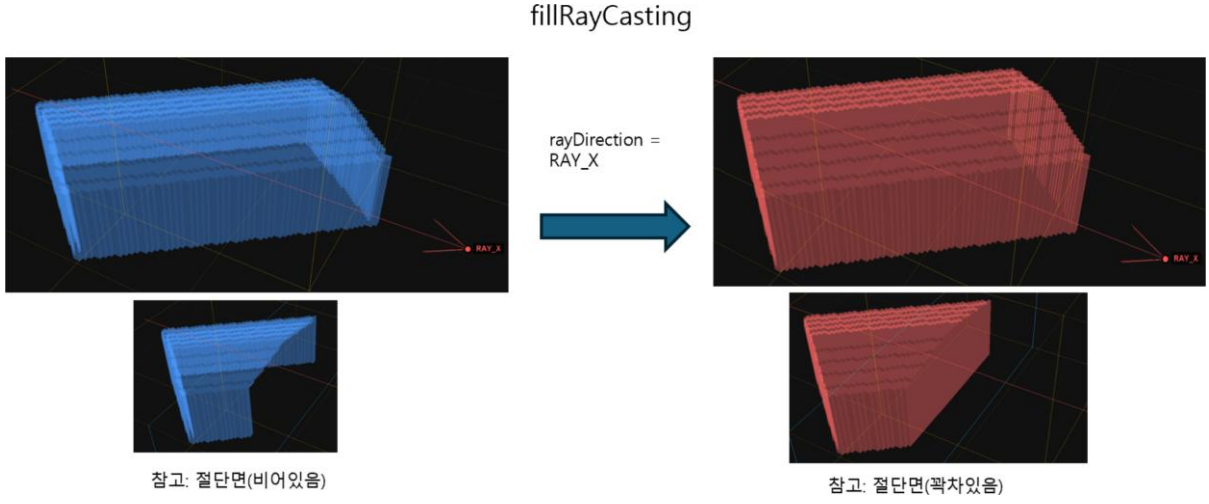
개수가 홀수인 지점의 0 복셀을 내부로 판정해 1 로 바꿈.

직선에서 **마지막 벽체보다 뒤에 있는 0 은 채우지 않음.**

그래서 한쪽 벽이 잘려 나간 채로 들어온 객체는 채우지 않음.

fillRayCasting

메서드	파라미터 / 반환	설명
fillRayCasting	(	큐브내 객체 기준으로 레이캐스팅(RayCasting)방법으로 내부 채움 연산. (cubes 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes	기준이되는 객체를 포함하는 큐브 리스트. 지원타입: bit
	int[] rayDirection)	레이캐스팅 방향(+X,+Y,+Z) 및 순서 RAY_X, RAY_Y, RAY_Z 방향중 1~3 개로 구성된 배열 (순서중요) RAY_X: +X 방향으로 레이캐스팅하여 채움 RAY_Y: +Y 방향으로 레이캐스팅하여 채움 RAY_Z: +Z 방향으로 레이캐스팅하여 채움 (여러큐브에 분할되어 있는 객체는 연산불가 -> 하나의 객체로 병합) (해당복셀의 레이선상에 벽체가 짝수개 이면 외부, 홀수개 이면 내부로 판단)
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. (dataBlockCode='V', sentinel=0) 반환타입: bit



`rayDirection` 에 방향을 여러 개 담으면 **배열 순서대로 차례로 적용**하며, 뒤 방향은 앞 방향이 채운 결과 위에서 계산함.

따라서 순서가 다르면 결과도 다를 수 있음.

```
// +Z 한 방향만
VoxelOpFill.fillRayCasting(cubes, new int[]{ VoxelOpFill.RAY_Z });


// +X, +Y, +Z 순서로 세 번
VoxelOpFill.fillRayCasting(cubes, new int[]{
        VoxelOpFill.RAY_X, VoxelOpFill.RAY_Y, VoxelOpFill.RAY_Z });
```

예제

시험 시나리오 TS-022-FIL(바닥뚫린 건물 X 방향)의 입력 파일이 있는 폴더에서 실행.

```
TS-022-FIL>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-022-FIL>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpFill;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        int[] rayDirection = {VoxelOpFill.RAY_X};
        BasicDataCube[] result = VoxelOpFill.fillRayCasting(cubes, rayDirection);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - 객체를 담은 큐브 리스트.

bit 타입이어야 하고, 원소가 하나 이상이어야 하며, 모든 큐브의 level 과 resolution 이 같아야 하고, `(col, row, floor)` 좌표가 서로 달라야 함

`rayDirection` - 레이 방향과 적용 순서.

`RAY_X`, `RAY_Y`, `RAY_Z` 중 1 개에서 3 개로 구성된 배열.

같은 방향을 두 번 담을 수 없음

Returns:

내부가 채워진 큐브 리스트 (bit).

`cubes` 와 길이·순서·좌표가 같음

Throws:

`IllegalArgumentException` - `cubes` 가 `null` 이거나 비었을 때, `cubes` 에 `null` 원소가 있을 때, bit 가 아닌 큐브가 섞였을 때, 한 리스트 안에서 level 또는 resolution 이 서로 다를 때, `(col, row, floor)`가 중복될 때, `rayDirection` 이 `null` 이거나 비었거나 원소가 3 개를 넘을 때, `rayDirection` 에 `RAY_X`, `RAY_Y`, `RAY_Z` 가 아닌 값이 있을 때, `rayDirection` 에 같은 방향이 두 번 있을 때

9.3. fillDEM API

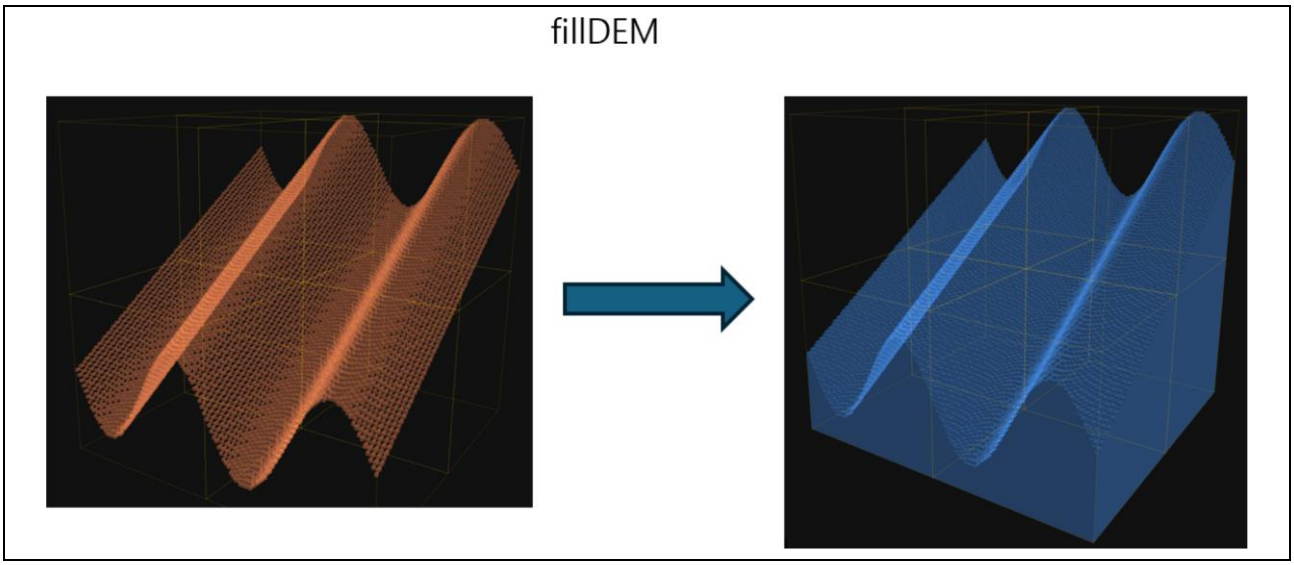
```
public static BasicDataCube[] fillDEM(BasicDataCube[] cubes)
```

DEM 표면 아래를 채움.

수직 기둥마다 위에서 아래로 내려가며, 1 을 처음 만난 지점부터 기둥 바닥까지 모두 1 로 바꿈.
1 을 하나도 만나지 못한 기둥은 그대로 둠.

fillDEM

메서드	파라미터 / 반환	설명
fillDEM	(	DEM 복셀 데이터를 -Z 방향으로 레이캐스팅하여 내부 채움 연산. (cubes 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐) (가능한 한 지형높이가 큐브 바닥과 천장을 벗어나는 곳이 없도록 큐브를 병합) (만약 바닥과 천장을 벗어나는 곳이 있으면 그곳은 채우지 않음) (-Z 방향으로 내려가다 1 인곳을 만나면 그 이후는 모두 1 로 채움)
	BasicDataCube[] cubes)	DEM 복셀 데이터를 포함하는 큐브 리스트. 지원타입: bit
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. (dataBlockCode='V', sentinel=0) 반환타입: bit



예제

시험 시나리오 TS-027-FIL(실데이터)의 입력 파일이 있는 폴더에서 실행.

```
TS-027-FIL>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-027-FIL>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpFill;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpFill.fillDEM(cubes);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
```

```
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }

}
```

Parameters:

`cubes` - DEM 복셀 데이터를 담은 큐브 리스트.

bit 타입이어야 하고, 원소가 하나 이상이어야 하며, 모든 큐브의 level 과 resolution 이 같아야 하고, `(col, row, floor)` 좌표가 서로 달라야 함

Returns:

표면 아래가 채워진 큐브 리스트 (bit).

`cubes` 와 길이·순서·좌표가 같음

Throws:

`IllegalArgumentException` - `cubes` 가 `null` 이거나 비었을 때, `cubes` 에 `null` 원소가 있을 때, bit 가 아닌 큐브가 섞였을 때, 한 리스트 안에서 level 또는 resolution 이 서로 다를 때, `(col, row, floor)`가 중복될 때

9.4. fillCircle API

```
public static BasicDataCube[] fillCircle(BasicDataCube[] cubes,
    int[] circleFace)
```

평면 단면마다 닫힌 영역을 찾아 내부를 채움.

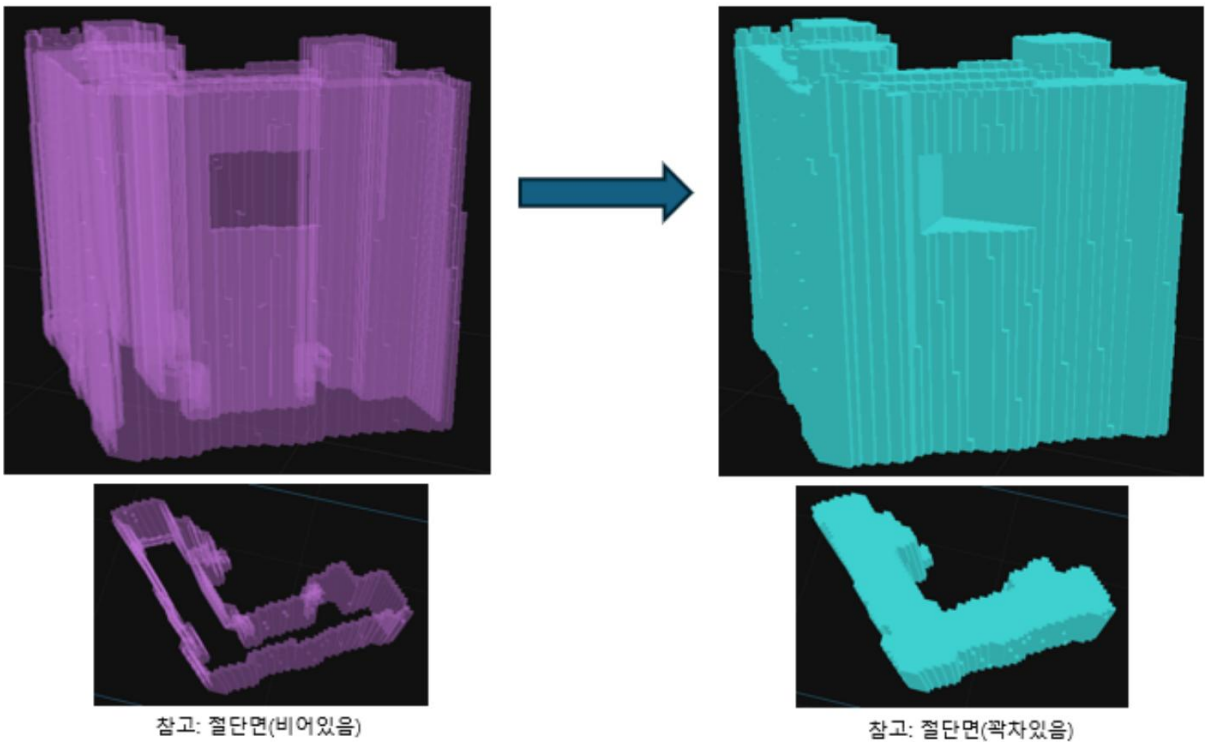
기준면과 평행한 단면을 한 장씩 보며, 단면 안에서 1 로 둘러싸인 0 영역을 내부로 판정해 1 로 바꿈.
단면의 바깥 테두리에 닿아 있는 0 복셀에서 출발해 단면 안의 0 복셀만 타고 번지며, 끝까지 닿지 않은 0 복셀이 닫힌 영역임.
한 단면에 닫힌 영역이 여러 개면 모두 채움.

fillCircle

메서드	파라미터 / 반환	설명
fillCircle	(	큐브내 객체 기준으로 닫힌영역검출(Circle)방법으로 내부 채움 연산. circleFace 를 기준면으로 하여 닫힌영역을 모두 검출하여 내부를 채움. (cubes 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes	기준이되는 객체를 포함하는 큐브 리스트. 지원타입: bit
	int[] circleFace)	닫힌영역을 검출하기 위한 평면(XY,YZ,ZX) 및 순서

		CIRCLE_XY, CIRCLE_YZ, CIRCLE_ZX 평면중 1~3 개로 구성된 배열 (순서중요) CIRCLE_XY: XY 평면을 +Z 방향으로 이동하면서 닫힌영역을 검출하여 채움 CIRCLE_YZ: YZ 평면을 +X 방향으로 이동하면서 닫힌영역을 검출하여 채움 CIRCLE_ZX: ZX 평면을 +Y 방향으로 이동하면서 닫힌영역을 검출하여 채움 (여러큐브에 분할되어 있는 객체는 연산불가 -> 하나의 객체로 병합) (해당 평면에 존재하는 모든 닫힌 영역을 채움)
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. (dataBlockCode='V', sentinel=0) 반환타입: bit

fillCircle



판정은 **단면 하나만 보고** 함.

위아래 단면이 뚫려 있어도 그 단면에서 닫혀 있으면 채우므로, 위아래가 뚫린 원기둥 모양 객체도 `CIRCLE_XY` 로 속이 채워짐.

0 복셀은 단면 안에서 **변을 맞댄 4 방향으로만** 이어진 것으로 봄.

그래서 꼭짓점으로만 이어진 1(계단 모양의 원 둘레 등)도 닫힌 벽으로 봄.

`circleFace` 에 기준면을 여러 개 담으면 **배열 순서대로 차례로 적용**하며, 뒤 기준면은 앞 기준면이 채운 결과 위에서 계산함.

따라서 순서가 다르면 결과도 다를 수 있음.

```
// 수평 단면만 – 원기둥 모양 건물의 속을 채움
VoxelOpFill.fillCircle(cubes, new int[]{ VoxelOpFill.CIRCLE_XY });


// XY, YZ, ZX 순서로 세 번
VoxelOpFill.fillCircle(cubes, new int[]{
        VoxelOpFill.CIRCLE_XY, VoxelOpFill.CIRCLE_YZ, VoxelOpFill.CIRCLE_ZX });
```

예제

시험 시나리오 TS-028-FIL(바닥뚫린 건물 XY 단면)의 입력 파일이 있는 폴더에서 실행.

```
TS-028-FIL>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
```

```
TS-028-FIL>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpFill;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        int[] circleFace = {VoxelOpFill.CIRCLE_XY};
        BasicDataCube[] result = VoxelOpFill.fillCircle(cubes, circleFace);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - 객체를 담은 큐브 리스트.

bit 타입이어야 하고, 원소가 하나 이상이어야 하며, 모든 큐브의 level 과 resolution 이 같아야 하고, `(col, row, floor)` 좌표가 서로 달라야 함

`circleFace` - 닫힌 영역을 찾을 기준면과 적용 순서.

`CIRCLE_XY`, `CIRCLE_YZ`, `CIRCLE_ZX` 중 1 개에서 3 개로 구성된 배열.

같은 기준면을 두 번 담을 수 없음

Returns:

내부가 채워진 큐브 리스트 (bit).

`cubes` 와 길이·순서·좌표가 같음

Throws:

`IllegalArgumentException` - `cubes` 가 `null` 이거나 비었을 때, `cubes` 에 `null` 원소가 있을 때, bit 가 아닌 큐브가 섞였을 때, 한 리스트 안에서 level 또는 resolution 이 서로 다를 때, `(col, row, floor)`가 중복될 때, `circleFace` 가 `null` 이거나 비었거나 원소가 3 개를 넘을 때, `circleFace` 에 `CIRCLE_XY`, `CIRCLE_YZ`, `CIRCLE_ZX` 가 아닌 값이 있을 때, `circleFace` 에 같은 기준면이 두 번 있을 때

10. VoxelOpArithmetic [레스터]

클래스 설명

```
public final class VoxelOpArithmetic
    extends Object
```

[레스터] 산술 연산자.

큐브의 복셀값을 계산하거나 큐브 하나를 하나의 값으로 요약함.

인스턴스를 만들 수 없으며 연산마다 정적 메서드를 제공함.

연산마다 형태가 둘 있음 — **큐브끼리** 계산하는 것과, **큐브에 상수를** 계산하는 것.

```
VoxelOpArithmetic.add(cube1, cube2);       // 같은 자리 복셀끼리 더함
VoxelOpArithmetic.add(cube, (short) 10);   // 모든 복셀에 10을 더함
```

입력 큐브는 변경하지 않고 언제나 새 큐브를 만들어 돌려줌.

모두 상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

연산 목록

부를 수 있는 메서드

메서드	계산	반환
`add(cube1, cube2)` `add(cube, number)`	`cube1 + cube2` `cube + number`	`BasicDataCube`
`sub(cube1, cube2)` `sub(cube, number)`	`cube1 - cube2` `cube - number`	`BasicDataCube`
`mul(cube1, cube2)` `mul(cube, number)`	`cube1 * cube2` `cube * number`	`BasicDataCube`
`div(cube1, cube2)` `div(cube, number)`	`cube1 / cube2` `cube / number`	`BasicDataCube`
`mod(cube1, cube2)` `mod(cube, number)`	`cube1 % cube2` `cube % number` (나머지)	`BasicDataCube`
`mean(cubes)`	큐브마다 복셀값의 평균	`double[]`
`min(cubes)`	큐브마다 복셀값 중 가장 작은 값	`Object[]`
`max(cubes)`	큐브마다 복셀값 중 가장 큰 값	`Object[]`
`variance(cubes)`	큐브마다 복셀값의 분산	`double[]`
`standardDeviation(cubes)`	큐브마다 복셀값의 표준편차	`double[]`
`pow(cubes)`	큐브마다 복셀값을 제곱	`BasicDataCube[]`
`sqrt(cubes)`	큐브마다 복셀값의 제곱근	`BasicDataCube[]` (모두 double)

`log(cubes)`	큐브마다 복셀값의 자연로그	`BasicDataCube[]` (모두 double)
`sin(cubes, option)`	큐브마다 복셀값의 싸인	`BasicDataCube[]` (모두 double)
`cos(cubes, option)`	큐브마다 복셀값의 코싸인	`BasicDataCube[]` (모두 double)
`tan(cubes, option)`	큐브마다 복셀값의 탄젠트	`BasicDataCube[]` (모두 double)

큐브 리스트를 받는 연산은 입력과 길이·순서가 같은 배열을 돌려줌.

던지는 예외

예외와 원인

예외	언제
`NullPointerException`	큐브나 상수 인자가 `null` 일 때
`IllegalArgumentException`	두 큐브의 level, `(col, row, floor)`, resolution 이 다를 때, 큐브 타입이 bit 나 벡터일 때, 상수가 아래 다섯 객체가 아니거나 NaN·무한대일 때, 삼각함수의 `option` 이 `TRIG_FUNC_RADIAN` 도 `TRIG_FUNC_DEGREE` 도 아닐 때
`ArithmeticException`	결과가 double 을 넘을 때, `'V'` 큐브의 복셀값이 NaN 일 때, 계산 결과가 NaN 일 때, `'V'` 큐브를 0 으로 나누거나 0 으로 나머지를 구할 때, 나누는 상수가 0 일 때, 복셀값이 연산의 정의역 밖일 때 (`sqrt` 의 음수, `log` 의 0 이하, 도 단위 `tan` 의 `90 + 180k`)

입력 제약

두 큐브를 받는 형태에서는 level, `(col, row, floor)`, resolution 이 **모두 같아야 함**.
복셀 단위로 같은 자리끼리 계산하므로 큐브 경계를 넘지 않음.

큐브와 상수를 받는 형태에서는 상수를 `Byte`, `Short`, `Integer`, `Float`, `Double` 객체로 받음.
`Long` 은 받지 않음.

큐브 리스트를 받는 형태는 큐브끼리 level, `(col, row, floor)`, resolution, 타입이 달라도 됨.

지원 타입

`BasicDataCube.DataType.BYTE`, `BasicDataCube.DataType.SHORT`, `BasicDataCube.DataType.INT`,
`BasicDataCube.DataType.FLOAT`, `BasicDataCube.DataType.DOUBLE`.

결과 큐브의 헤더

헤더 중 **자리를 가리키는 값은 그대로 물려받고, 데이터를 설명하는 값은 다시 정함.**

입력에서 물려받는 것과 다시 정하는 것

헤더 필드	결과값

`level`, `col`, `row`, `floor`, `resolution`	첫 번째 인자 큐브의 것을 그대로 물려받음
`datatype`	입력 타입과 **실제 계산 결과**를 함께 보고 정함. 아래 <반환 타입과 승격> 참고
`dataBlockCode`	입력이 모두 `'V'`이면 `'V'`, 하나라도 `'S'`이면 `'S'`. 상수는 센티넬이 없으므로 큐브만 봄
`sentinel`	`'S'`이면 byte/short/int 는 반환 타입의 최대값, float/double 은 NaN. `'V'`이면 없음(`null`)

타입은 **같은 메서드라도 넘어온 복셀값에 따라 달라짐.**

byte 큐브 둘을 더해도 결과가 byte 일 수도 short 일 수도 있음.

그러므로 값을 읽기 전에 반드시 `BasicDataCube.datatype()`을 확인해야 함.

반환 타입과 승격

먼저 두 입력 중 **큰 쪽의 타입**을 고름 (byte < short < int < float < double).

큐브끼리 계산하지 않는 `pow` 는 각 큐브의 타입에서 시작함.

그 타입에 실제 계산 결과가 담기지 않으면 아래 순서로 승격함.

```
byte -> short -> int -> double
float -> double
```

계산은 double 로 하고 정해진 반환 타입으로 저장하므로 반올림이 한 번만 일어남.

정수 타입끼리 계산했는데 결과에 소수가 있으면 (나누기) double 이 됨.

`sqrt`, `log`, `sin`, `cos`, `tan` 은 입력 타입과 상관없이 언제나 double 을 돌려주므로 위 사다리를 타지 않음.

반환 타입은 **큐브 단위로 하나**임.

복셀 하나만 넘쳐도 큐브 전체가 승격됨.

double 에서도 넘치면 `ArithmeticException` 을 던짐.

복셀값 자체가 무한대여도 결과가 double 을 넘으므로 마찬가지임.

더하기의 반환 타입 예

입력 타입	가장 큰 계산 결과	반환 타입	까닭
byte + byte	30	byte	그대로 담김
byte + byte	200	short	byte 에 안 담김
byte + int	8	int	큰 쪽을 따름
int + int	4000000000	double	int 다음이 long 이 아님
int + float	3.5	float	큰 쪽을 따름

센티넬과 NaN

복셀 하나하나를 볼 때:

- `'S'`이고 복셀값이 센티넬이면 그 자리의 결과도 센티넬
- `'S'`이고 float/double 복셀값이 NaN 이면 그 자리의 결과도 센티넬
- `'V'`이고 float/double 복셀값이 NaN 이면 `ArithmeticException`
- 계산 결과가 NaN 이면 `ArithmeticException` (무한대가 섞인 자리)

결과 큐브의 센티넬 값은 **반환 타입에 따라 정해짐** (승격하면 승격한 타입의 것).

반환 타입별 센티넬

반환 타입	센티넬
byte, short, int	그 타입의 최대값 (`Byte.MAX_VALUE`, `Short.MAX_VALUE`, `Integer.MAX_VALUE`)
float, double	`Float.NaN`, `Double.NaN`

그래서 결과가 `'S'`이고 **정수 타입이면 최대값은 센티넬 자리로 예약되어 값으로 쓸 수 없음.**

계산 결과가 그 값이면 넘친 것으로 보고 승격함.

예를 들어 byte `100 + 27 = 127` 은 `Byte.MAX_VALUE` 와 같으므로 결과는 short 가 됨.

결과가 `'V'`이면 센티넬이 없으므로 예약하지 않고 byte 127 에 그대로 담음.

float, double 은 센티넬이 NaN 이라 최대값을 예약하지 않음.

`Float.MAX_VALUE`, `Double.MAX_VALUE` 도 값으로 쓸 수 있음.

Field Summary

Fields

Modifier and Type	Field	Description
`static final int`	`TRIG_FUNC_DEGREE`	삼각함수의 `option` 값 — 복셀값을 **각도**로 봄.
`static final int`	`TRIG_FUNC_RADIAN`	삼각함수의 `option` 값 — 복셀값을 **라디안**으로 봄.

Method Summary

Modifier and Type	Method	Description
`static BasicDataCube`	`add(BasicDataCube cube1, BasicDataCube cube2)`	두 큐브의 같은 자리 복셀값끼리 더함.
`static BasicDataCube`	`add(BasicDataCube cube, Object number)`	큐브의 복셀값마다 같은 상수를 더함.
`static BasicDataCube[]`	`cos(BasicDataCube[] cubes, int option)`	큐브마다 복셀값의 코싸인을 구함.
`static BasicDataCube`	`div(BasicDataCube cube1, BasicDataCube cube2)`	두 큐브의 같은 자리 복셀값끼리 나눔.

static BasicDataCube	div(BasicDataCube cube, Object number)	큐브의 복셀값마다 같은 상수로 나눔.
static BasicDataCube[]	log(BasicDataCube[] cubes)	큐브마다 복셀값의 자연로그를 구함.
static Object[]	max(BasicDataCube[] cubes)	큐브마다 복셀값 중 가장 큰 값을 구함.
static double[]	mean(BasicDataCube[] cubes)	큐브마다 복셀값 평균을 구함.
static Object[]	min(BasicDataCube[] cubes)	큐브마다 복셀값 중 가장 작은 값을 구함.
static BasicDataCube	mod(BasicDataCube cube1, BasicDataCube cube2)	두 큐브의 같은 자리 복셀값끼리 나눈 나머지를 구함.
static BasicDataCube	mod(BasicDataCube cube, Object number)	큐브의 복셀값마다 같은 상수로 나눈 나머지를 구함.
static BasicDataCube	mul(BasicDataCube cube1, BasicDataCube cube2)	두 큐브의 같은 자리 복셀값끼리 곱함.
static BasicDataCube	mul(BasicDataCube cube, Object number)	큐브의 복셀값마다 같은 상수를 곱함.
static BasicDataCube[]	pow(BasicDataCube[] cubes)	큐브마다 복셀값의 제곱을 구함.
static BasicDataCube[]	sin(BasicDataCube[] cubes, int option)	큐브마다 복셀값의 싸인을 구함.
static BasicDataCube[]	sqrt(BasicDataCube[] cubes)	큐브마다 복셀값의 제곱근을 구함.
static double[]	standardDeviation(BasicDataCube[] cubes)	큐브마다 복셀값 표준편차를 구함.
static BasicDataCube	sub(BasicDataCube cube1, BasicDataCube cube2)	두 큐브의 같은 자리 복셀값끼리 뺌.
static BasicDataCube	sub(BasicDataCube cube, Object number)	큐브의 복셀값마다 같은 상수를 뺌.
static BasicDataCube[]	tan(BasicDataCube[] cubes, int option)	큐브마다 복셀값의 탄젠트를 구함.
static double[]	variance(BasicDataCube[] cubes)	큐브마다 복셀값 분산을 구함.

Field Details

TRIG_FUNC_RADIAN

```
public static final int TRIG_FUNC_RADIAN
```

삼각함수의 `option` 값 — 복셀값을 **라디안**으로 봄.

값은 `1`.

`Math.PI / 2` 가 90 도임.

See Also:

- Constant Field Values

TRIG_FUNC_DEGREE

```
public static final int TRIG_FUNC_DEGREE
```

삼각함수의 `option` 값 — 복셀값을 **각도**로 봄.

값은 `2`.

라디안으로 바꾼 뒤 계산하므로 `90` 이 `Math.PI / 2` 와 같음.

See Also:

- Constant Field Values

10.1. add API

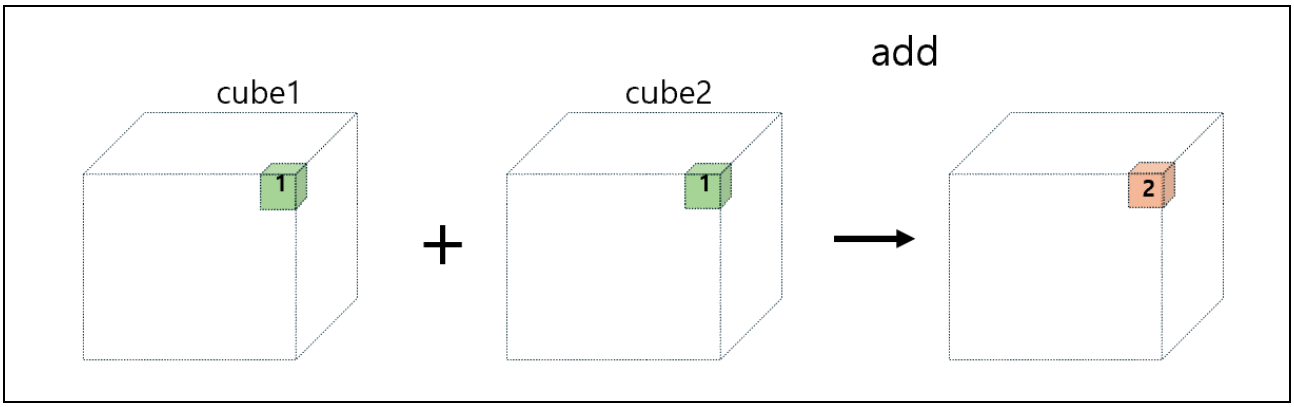
add(BasicDataCube cube1, BasicDataCube cube2)

```
public static BasicDataCube add(BasicDataCube cube1,
    BasicDataCube cube2)
```

두 큐브의 같은 자리 복셀값끼리 더함.

add

메서드	파라미터 / 반환	설명
add	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 두 큐브에 대한 더하기 연산. 입력되는 두 큐브가 서로 다른 타입이어도 됨. (이후 연산도 마찬가지 임)
	BasicDataCube cube1	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double
	BasicDataCube cube2)	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: cube1 과 cube2 중에서 큰쪽의 타입을 반환. dataBlockCode: 입력 큐브가 모두 'V'일때 'V'이며 sentinel=0. 그렇지 않으면 'S'이며 이때 sentinel 은: * byte,short,int 일때: 반환타입의 최대값 (타입승격시 승격타입의 최대값) (주의: 오버플로 체크시 반환타입의 최대값 고려할것) (참고: int 에서 double 로 타입승격시 Double.NaN 사용) * float 일때: Float.NaN (0x7FC00000). (타입승격시 Double.NaN) * double 일때: Double.NaN (0x7FF8000000000000) (참고: 이후에 나오는 산술연산도 이와 같은 방식으로 처리)
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환. (byte->short, short->int, int->double, float->double, double->예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-031-ARI(byte 두 파형)의 입력 파일이 있는 폴더에서 실행.

```
TS-031-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-031-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube1 = new BasicDataCube("cube1.bdc");
        BasicDataCube cube2 = new BasicDataCube("cube2.bdc");
        BasicDataCube result = VoxelOpArithmetic.add(cube1, cube2);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


}
```

Parameters:

`cube1` - 레스터 데이터를 가지는 단일 큐브.

byte, short, int, float, double 중 하나여야 함

`cube2` - `cube1` 과 level, `(col, row, floor)`, resolution 이 같은 단일 큐브.

byte, short, int, float, double 중 하나여야 함

Returns:

더한 결과를 담은 새 큐브

Throws:

`NullPointerException` - `cube1` 또는 `cube2` 가 `null` 일 때

`IllegalArgumentException` - 두 큐브의 level, `(col, row, floor)`, resolution 이 다를 때, 지원하지 않는 타입일 때

`ArithmeticException` - 결과가 double 에서도 넘칠 때, `'V'` 큐브의 복셀값이 NaN 이거나 계산 결과가 NaN 일 때

See Also:

- `add(BasicDataCube, Object)`

add(BasicDataCube cube, Object number)

```
public static BasicDataCube add(BasicDataCube cube,
    Object number)
```

큐브의 복셀값마다 같은 상수를 더함.

add

메서드	파라미터 / 반환	설명
add	(	입력된 레스터 데이터 큐브에 대한 상수값 더하기 연산.
	BasicDataCube cube	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double
	Object number)	상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: cube 와 value 중에서 큰쪽의 타입을 반환.
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환. (byte->short, short->int, int->double, float->double, double->예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.



`number` 의 타입도 반환 타입에 들어감.

byte 큐브에 `Integer` 를 더하면 값이 작더라도 결과는 int 임.

예제

시험 시나리오 TS-034-ARI(정수 상수)의 입력 파일이 있는 폴더에서 실행.

```
TS-034-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-034-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {
```

```
    public static void main(String[] args) throws Exception {
        BasicDataCube cube = new BasicDataCube("cube.bdc");
        BasicDataCube result = VoxelOpArithmetic.add(cube, 100);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }

}
```

Parameters:

`cube` - 레스터 데이터를 가지는 단일 큐브.

byte, short, int, float, double 중 하나여야 함

`number` - 더할 상수.

`Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나여야 함

Returns:

더한 결과를 담은 새 큐브

Throws:

`NullPointerException` - `cube` 또는 `number` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입일 때, `number` 가 위 다섯 가지 객체가 아닐 때, `number` 가 NaN 이거나 무한대일 때

`ArithmeticException` - 결과가 double 에서도 넘칠 때, `'V'` 큐브의 복셀값이 NaN 이거나 계산 결과가 NaN 일 때

See Also:

- `add(BasicDataCube, BasicDataCube)`

10.2. sub API

sub(BasicDataCube cube1, BasicDataCube cube2)

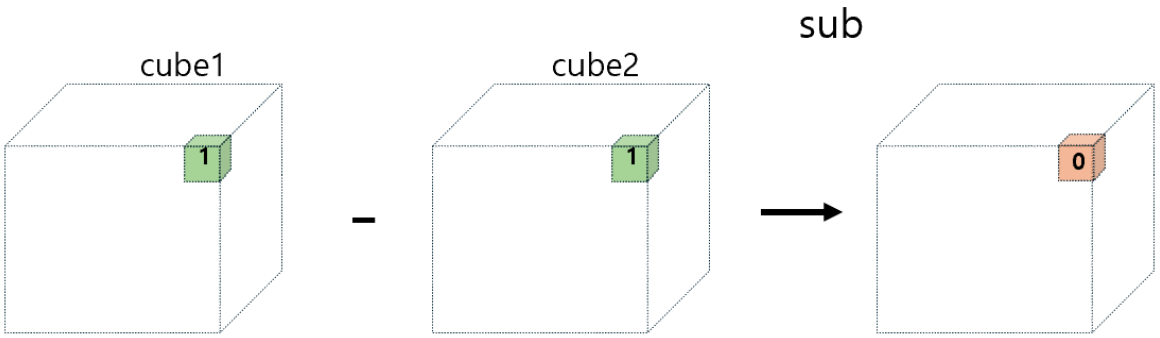
```
public static BasicDataCube sub(BasicDataCube cube1,
    BasicDataCube cube2)
```

두 큐브의 같은 자리 복셀값끼리 뺌.

`cube1 - cube2` 를 계산함.

sub

메서드	파라미터 / 반환	설명
sub	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 두 큐브에 대한 빼기 연산.
	BasicDataCube cube1	레스터 데이터를 가지는 단일 큐브.

		지원타입: byte,short,int,float,double
	BasicDataCube cube2)	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: cube1 과 cube2 중에서 큰쪽의 타입을 반환.
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환. (byte->short, short->int, int->double, float->double, double->예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.
		

예를 들어 byte `-100 - 100 = -200` 은 byte 에 담기지 않아 결과가 short 가 됨.

예제

시험 시나리오 TS-037-ARI(int 두 필드)의 입력 파일이 있는 폴더에서 실행.

```
TS-037-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-037-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube1 = new BasicDataCube("cube1.bdc");
        BasicDataCube cube2 = new BasicDataCube("cube2.bdc");
        BasicDataCube result = VoxelOpArithmetic.sub(cube1, cube2);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


}
```

Parameters:

`cube1` - 빼기의 왼쪽 큐브.

레스터 데이터를 가지는 단일 큐브로, byte, short, int, float, double 중 하나여야 함

`cube2` - 빼기의 오른쪽 큐브.

`cube1` 과 level, `(col, row, floor)`, resolution 이 같아야 하고, byte, short, int, float, double 중 하나여야 함

Returns:

뺀 결과를 담은 새 큐브

Throws:

`NullPointerException` - `cube1` 또는 `cube2` 가 `null` 일 때

`IllegalArgumentException` - 두 큐브의 level, `(col, row, floor)`, resolution 이 다를 때, 지원하지 않는 타입일 때

`ArithmeticException` - 결과가 double 에서도 넘칠 때, `'V'` 큐브의 복셀값이 NaN 이거나 계산 결과가 NaN 일 때

See Also:

- `sub(BasicDataCube, Object)`
- `add(BasicDataCube, BasicDataCube)`

sub(BasicDataCube cube, Object number)

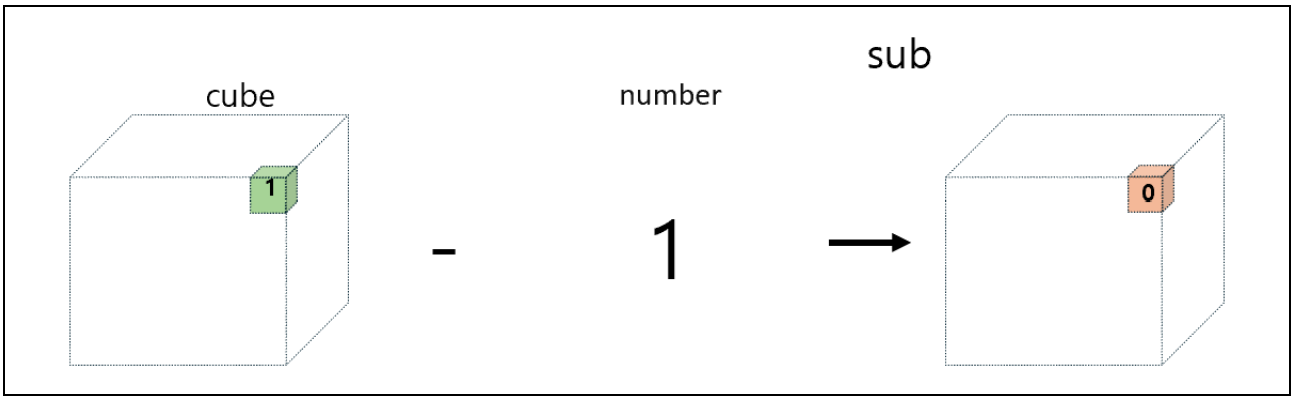
```
public static BasicDataCube sub(BasicDataCube cube,
    Object number)
```

큐브의 복셀값마다 같은 상수를 뺌.

`cube - number` 를 계산함.

sub

메서드	파라미터 / 반환	설명
sub	(	입력된 레스터 데이터 큐브에 대한 상수값 빼기 연산.
	BasicDataCube cube	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double
	Object number)	상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: cube 와 value 중에서 큰쪽의 타입을 반환.
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환. (byte->short, short->int, int->double, float->double, double->예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.



`number` 의 타입도 반환 타입에 들어감.

byte 큐브에서 `Integer` 를 빼면 값이 작더라도 결과는 int 임.

예제

시험 시나리오 TS-040-ARI(정수 상수)의 입력 파일이 있는 폴더에서 실행.

```
TS-040-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-040-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube = new BasicDataCube("cube.bdc");
        BasicDataCube result = VoxelOpArithmetic.sub(cube, 55);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


}
```

Parameters:

`cube` - 레스터 데이터를 가지는 단일 큐브.

byte, short, int, float, double 중 하나여야 함

`number` - 뺄 상수.

`Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나여야 함

Returns:

뺀 결과를 담은 새 큐브

Throws:

`NullPointerException` - `cube` 또는 `number` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입일 때, `number` 가 위 다섯 가지 객체가 아닐 때, `number` 가 NaN 이거나 무한대일 때

`ArithmeticException` - 결과가 double 에서도 넘칠 때, `'V'` 큐브의 복셀값이 NaN 이거나 계산 결과가 NaN 일 때

See Also:

- `sub(BasicDataCube, BasicDataCube)`
- `add(BasicDataCube, Object)`

10.3. mul API

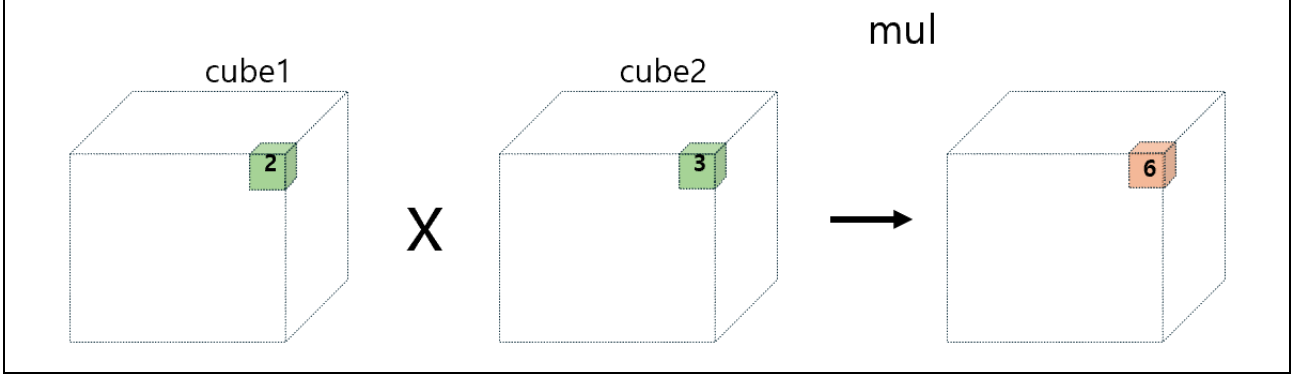
mul(BasicDataCube cube1, BasicDataCube cube2)

```
public static BasicDataCube mul(BasicDataCube cube1,
    BasicDataCube cube2)
```

두 큐브의 같은 자리 복셀값끼리 곱함.

mul

메서드	파라미터 / 반환	설명
mul	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 두 큐브에 대한 곱하기 연산.
	BasicDataCube cube1	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double
	BasicDataCube cube2)	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: cube1 과 cube2 중에서 큰쪽의 타입을 반환.
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환. (byte->short, short->int, int->double, float->double, double->예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.



결과의 절대값이 253 을 넘으면 double 이 담을 수 있는 데까지만 정확함.

예를 들어 byte `100 * 100 = 10000` 은 byte 에 담기지 않아 결과가 short 가 됨.

예제

시험 시나리오 TS-043-ARI(short 두 필드)의 입력 파일이 있는 폴더에서 실행.

```
TS-043-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-043-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube1 = new BasicDataCube("cube1.bdc");
        BasicDataCube cube2 = new BasicDataCube("cube2.bdc");
        BasicDataCube result = VoxelOpArithmetic.mul(cube1, cube2);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


}
```

Parameters:

`cube1` - 레스터 데이터를 가지는 단일 큐브.

byte, short, int, float, double 중 하나여야 함

`cube2` - `cube1` 과 level, `(col, row, floor)`, resolution 이 같은 단일 큐브.

byte, short, int, float, double 중 하나여야 함

Returns:

곱한 결과를 담은 새 큐브

Throws:

`NullPointerException` - `cube1` 또는 `cube2` 가 `null` 일 때

`IllegalArgumentException` - 두 큐브의 level, `(col, row, floor)`, resolution 이 다를 때, 지원하지 않는 타입일 때

`ArithmeticException` - 결과가 double 에서도 넘칠 때, `'V'` 큐브의 복셀값이 NaN 이거나 계산 결과가 NaN 일 때

See Also:

- `mul(BasicDataCube, Object)`

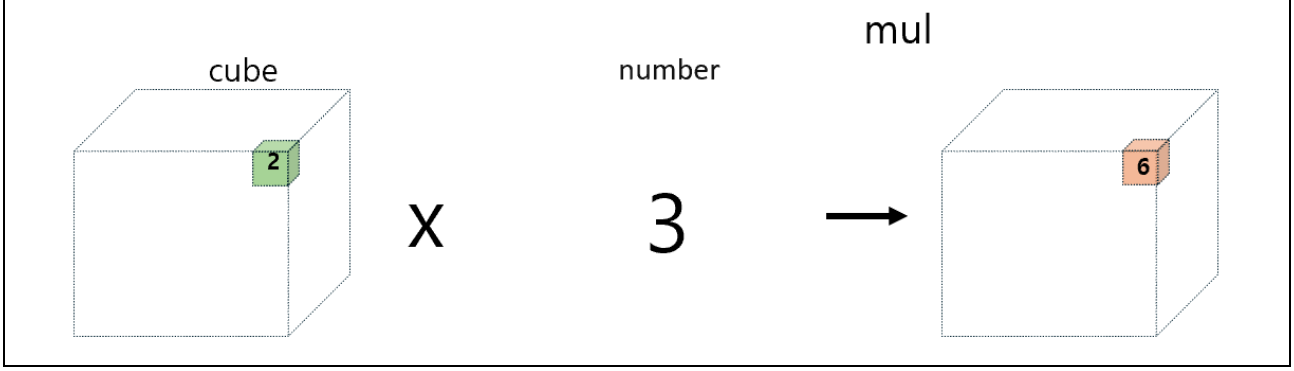
mul(BasicDataCube cube, Object number)

```
public static BasicDataCube mul(BasicDataCube cube,
    Object number)
```

큐브의 복셀값마다 같은 상수를 곱함.

mul

메서드	파라미터 / 반환	설명
mul	(	입력된 레스터 데이터 큐브에 대한 상수값 곱하기 연산.
	BasicDataCube cube	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double
	Object number)	상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: cube 와 value 중에서 큰쪽의 타입을 반환.
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환. (byte->short, short->int, int->double, float->double, double->예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.



결과의 절대값이 253 을 넘으면 double 이 담을 수 있는 데까지만 정확함.

`number` 의 타입도 반환 타입에 들어감.

byte 큐브에 `Integer` 를 곱하면 값이 작더라도 결과는 int 임.

예제

시험 시나리오 TS-046-ARI(정수 상수)의 입력 파일이 있는 폴더에서 실행.

```
TS-046-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-046-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube = new BasicDataCube("cube.bdc");
        BasicDataCube result = VoxelOpArithmetic.mul(cube, 3);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }
```

```
}
```

Parameters:

`cube` - 레스터 데이터를 가지는 단일 큐브.

byte, short, int, float, double 중 하나여야 함

`number` - 곱할 상수.

`Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나여야 함

Returns:

곱한 결과를 담은 새 큐브

Throws:

`NullPointerException` - `cube` 또는 `number` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입일 때, `number` 가 위 다섯 가지 객체가 아닐 때, `number` 가 NaN 이거나 무한대일 때

`ArithmeticException` - 결과가 double 에서도 넘칠 때, `'V'` 큐브의 복셀값이 NaN 이거나 계산 결과가 NaN 일 때

See Also:

- `mul(BasicDataCube, BasicDataCube)`

10.4. div API

div(BasicDataCube cube1, BasicDataCube cube2)

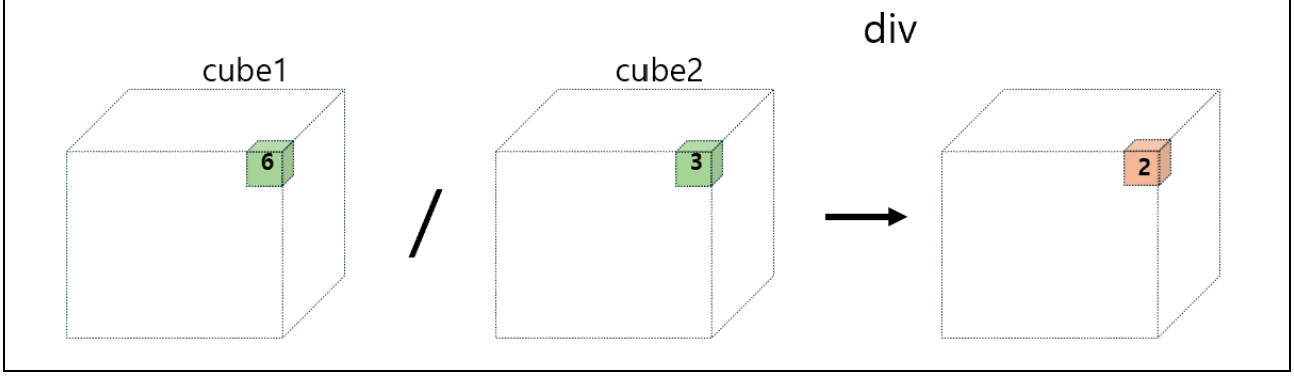
```
public static BasicDataCube div(BasicDataCube cube1,
    BasicDataCube cube2)
```

두 큐브의 같은 자리 복셀값끼리 나눔.

`cube1 / cube2` 를 계산함.

div

메서드	파라미터 / 반환	설명
div	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 두 큐브에 대한 나누기 연산. 큐브 2 의 복셀값이 0 일때: * 큐브의 data_block_code 가 'V'이면 예외처리 * 큐브의 data_block_code 가 'S'이면 결과는 sentinel 값임
	BasicDataCube cube1	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double
	BasicDataCube cube2)	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double

	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: cube1 과 cube2 중에서 큰쪽의 타입을 반환. 정수와 정수간의 연산결과가 실수일때 double 로 타입승격.
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환. (byte->short, short->int, int->double, float->double, double->예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.
		

정수끼리 나눠도 버리지 않음.

결과에 소수가 있으면 반환 타입이 double 이 됨.

예를 들어 int `7 / 2 = 3.5` 는 double 이고, int `8 / 2 = 4` 는 int 임.

`cube2` 의 복셀값이 0 인 자리는 결과가 `'S'`이면 센티넬이 되고, `'V'`이면 `ArithmeticException` 을 던짐.

센티넬 자리는 계산하지 않으므로 센티넬 값이 0 이어도 상관없음.

예제

시험 시나리오 TS-049-ARI(float ÷ 기울기)의 입력 파일이 있는 폴더에서 실행.

```
TS-049-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-049-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube1 = new BasicDataCube("cube1.bdc");
        BasicDataCube cube2 = new BasicDataCube("cube2.bdc");
        BasicDataCube result = VoxelOpArithmetic.div(cube1, cube2);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


}
```

Parameters:

`cube1` - 나눗셈의 왼쪽 큐브.

레스터 데이터를 가지는 단일 큐브로, byte, short, int, float, double 중 하나여야 함

`cube2` - 나눗셈의 오른쪽 큐브.

`cube1` 과 level, `(col, row, floor)`, resolution 이 같아야 하고, byte, short, int, float, double 중 하나여야 함.

결과가 `'V'`이면 센티넬이 아닌 복셀값에 0 이 없어야 함

Returns:

나눈 결과를 담은 새 큐브

Throws:

`NullPointerException` - `cube1` 또는 `cube2` 가 `null` 일 때

`IllegalArgumentException` - 두 큐브의 level, `(col, row, floor)`, resolution 이 다를 때, 지원하지 않는 타입일 때

`ArithmeticException` - 결과가 `'V'`인데 `cube2` 의 복셀값이 0 일 때, 결과가 double 에서도 넘칠 때, `'V'` 큐브의 복셀값이 NaN 이거나 계산 결과가 NaN 일 때

See Also:

- `div(BasicDataCube, Object)`

div(BasicDataCube cube, Object number)

```
public static BasicDataCube div(BasicDataCube cube,
    Object number)
```

큐브의 복셀값마다 같은 상수로 나눔.

`cube / number` 를 계산함.

div

메서드	파라미터 / 반환	설명
div	(	입력된 레스터 데이터 큐브에 대한 상수값 나누기 연산.
	BasicDataCube cube	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double
	Object number)	상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 (이 값이 0 이면 예외처리)
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: cube 와 value 중에서 큰쪽의 타입을 반환. 정수와 정수간의 연산결과가 실수일때 double 로 타입승격.
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환. (byte->short, short->int, int->double, float->double, double-

		>예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.



정수끼리 나눠도 버리지 않음.

결과에 소수가 있으면 반환 타입이 double 이 됨.

`number` 가 0 이면 큐브를 보지 않고 `ArithmeticException` 을 던짐.

`number` 의 타입도 반환 타입에 들어감.

byte 큐브를 `Integer` 로 나누면 나누어떨어지더라도 결과는 int 임.

예제

시험 시나리오 TS-052-ARI(실수 상수)의 입력 파일이 있는 폴더에서 실행.

```
TS-052-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-052-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube = new BasicDataCube("cube.bdc");
        BasicDataCube result = VoxelOpArithmetic.div(cube, 4.0);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


}
```

Parameters:

`cube` - 레스터 데이터를 가지는 단일 큐브.

byte, short, int, float, double 중 하나여야 함

`number` - 나눌 상수.

`Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나여야 하고 0 이 아니어야 함

Returns:

나눈 결과를 담은 새 큐브

Throws:

`NullPointerException` - `cube` 또는 `number` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입일 때, `number` 가 위 다섯 가지 객체가 아닐 때, `number` 가 NaN 이거나 무한대일 때

`ArithmeticException` - `number` 가 0 일 때, 결과가 double 에서도 넘칠 때, `'V'` 큐브의 복셀값이 NaN 이거나 계산 결과가 NaN 일 때

See Also:

- `div(BasicDataCube, BasicDataCube)`

10.5. mod API

mod(BasicDataCube cube1, BasicDataCube cube2)

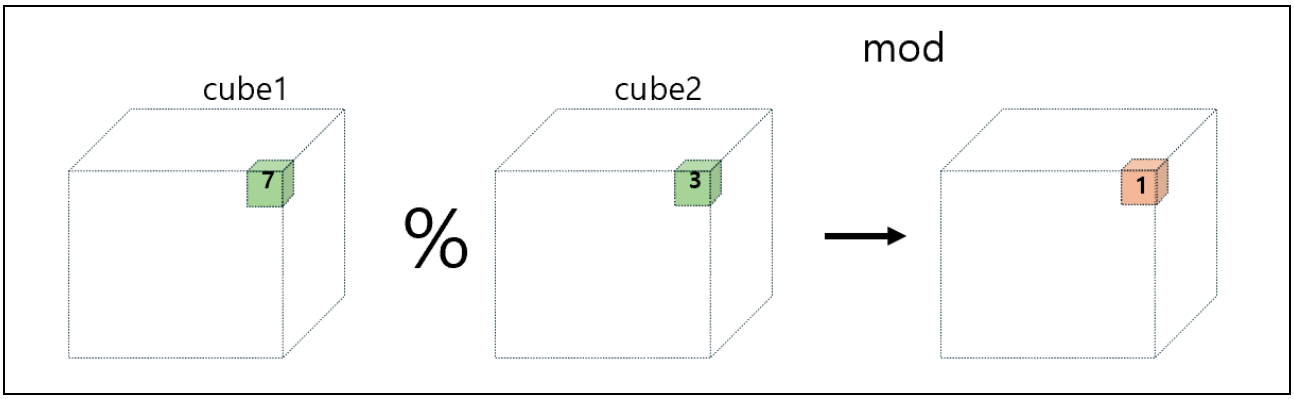
```
public static BasicDataCube mod(BasicDataCube cube1,
    BasicDataCube cube2)
```

두 큐브의 같은 자리 복셀값끼리 나눈 나머지를 구함.

`cube1 % cube2` 를 계산함.

mod

메서드	파라미터 / 반환	설명
mod	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 두 큐브에 대한 MOD 연산. 큐브 2 의 복셀값이 0 일때: * 큐브의 data_block_code 가 'V'이면 예외처리 * 큐브의 data_block_code 가 'S'이면 결과는 sentinel 값임
	BasicDataCube cube1	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double
	BasicDataCube cube2)	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: cube1 과 cube2 중에서 큰쪽의 타입을 반환.
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환. (byte->short, short->int, int->double, float->double, double->예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.



나머지의 부호는 **나누어지는 쪽(`cube1`)을 따름** — 자바 `%`와 같음.

`7 % 3 = 1`, `-7 % 3 = -1`, `7 % -3 = 1`, `-7 % -3 = -1` 임.

`cube2` 의 복셀값이 0 인 자리는 결과가 `'S'`이면 센티넬이 되고, `'V'`이면 `ArithmeticException` 을 던짐.
센티넬 자리는 계산하지 않으므로 센티넬 값이 0 이어도 상관없음.

예제

시험 시나리오 TS-055-ARI(int % 줄무늬)의 입력 파일이 있는 폴더에서 실행.

```
TS-055-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-055-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube1 = new BasicDataCube("cube1.bdc");
        BasicDataCube cube2 = new BasicDataCube("cube2.bdc");
        BasicDataCube result = VoxelOpArithmetic.mod(cube1, cube2);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


}
```

Parameters:

`cube1` - 나누어지는 큐브.

레스터 데이터를 가지는 단일 큐브로, byte, short, int, float, double 중 하나여야 함

`cube2` - 나누는 큐브.

`cube1` 과 level, `(col, row, floor)`, resolution 이 같아야 하고, byte, short, int, float, double 중 하나여야 함.

결과가 `'V'`이면 센티넬이 아닌 복셀값에 0 이 없어야 함

Returns:

나머지를 담은 새 큐브

Throws:

`NullPointerException` - `cube1` 또는 `cube2` 가 `null` 일 때

`IllegalArgumentException` - 두 큐브의 level, `(col, row, floor)`, resolution 이 다를 때, 지원하지 않는 타입일 때

`ArithmeticException` - 결과가 `'V'` 인데 `cube2` 의 복셀값이 0 일 때, 결과가 double 에서도 넘칠 때, `'V'` 큐브의 복셀값이 NaN 이거나 계산 결과가 NaN 일 때

See Also:

- `mod(BasicDataCube, Object)`

mod(BasicDataCube cube, Object number)

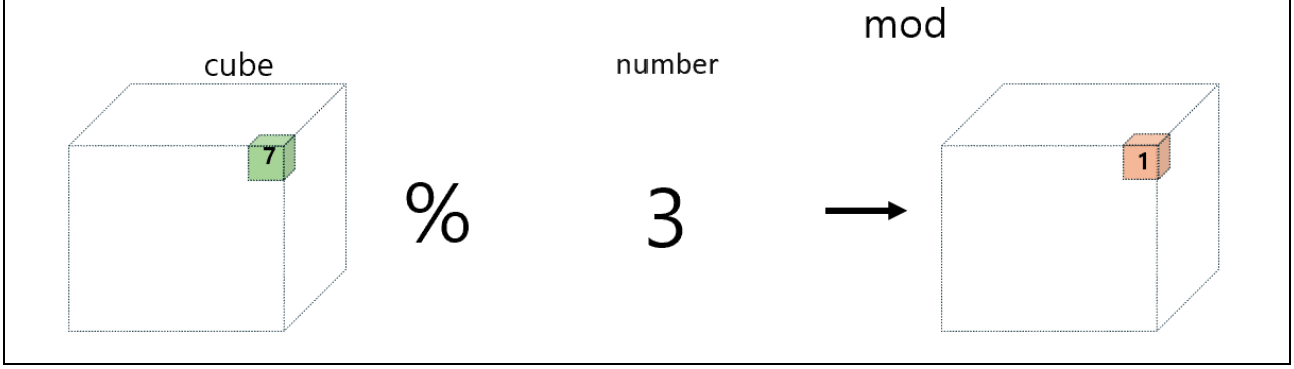
```
public static BasicDataCube mod(BasicDataCube cube,
    Object number)
```

큐브의 복셀값마다 같은 상수로 나눈 나머지를 구함.

`cube % number` 를 계산함.

mod

메서드	파라미터 / 반환	설명
mod	(	입력된 레스터 데이터 큐브에 대한 상수값 MOD 연산.
	BasicDataCube cube	레스터 데이터를 가지는 단일 큐브. 지원타입: byte,short,int,float,double
	Object number)	상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 (이 값이 0 이면 예외처리)
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: cube 와 value 중에서 큰쪽의 타입을 반환.
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환. (byte->short, short->int, int->double, float->double, double->예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.



나머지의 부호는 **나누어지는 쪽(cube)을 따름** — 자바 `%`와 같음.

`7 % 3 = 1`, `-7 % 3 = -1`, `7 % -3 = 1`, `-7 % -3 = -1` 임.

`number` 가 0 이면 큐브를 보지 않고 `ArithmeticException` 을 던짐.

`number` 의 타입도 반환 타입에 들어감.

byte 큐브를 `Integer` 로 나누면 결과는 int 임.

예제

시험 시나리오 TS-058-ARI(정수 상수)의 입력 파일이 있는 폴더에서 실행.

```
TS-058-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-058-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube = new BasicDataCube("cube.bdc");
        BasicDataCube result = VoxelOpArithmetic.mod(cube, 7);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


}
```

Parameters:

`cube` - 레스터 데이터를 가지는 단일 큐브.

byte, short, int, float, double 중 하나여야 함

`number` - 나누는 상수.

`Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나여야 하고 0 이 아니어야 함

Returns:

나머지를 담은 새 큐브

Throws:

`NullPointerException` - `cube` 또는 `number` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입일 때, `number` 가 위 다섯 가지 객체가 아닐 때, `number` 가 NaN 이거나 무한대일 때

`ArithmeticException` - `number` 가 0 일 때, 결과가 double 에서도 넘칠 때, `'V'` 큐브의 복셀값이 NaN 이거나 계산 결과가 NaN 일 때

See Also:

- `mod(BasicDataCube, BasicDataCube)`

10.6. mean API

```
public static double[] mean(BasicDataCube[] cubes)
```

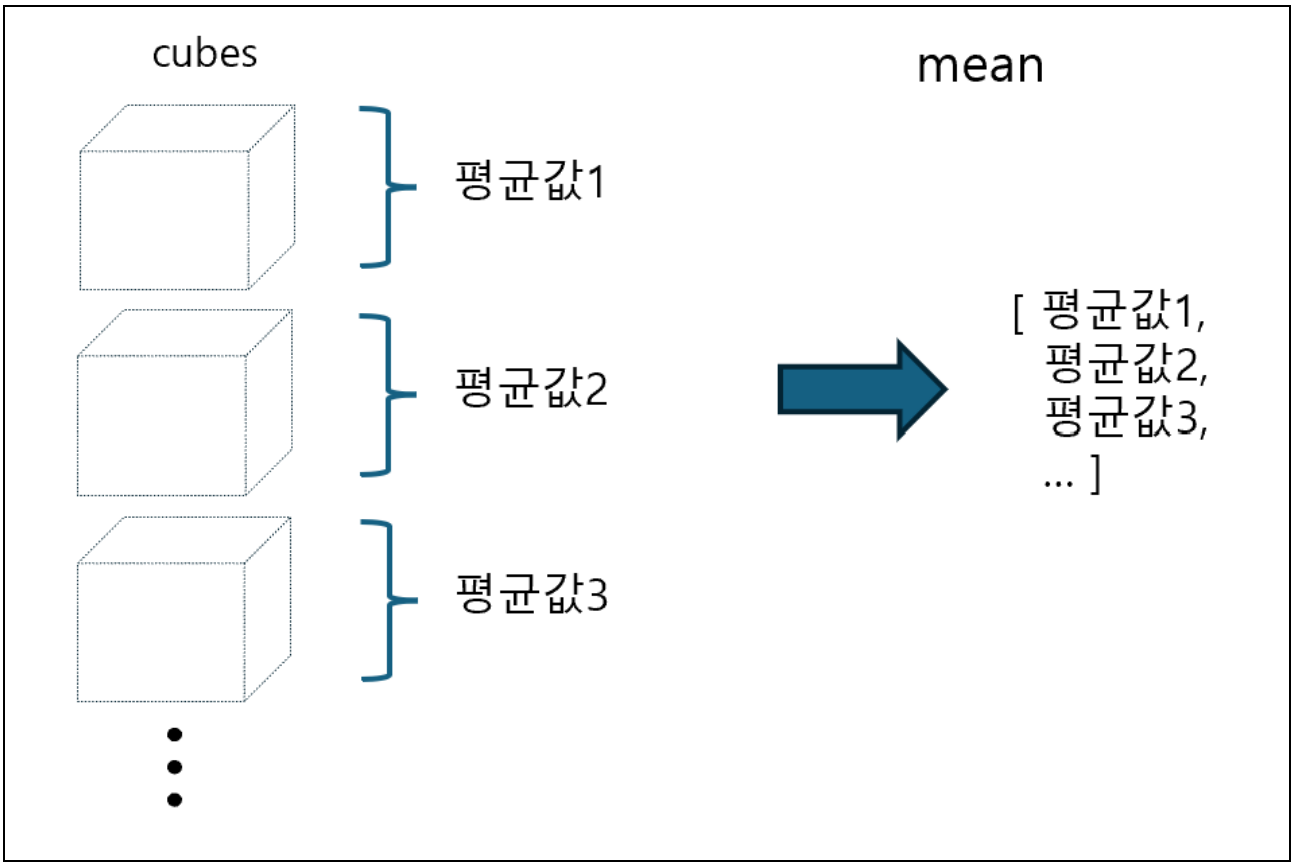
큐브마다 복셀값 평균을 구함.

mean

메서드	파라미터 / 반환	설명

mean	(	입력된 레스터 데이터 큐브들에 대해서
		각 큐브별로 큐브내 모든 복셀값들의 평균값 연산.
		(참고: 모든 복셀값의 평균을 구하기 위해서는 먼저 모든 복셀값을 더해야 함.
		이때, 복셀값을 더해 나가는 도중에 오버플로 발생시 다음과 같은 방법으로 처리:
		<double 타입일때 예제>
		double average=0.0; int n=0;
		for (double x : values) { n++; average+=(x-average)/n; }
		구현이 간단하며, 합계 오버플로를 많은 부분 방지함)
		(공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)

	BasicDataCube[] cubes)	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	double[]	큐브별로 구해진 평균값들을 double 리스트로 반환. 반환값의 순서는 입력 cubes 의 순서와 일치.
		bit/V2/V3/V4 타입에 대한 지원은 미정.



센티넬 자리는 세지 않음.

`'S'`이고 float/double 복셀값이 NaN 인 자리도 센티넬로 보고 빼놓음.

값을 넣지 않은 복셀은 0 이며 함께 봄.

셀 값이 하나도 없는 큐브는 `Double.NaN`.

예제

시험 시나리오 TS-063-ARI(타입 혼합)의 입력 파일이 있는 폴더에서 실행.

```
TS-063-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-063-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        double[] result = VoxelOpArithmetic.mean(cubes);
        System.out.println("result: " + Arrays.toString(result));
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
```

```
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

원소가 `null` 이 아니어야 하고, byte, short, int, float, double 중 하나여야 함

Returns:

큐브별 평균.

입력과 길이가 같고 순서도 같음.

빈 배열을 넘기면 빈 배열을 돌려줌

Throws:

`NullPointerException` - `cubes` 또는 그 원소가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입이 섞였을 때

`ArithmeticException` - 어느 큐브의 평균이 double 을 넘을 때, 복셀값이 무한대일 때, `'V'` 큐브의 복셀값이 NaN 일 때

10.7. min API

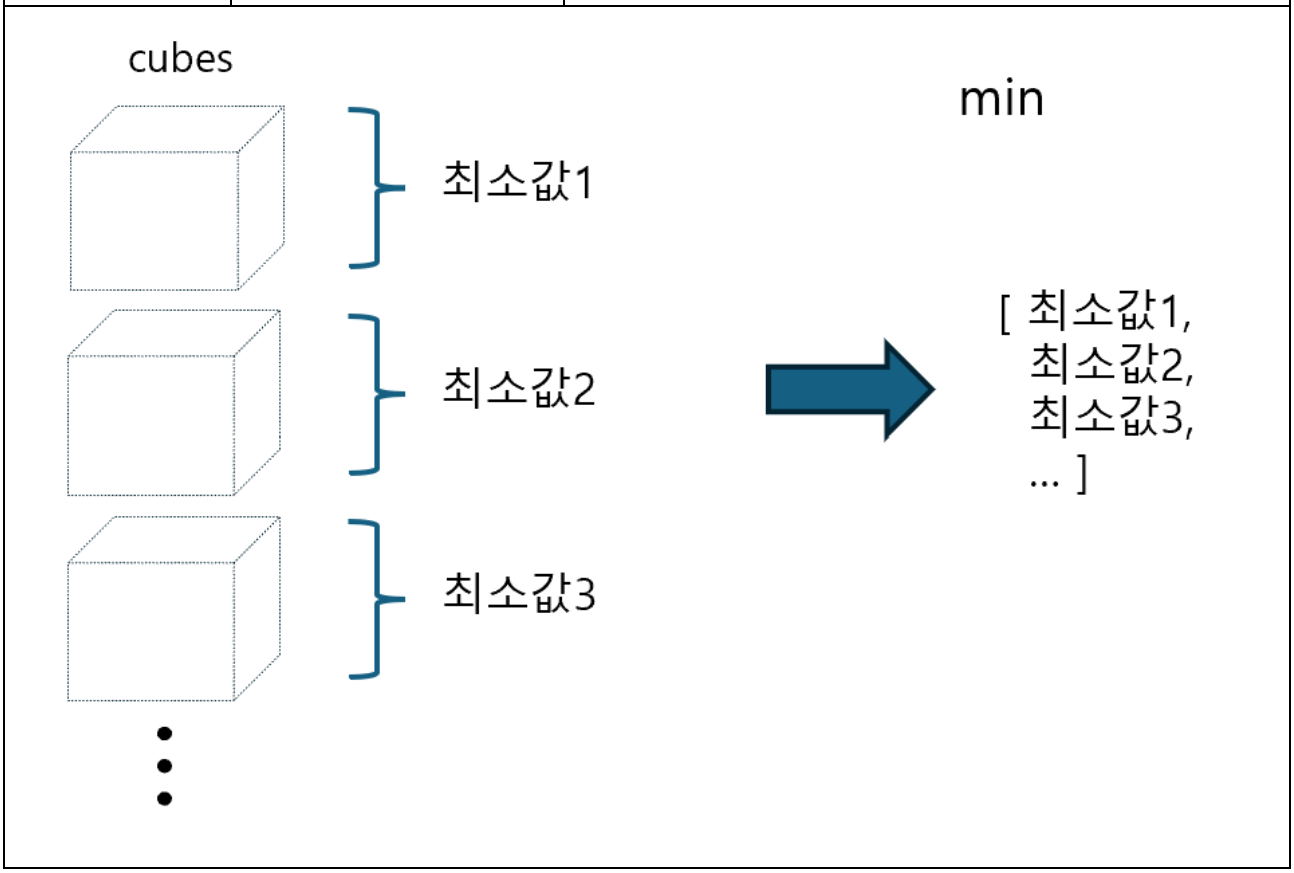
```
public static Object[] min(BasicDataCube[] cubes)
```

큐브마다 복셀값 중 가장 작은 값을 구함.

min

메서드	파라미터 / 반환	설명
min	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 모든 복셀값들 중에서 최소값 연산. (공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes)	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	Object[]	큐브별로 구해진 최소값들을 각 큐브의 데이터 타입에 해당하는 Object 리스트로 반환. 반환값의 순서는 입력 cubes 의 순서와 일치. Object: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체

		bit/V2/V3/V4 타입에 대한 지원은 미정.



원소마다 그 큐브의 타입에 해당하는 객체가 들어가며, 배열 안에 여러 타입이 섞일 수 있음.

센티넬 자리는 세지 않음.

`'S'`이고 float/double 복셀값이 NaN 인 자리도 센티넬로 보고 빼놓음.

값을 넣지 않은 복셀은 0 이며 함께 봄.

셀 값이 하나도 없는 큐브는 `null`.

예제

시험 시나리오 TS-066-ARI(타입 혼합)의 입력 파일이 있는 폴더에서 실행.

```
TS-066-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-066-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        Object[] result = VoxelOpArithmetic.min(cubes);
```

```
        System.out.println("result: " + Arrays.toString(result));
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

원소가 `null` 이 아니어야 하고, byte, short, int, float, double 중 하나여야 함

Returns:

큐브별 최소값.

입력과 길이가 같고 순서도 같음.

빈 배열을 넘기면 빈 배열을 돌려줌

Throws:

`NullPointerException` - `cubes` 또는 그 원소가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입이 섞였을 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때

10.8. max API

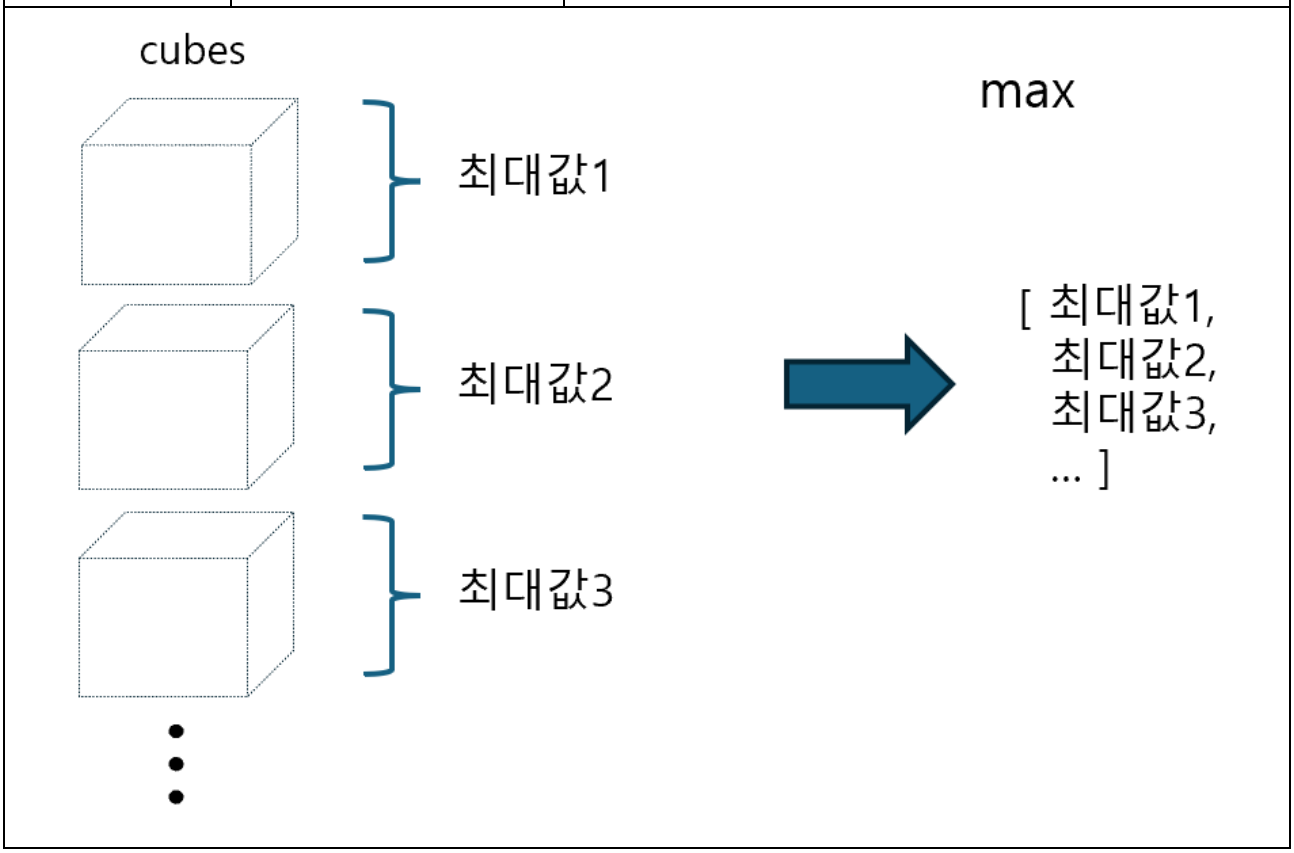
`public static Object[] max(BasicDataCube[] cubes)`

큐브마다 복셀값 중 가장 큰 값을 구함.

max

메서드	파라미터 / 반환	설명
max	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 모든 복셀값들 중에서 최대값 연산. (공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes)	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	Object[]	큐브별로 구해진 최대값들을 각 큐브의 데이터 타입에 해당하는 Object 리스트로 반환. 반환값의 순서는 입력 cubes 의 순서와 일치.

		Object: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체
		bit/V2/V3/V4 타입에 대한 지원은 미정.



원소마다 그 큐브의 타입에 해당하는 객체가 들어가며, 배열 안에 여러 타입이 섞일 수 있음.

센티넬 자리는 세지 않음.

`'S'`이고 float/double 복셀값이 NaN 인 자리도 센티넬로 보고 빼놓음.

값을 넣지 않은 복셀은 0 이며 함께 봄.

셀 값이 하나도 없는 큐브는 `null`.

예제

시험 시나리오 TS-069-ARI(타입 혼합)의 입력 파일이 있는 폴더에서 실행.

```
TS-069-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-069-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {
```

```
    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        Object[] result = VoxelOpArithmetic.max(cubes);
        System.out.println("result: " + Arrays.toString(result));
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

원소가 `null` 이 아니어야 하고, byte, short, int, float, double 중 하나여야 함

Returns:

큐브별 최대값.

입력과 길이가 같고 순서도 같음.

빈 배열을 넘기면 빈 배열을 돌려줌

Throws:

`NullPointerException` - `cubes` 또는 그 원소가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입이 섞였을 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때

See Also:

- `min(BasicDataCube[])`

10.9. variance API

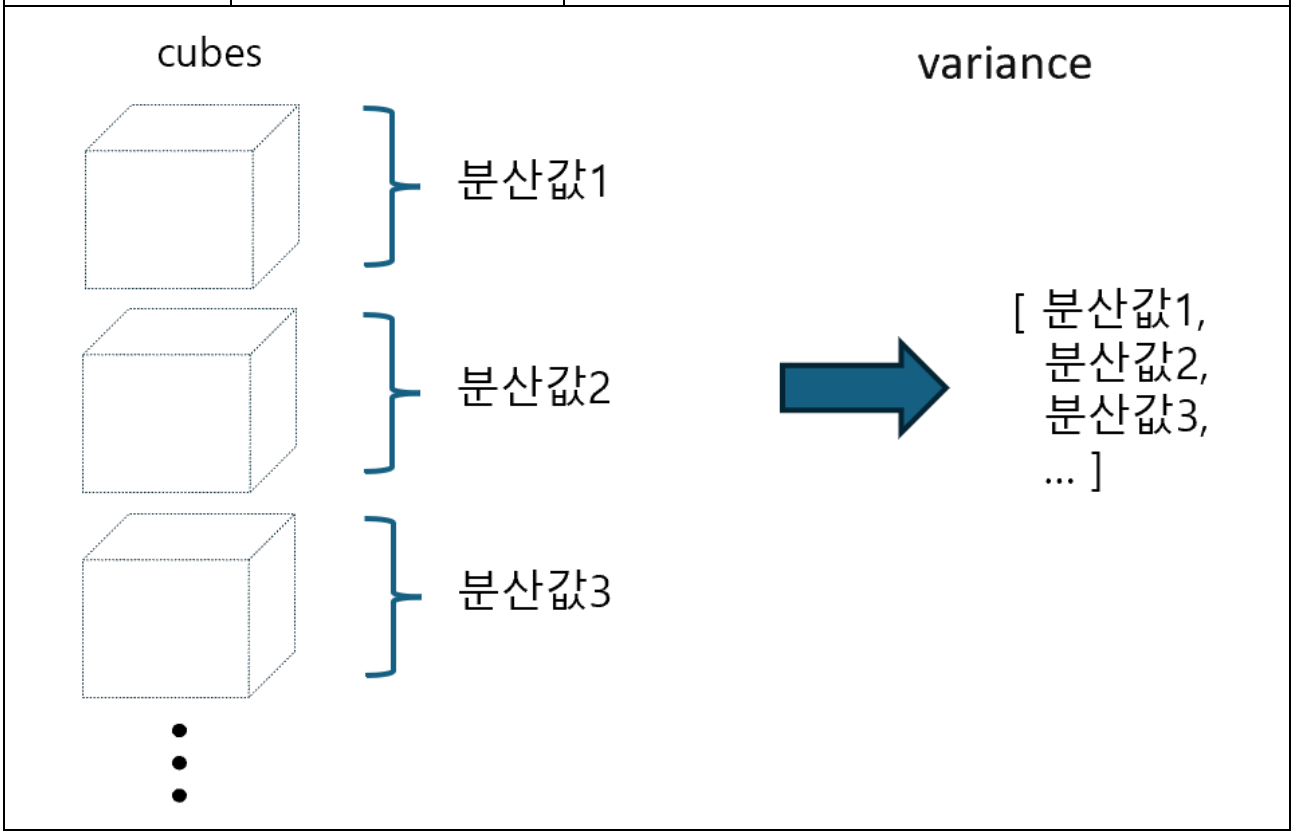
`public static double[] variance(BasicDataCube[] cubes)`

큐브마다 복셀값 분산을 구함.

variance

메서드	파라미터 / 반환	설명
variance	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 모든 복셀값들의 분산값 연산. (공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes)	레스터 데이터를 가지는 큐브 리스트.

		지원타입: byte,short,int,float,double
	double[]	큐브별로 구해진 분산값들을 double 리스트로 반환. 반환값의 순서는 입력 cubes 의 순서와 일치.
		bit/V2/V3/V4 타입에 대한 지원은 미정.



센티넬 자리는 세지 않음.

`'S'`이고 float/double 복셀값이 NaN 인 자리도 센티넬로 보고 빼놓음.

값을 넣지 않은 복셀은 0 이며 함께 봄.

셀 값이 하나도 없는 큐브는 `Double.NaN`.

예제

시험 시나리오 TS-075-ARI(타입 혼합)의 입력 파일이 있는 폴더에서 실행.

```
TS-075-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-075-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {
```

```
    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        double[] result = VoxelOpArithmetic.variance(cubes);
        System.out.println("result: " + Arrays.toString(result));
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

원소가 `null` 이 아니어야 하고, byte, short, int, float, double 중 하나여야 함

Returns:

큐브별 분산.

입력과 길이가 같고 순서도 같음.

빈 배열을 넘기면 빈 배열을 돌려줌

Throws:

`NullPointerException` - `cubes` 또는 그 원소가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입이 섞였을 때

`ArithmeticException` - 어느 큐브의 분산이나 평균이 double 을 넘을 때, 복셀값이 무한대일 때, `'V'`

큐브의 복셀값이 NaN 일 때

10.10. standardDeviation API

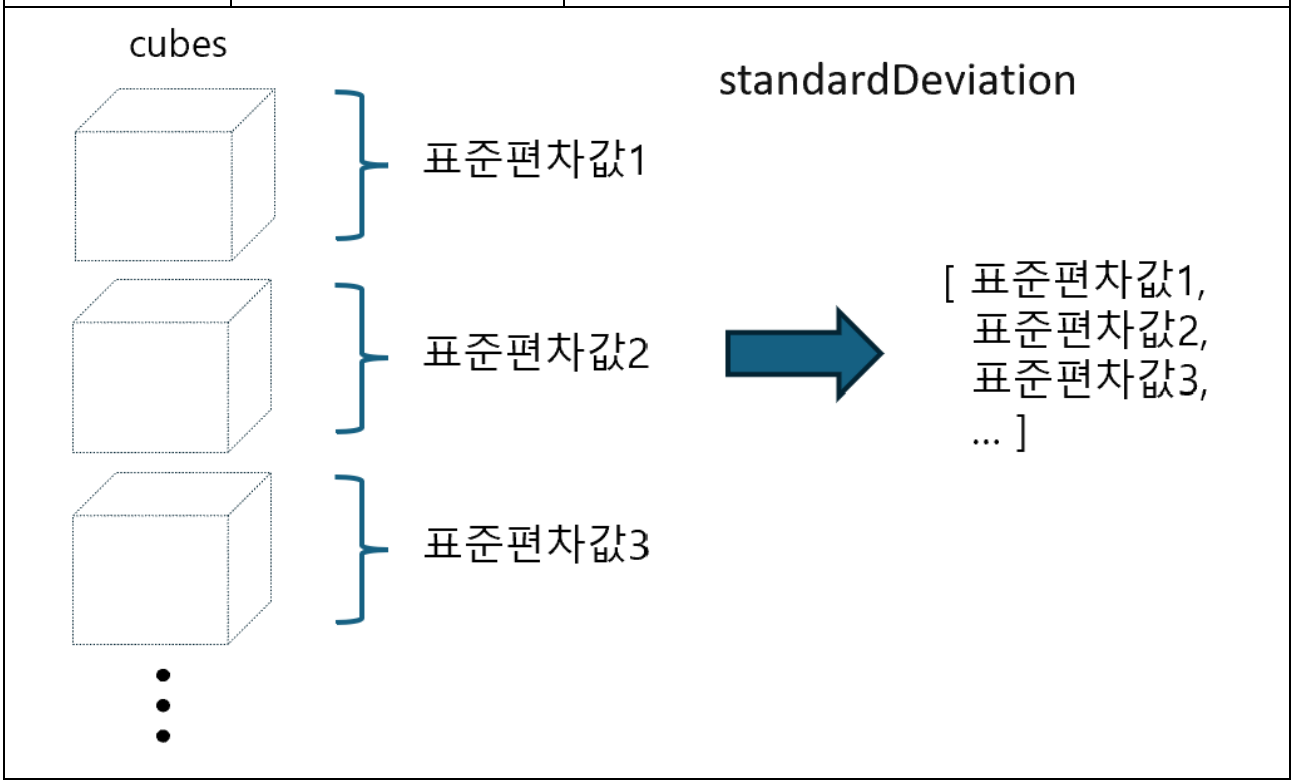
`public static double[] standardDeviation(BasicDataCube[] cubes)`

큐브마다 복셀값 표준편차를 구함.

standardDeviation

메서드	파라미터 / 반환	설명
standardDeviation	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 모든 복셀값들의 표준편차값 연산. (공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes)	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double

	double[]	큐브별로 구해진 표준편차값들을 double 리스트로 반환. 반환값의 순서는 입력 cubes 의 순서와 일치.
		bit/V2/V3/V4 타입에 대한 지원은 미정.



센티넬 자리는 세지 않음.

`'S'`이고 float/double 복셀값이 NaN 인 자리도 센티넬로 보고 빼놓음.

값을 넣지 않은 복셀은 0 이며 함께 봄.

셀 값이 하나도 없는 큐브는 `Double.NaN`.

예제

시험 시나리오 TS-072-ARI(타입 혼합)의 입력 파일이 있는 폴더에서 실행.

```
TS-072-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-072-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        double[] result = VoxelOpArithmetic.standardDeviation(cubes);
        System.out.println("result: " + Arrays.toString(result));
    }
```

```
    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }

}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

원소가 `null` 이 아니어야 하고, byte, short, int, float, double 중 하나여야 함

Returns:

큐브별 표준편차.

입력과 길이가 같고 순서도 같음.

빈 배열을 넘기면 빈 배열을 돌려줌

Throws:

`NullPointerException` - `cubes` 또는 그 원소가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입이 섞였을 때

`ArithmeticException` - 복셀값이 무한대일 때, `'V'` 큐브의 복셀값이 NaN 일 때

10.11. pow API

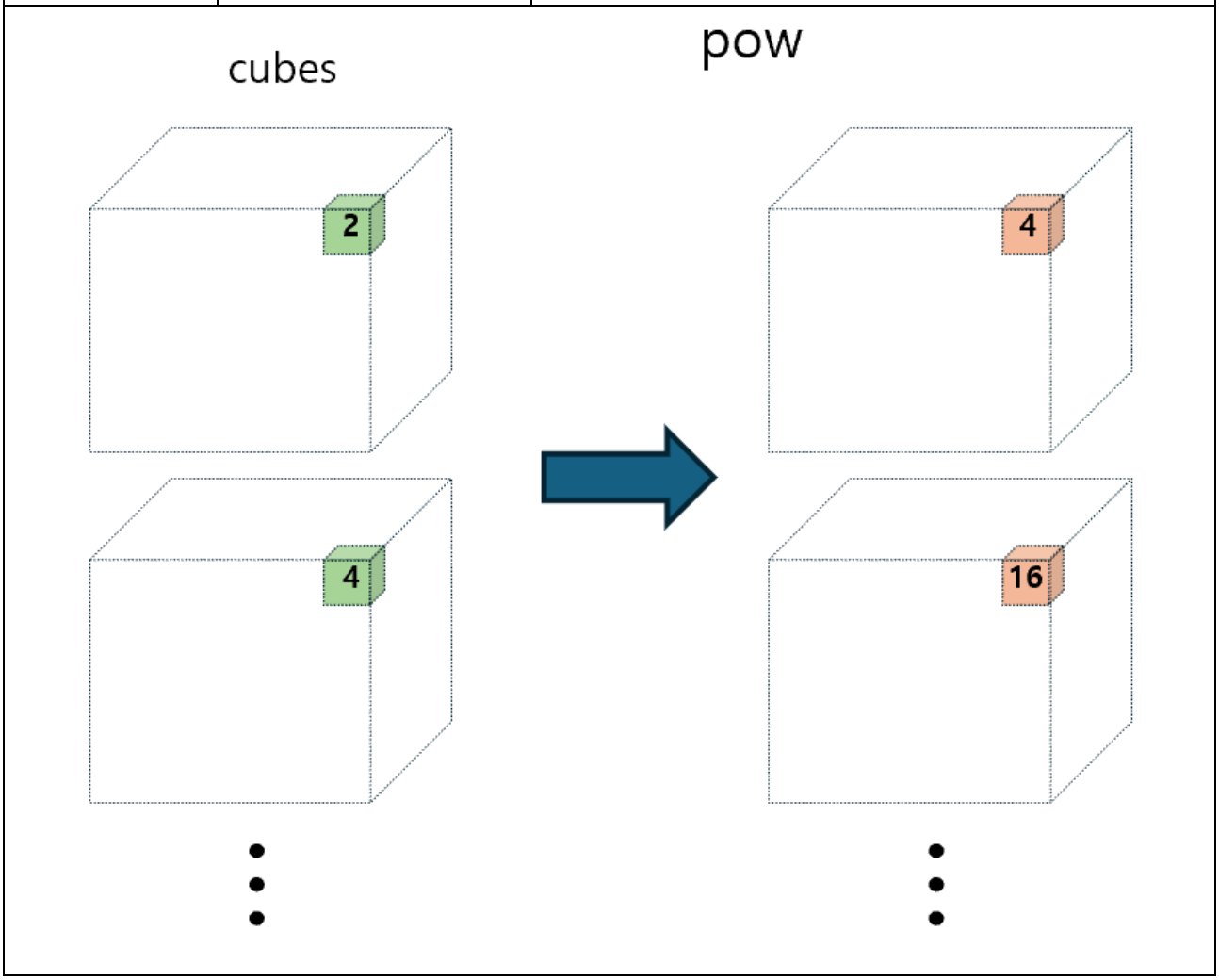
`public static BasicDataCube[] pow(BasicDataCube[] cubes)`

큐브마다 복셀값의 제곱을 구함.

pow

메서드	파라미터 / 반환	설명
pow	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 모든 복셀값들을 각 복셀별로 제곱값 연산. (공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes)	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 반환타입: 각 큐브별 데이터 타입에 해당하는 타입으로 반환. 반환큐브의 순서는 입력 cubes 의 순서와 일치.
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환.

		(byte->short, short->int, int->double, float->double, double->예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.



큐브끼리 계산하지 않고 큐브마다 따로 계산하므로 반환 타입도 큐브마다 정해짐.

예제

시험 시나리오 TS-076-ARI(short 제곱)의 입력 파일이 있는 폴더에서 실행.

```
TS-076-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-076-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
```

```
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpArithmetic.pow(cubes);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

원소가 `null` 이 아니어야 하고, byte, short, int, float, double 중 하나여야 함

Returns:

큐브별 제곱 결과.

입력과 길이가 같고 순서도 같음.

빈 배열을 넘기면 빈 배열을 돌려줌

Throws:

`NullPointerException` - `cubes` 또는 그 원소가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입이 섞였을 때

`ArithmeticException` - 어느 큐브의 결과가 double 에서도 넘칠 때, `'V'` 큐브의 복셀값이 NaN 일 때

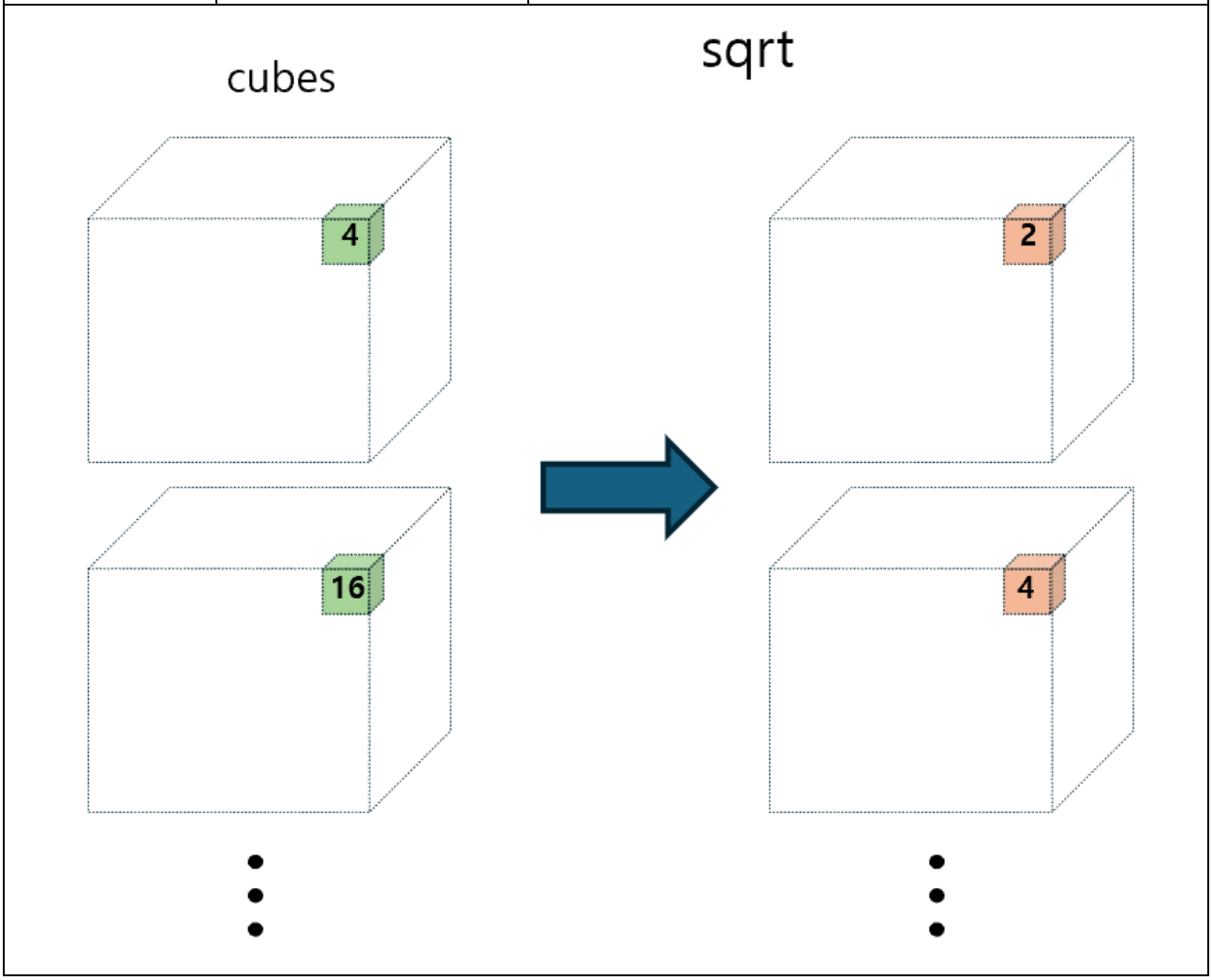
10.12. sqrt API

```
public static BasicDataCube[] sqrt(BasicDataCube[] cubes)
```

큐브마다 복셀값의 제곱근을 구함.

sqrt

메서드	파라미터 / 반환	설명
sqrt	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 모든 복셀값들을 각 복셀별로 제곱근값 연산. (공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes)	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 반환타입: double 반환큐브의 순서는 입력 cubes 의 순서와 일치.
		bit/V2/V3/V4 타입에 대한 지원은 미정.



반환 큐브는 입력 타입과 상관없이 모두 double 임.

예제

시험 시나리오 TS-079-ARI(int 제곱근)의 입력 파일이 있는 폴더에서 실행.

```
TS-079-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-079-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpArithmetic.sqrt(cubes);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

원소가 `null` 이 아니어야 하고, byte, short, int, float, double 중 하나여야 하며, 센티넬이 아닌 복셀값에 음수가 없어야 함

Returns:

큐브별 제곱근 결과.

입력과 길이가 같고 순서도 같음.

빈 배열을 넘기면 빈 배열을 돌려줌

Throws:

`NullPointerException` - `cubes` 또는 그 원소가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입이 섞였을 때

`ArithmeticException` - 어느 큐브의 복셀값이 음수이거나 무한대일 때, `'V'` 큐브의 복셀값이 NaN 일 때

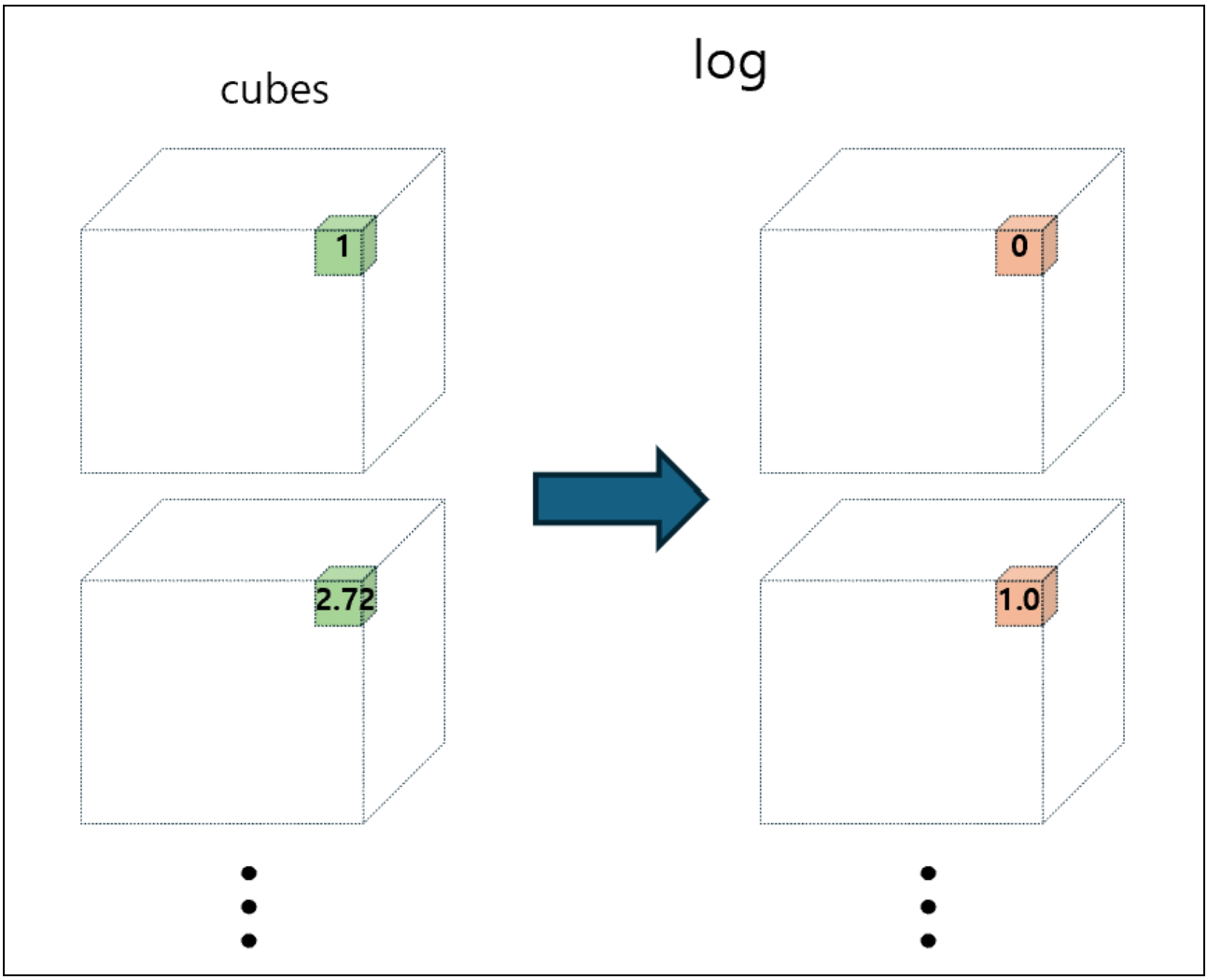
10.13. log API

```
public static BasicDataCube[] log(BasicDataCube[] cubes)
```

큐브마다 복셀값의 자연로그를 구함.

log

메서드	파라미터 / 반환	설명
log	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 모든 복셀값들을 각 복셀별로 자연로그값 연산. (공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes)	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 반환타입: double 반환큐브의 순서는 입력 cubes 의 순서와 일치.
		bit/V2/V3/V4 타입에 대한 지원은 미정.



반환 큐브는 입력 타입과 상관없이 모두 double 임.

예제

시험 시나리오 TS-082-ARI(int 로그)의 입력 파일이 있는 폴더에서 실행.

```
TS-082-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-082-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpArithmetic.log(cubes);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
```

```
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }

}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

원소가 `null` 이 아니어야 하고, byte, short, int, float, double 중 하나여야 하며, 센티넬이 아닌 복셀값이 모두 0 보다 커야 함

Returns:

큐브별 자연로그 결과.

입력과 길이가 같고 순서도 같음.

빈 배열을 넘기면 빈 배열을 돌려줌

Throws:

`NullPointerException` - `cubes` 또는 그 원소가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입이 섞였을 때

`ArithmeticException` - 어느 큐브의 복셀값이 0 이하이거나 무한대일 때, `'V'` 큐브의 복셀값이 NaN 일 때

10.14. sin API

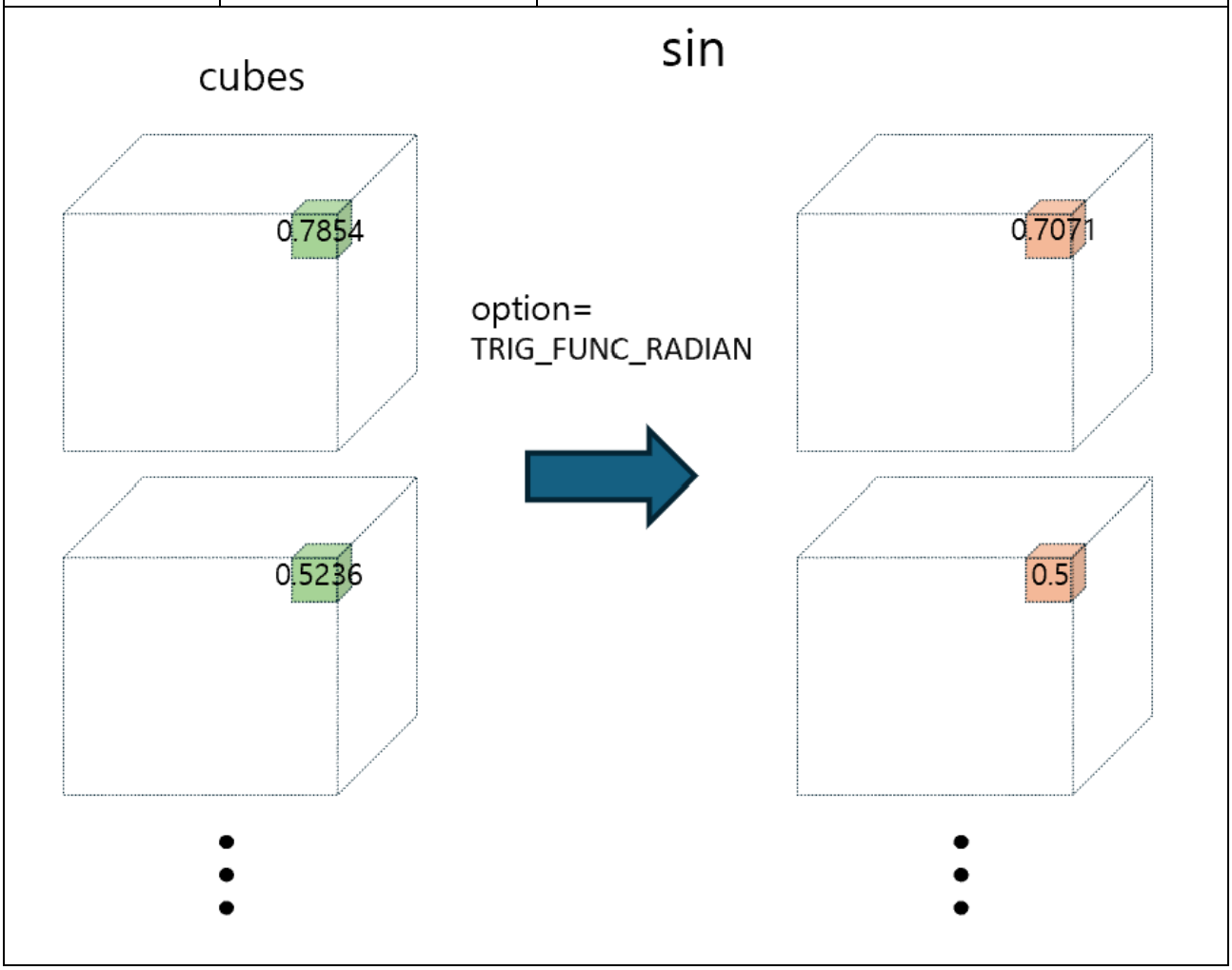
```
public static BasicDataCube[] sin(BasicDataCube[] cubes,
    int option)
```

큐브마다 복셀값의 싸인을 구함.

`option` 은 모든 큐브에 똑같이 적용됨.

sin

메서드	파라미터 / 반환	설명
sin	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 모든 복셀값들을 각 복셀별로 싸인값 연산. (공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	int option)	TRIG_FUNC_RADIAN : 입력 큐브의 복셀값을 라디안값으로 해석. TRIG_FUNC_DEGREE : 입력 큐브의 복셀값을 각도값으로 해석.
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 반환타입: double 반환큐브의 순서는 입력 cubes 의 순서와 일치.
		bit/V2/V3/V4 타입에 대한 지원은 미정.



반환 큐브는 입력 타입과 상관없이 모두 double 임.

예제

시험 시나리오 TS-085-ARI(라디안)의 입력 파일이 있는 폴더에서 실행.

```
TS-085-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-085-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpArithmetic.sin(cubes, VoxelOpArithmetic.TRIG_FUNC_RADIAN);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

원소가 `null` 이 아니어야 하고, byte, short, int, float, double 중 하나여야 함

`option` - `TRIG_FUNC_RADIAN` 또는 `TRIG_FUNC_DEGREE`

Returns:

큐브별 싸인 결과.

입력과 길이가 같고 순서도 같음.

빈 배열을 넘기면 빈 배열을 돌려줌

Throws:

`NullPointerException` - `cubes` 또는 그 원소가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입이 섞였을 때, `option` 이 위 두 값이 아닐 때

`ArithmeticException` - 어느 큐브의 복셀값이 무한대일 때, `'V'` 큐브의 복셀값이 NaN 일 때

10.15. cos API

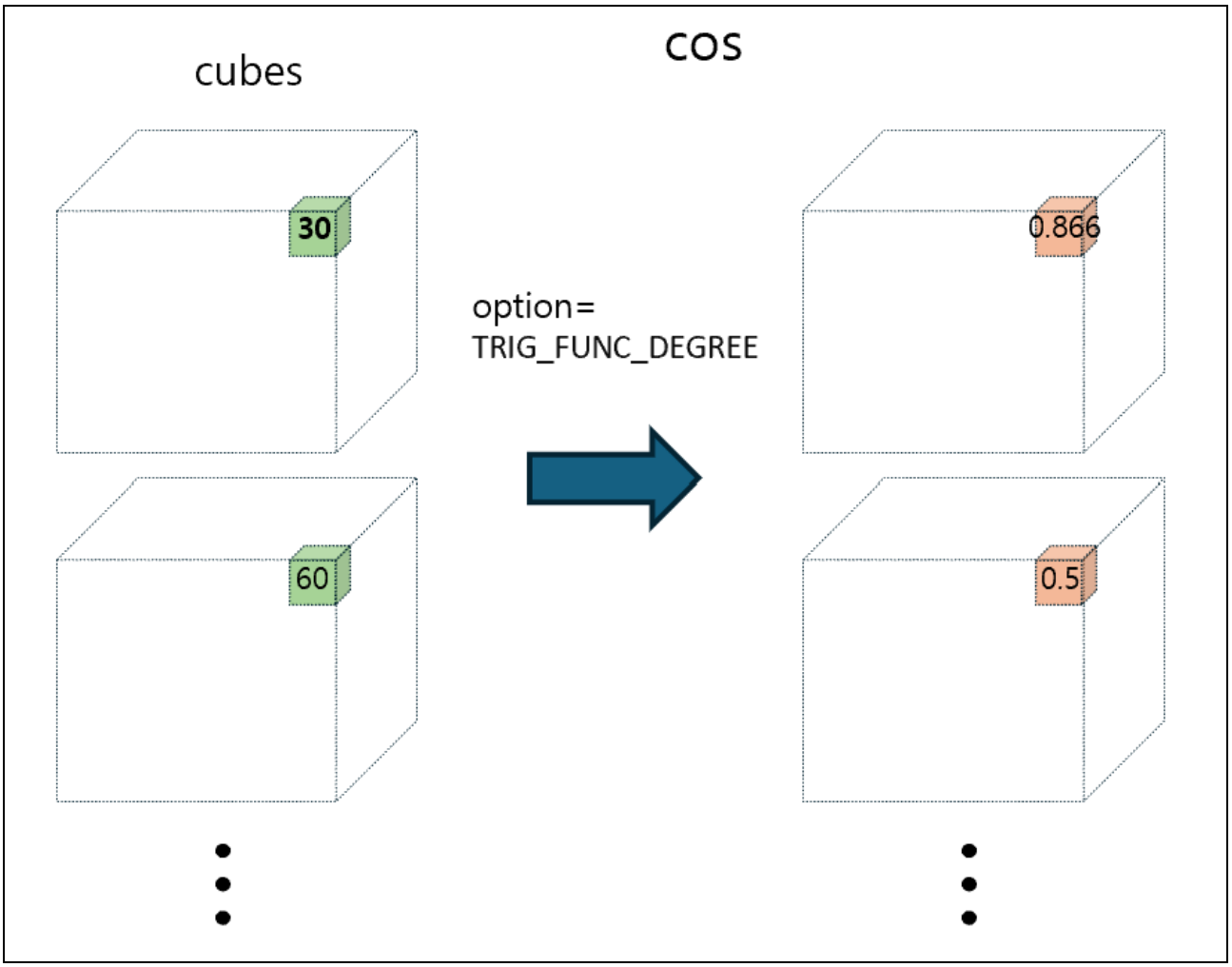
```
public static BasicDataCube[] cos(BasicDataCube[] cubes,
    int option)
```

큐브마다 복셀값의 코싸인을 구함.

`option` 은 모든 큐브에 똑같이 적용됨.

cos

메서드	파라미터 / 반환	설명
cos	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 모든 복셀값들을 각 복셀별로 코싸인값 연산. (공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	int option)	TRIG_FUNC_RADIAN : 입력 큐브의 복셀값을 라디안값으로 해석. TRIG_FUNC_DEGREE : 입력 큐브의 복셀값을 각도값으로 해석.
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 반환타입: double 반환큐브의 순서는 입력 cubes 의 순서와 일치.
		bit/V2/V3/V4 타입에 대한 지원은 미정.



반환 큐브는 입력 타입과 상관없이 모두 double 임.

예제

시험 시나리오 TS-088-ARI(라디안)의 입력 파일이 있는 폴더에서 실행.

```
TS-088-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-088-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpArithmetic.cos(cubes, VoxelOpArithmetic.TRIG_FUNC_RADIAN);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }
```

```
    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }

}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

원소가 `null` 이 아니어야 하고, byte, short, int, float, double 중 하나여야 함

`option` - `TRIG_FUNC_RADIAN` 또는 `TRIG_FUNC_DEGREE`

Returns:

큐브별 코싸인 결과.

입력과 길이가 같고 순서도 같음.

빈 배열을 넘기면 빈 배열을 돌려줌

Throws:

`NullPointerException` - `cubes` 또는 그 원소가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입이 섞였을 때, `option` 이 위 두 값이 아닐 때

`ArithmeticException` - 어느 큐브의 복셀값이 무한대일 때, `'V'` 큐브의 복셀값이 NaN 일 때

10.16. tan API

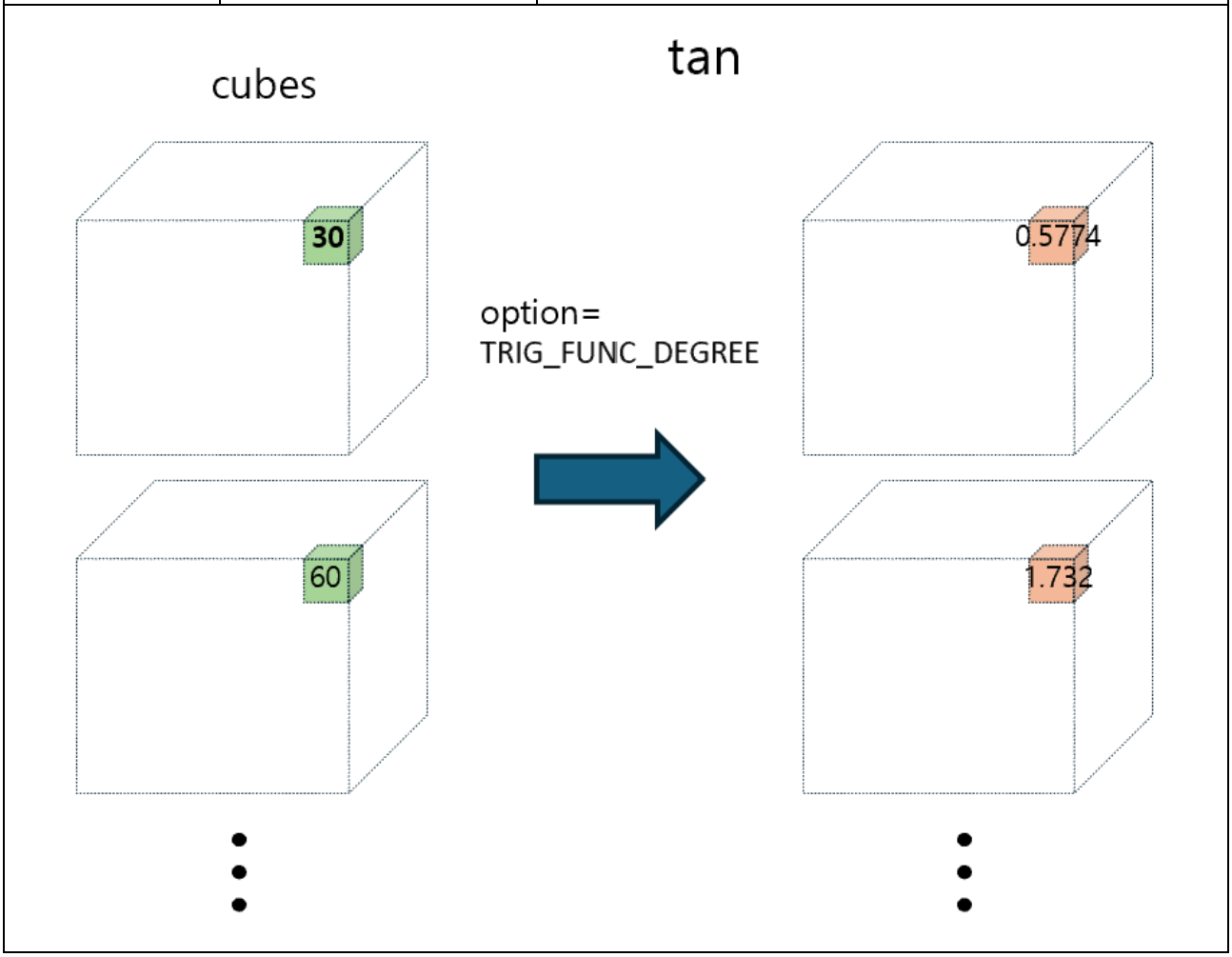
```
public static BasicDataCube[] tan(BasicDataCube[] cubes,
    int option)
```

큐브마다 복셀값의 탄젠트를 구함.

`option` 은 모든 큐브에 똑같이 적용됨.

tan

메서드	파라미터 / 반환	설명
tan	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 모든 복셀값들을 각 복셀별로 탄젠트값 연산. (공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	int option)	TRIG_FUNC_RADIAN : 입력 큐브의 복셀값을 라디안값으로 해석. TRIG_FUNC_DEGREE : 입력 큐브의 복셀값을 각도값으로 해석.
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 반환타입: double 반환큐브의 순서는 입력 cubes 의 순서와 일치.
		bit/V2/V3/V4 타입에 대한 지원은 미정.



반환 큐브는 입력 타입과 상관없이 모두 double 임.

예제

시험 시나리오 TS-091-ARI(라디안)의 입력 파일이 있는 폴더에서 실행.

```
TS-091-ARI>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-091-ARI>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpArithmetic;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpArithmetic.tan(cubes, VoxelOpArithmetic.TRIG_FUNC_RADIAN);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

원소가 `null` 이 아니어야 하고, byte, short, int, float, double 중 하나여야 함

`option` - `TRIG_FUNC_RADIAN` 또는 `TRIG_FUNC_DEGREE`

Returns:

큐브별 탄젠트 결과.

입력과 길이가 같고 순서도 같음.

빈 배열을 넘기면 빈 배열을 돌려줌

Throws:

`NullPointerException` - `cubes` 또는 그 원소가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입이 섞였을 때, `option` 이 위 두 값이 아닐 때

`ArithmeticException` - 어느 큐브의 복셀값이 무한대일 때, `'V'` 큐브의 복셀값이 NaN 일 때, 도 단위에서 센티넬이 아닌 값이 정확히 `90 + 180k` 일 때, 계산 결과가 double 범위를 넘을 때

11. VoxelOpComparison [레스터]

클래스 설명

```
public final class VoxelOpComparison
    extends Object
```

[레스터] 비교 연산자.

복셀값을 비교하여 BIT 큐브로 돌려줌.

인스턴스를 만들 수 없으며 비교 형태마다 정적 메서드를 제공함.

```
// 같은 자리 복셀끼리 비교
VoxelOpComparison.compare(cube1, cube2, VoxelOpComparison.COMPARE_OP_GREATER_THAN);


// 모든 복셀을 10 과 비교
VoxelOpComparison.compare(cube, (short) 10, VoxelOpComparison.COMPARE_OP_GREATER_THAN);


// 0 이상 100 미만인 복셀 찾기 (100 은 제외)
VoxelOpComparison.compareRange(cube, 0, 100);
```

상수와 비교할 때 `number` 자리에 `null` 을 그대로 넘기면 큐브끼리 비교하는 쪽으로 붙어 `NullPointerException` 이 됨.

상수 형태를 부르려면 `(Object) null` 로 넘겨야 함 — 다만 `number` 는 `null` 일 수 없으므로 어느 쪽이든 예외이며, 예외가 나오는 자리만 달라짐.

입력 큐브는 변경하지 않고 언제나 새 큐브를 만들어 돌려줌.

상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

비교 연산

`compareOp` 에 넘길 수 있는 값은 아래 여섯 개임.

compareOp 상수

상수	값	연산	`true` 가 되는 자리
COMPARE_OP_EQUAL	0	==	두 복셀값이 같을 때
COMPARE_OP_NOT_EQUAL	1	!=	두 복셀값이 다를 때
COMPARE_OP_GREATER_THAN	2	>	왼쪽 복셀값이 클 때
COMPARE_OP_LESS_THAN	3	<	왼쪽 복셀값이 작을 때
COMPARE_OP_GREATER_THAN_EQUAL	4	>=	왼쪽 복셀값이 크거나 같을 때
COMPARE_OP_LESS_THAN_EQUAL	5	<=	왼쪽 복셀값이 작거나

			같을 때

왼쪽은 큐브끼리 비교하면 `cube1`, 상수와 비교하면 `cube` 임.

즉 `COMPARE_OP_GREATER_THAN` 은 각각 `cube1 > cube2` 와 `cube > number` 를 뜻함.

값을 넣지 않은 복셀은 `0` 이며 그대로 비교함.

그래서 한쪽 귀퉁이만 채운 두 큐브를 `COMPARE_OP_EQUAL` 로 비교하면 나머지 자리가 모두 `0 == 0` 이 되어 `true` 가 됨.

결과의 `false` 는 뜻이 둘임 — 비교 조건이 성립하지 않았거나, 센티넬 자리라 비교하지 않았거나.

결과 큐브는 센티넬을 쓰지 않으므로 결과만으로는 둘을 구분할 수 없고, 구분하려면 입력 큐브의 `isSentinel(int)`를 다시 봐야 함.

지원 타입

`BasicDataCube.DataType.BIT`, `BasicDataCube.DataType.BYTE`, `BasicDataCube.DataType.SHORT`, `BasicDataCube.DataType.INT`, `BasicDataCube.DataType.FLOAT`, `BasicDataCube.DataType.DOUBLE`.

상수는 `Boolean`, `Byte`, `Short`, `Integer`, `Float`, `Double` 객체로 받음.

`Long` 은 받지 않음.

BIT 큐브와 `Boolean` 상수의 `false` 와 `true` 는 각각 수치 `0` 과 `1` 로 바꿔 비교함.

비교는 모두 `double` 로 하므로 큐브끼리 또는 큐브와 상수의 타입이 달라도 값끼리 비교됨.

결과 큐브의 헤더

결과 큐브가 갖는 값

헤더 필드	결과값
level, `(col, row, floor)`, resolution	입력 큐브(큐브끼리 비교는 `cube1`, 나머지는 `cube`)의 것을 그대로 물려받음
`datatype`	언제나 `BIT`
`dataBlockCode`	언제나 `'V'` — 입력이 `'S'`여도 결과는 센티넬을 쓰지 않음
`sentinel`	`0`. `sentinelObj()`는 `null`

센티넬과 NaN, 무한대

복셀 하나하나를 볼 때:

- `'S'`이고 복셀값이 센티넬이면 비교하지 않고 그 자리의 결과는 `false`
- `'S'`이고 float/double 복셀값이 NaN 이면 **센티넬 값이 무엇이든** 센티넬로 보아 비교하지 않고 그 자리의 결과는 `false`
- `'V'`이고 float/double 복셀값이 NaN 이면 `ArithmeticException`

- 두 큐브 중 한쪽만 센티넬이어도 그 자리는 비교하지 않음
- 상수는 센티넬을 갖지 않으므로, 상수 비교에서 센티넬 여부는 큐브만 보고 정함

무한대는 NaN 과 달리 예외가 아니라 값으로 비교하며, 이는 복셀값이든 상수든 같음 — `Float.POSITIVE_INFINITY` 는 어떤 유한한 값보다도 큼.

반면 상수가 NaN 이면 큐브를 보지 않고 `IllegalArgumentException` 을 던짐.

BIT 큐브에 센티넬을 지정하면 값 두 개 중 하나가 통째로 센티넬이 됨.

센티넬이 `true` 인 `'S'` BIT 큐브는 `true` 복셀이 모두 센티넬이므로 그 자리의 결과가 언제나 `false` 임.

Field Summary

Fields

Modifier and Type	Field	Description
`static final int`	`COMPARE_OP_EQUAL`	`compareOp` 값 — `==` 비교.
`static final int`	`COMPARE_OP_GREATER_THAN`	`compareOp` 값 — `>` 비교.
`static final int`	`COMPARE_OP_GREATER_THAN_EQUAL`	`compareOp` 값 — `>=` 비교.
`static final int`	`COMPARE_OP_LESS_THAN`	`compareOp` 값 — `<` 비교.
`static final int`	`COMPARE_OP_LESS_THAN_EQUAL`	`compareOp` 값 — `<=` 비교.
`static final int`	`COMPARE_OP_NOT_EQUAL`	`compareOp` 값 — `!=` 비교.

Method Summary

Modifier and Type	Method	Description
`static BasicDataCube`	`compare(BasicDataCube cube1, BasicDataCube cube2, int compareOp)`	같은 자리의 두 큐브 복셀값을 비교함.
`static BasicDataCube`	`compare(BasicDataCube cube, Object number, int compareOp)`	큐브의 복셀값마다 같은 상수와 비교함.
`static BasicDataCube`	`compareRange(BasicDataCube cube, Object minNumber, Object maxNumber)`	큐브의 복셀값마다 하한과 상한 사이에 드는지 봄.

Field Details

COMPARE_OP_EQUAL

```
public static final int COMPARE_OP_EQUAL
```

`compareOp` 값 — `==` 비교.

값은 `0`.

See Also:

- Constant Field Values

COMPARE_OP_NOT_EQUAL

```
public static final int COMPARE_OP_NOT_EQUAL
```

compareOp 값 — != 비교.

값은 1.

See Also:

- Constant Field Values

COMPARE_OP_GREATER_THAN

```
public static final int COMPARE_OP_GREATER_THAN
```

compareOp 값 — > 비교.

값은 2.

See Also:

- Constant Field Values

COMPARE_OP_LESS_THAN

```
public static final int COMPARE_OP_LESS_THAN
```

compareOp 값 — < 비교.

값은 3.

See Also:

- Constant Field Values

COMPARE_OP_GREATER_THAN_EQUAL

```
public static final int COMPARE_OP_GREATER_THAN_EQUAL
```

compareOp 값 — >= 비교.

값은 4.

See Also:

- Constant Field Values

COMPARE_OP_LESS_THAN_EQUAL

```
public static final int COMPARE_OP_LESS_THAN_EQUAL
```

compareOp 값 — <= 비교.

값은 5.

See Also:

- Constant Field Values

11.1. compare API

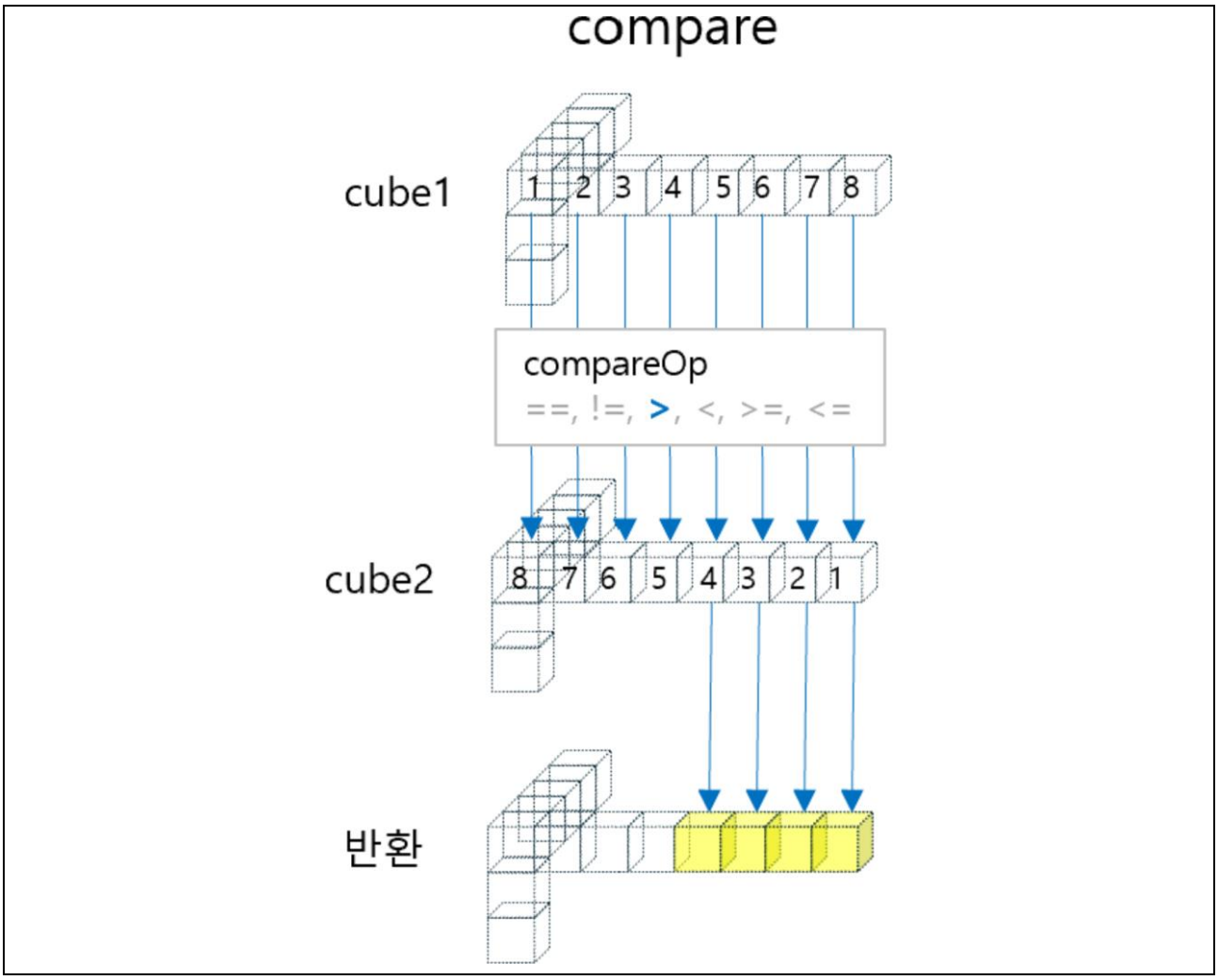
compare(BasicDataCube cube1, BasicDataCube cube2, int compareOp)

```
public static BasicDataCube compare(BasicDataCube cube1,
    BasicDataCube cube2,
    int compareOp)
```

같은 자리의 두 큐브 복셀값을 비교함.

compare

메서드	파라미터 / 반환	설명
compare	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 두 큐브의 복셀값들에 대한 비교 연산. 입력되는 두 큐브가 서로 다른 타입이어도 됨. (이후 연산도 마찬가지 임) 입력 큐브가 bit 일경우 false=0, true=1 로 수치화 하여 연산. (참고: 이후에 나오는 bit 큐브의 연산도 이와 같은 방식으로 처리)
	BasicDataCube cube1	레스터 데이터를 가지는 단일 큐브. 지원타입: bit,byte,short,int,float,double
	BasicDataCube cube2	레스터 데이터를 가지는 단일 큐브. 지원타입: bit,byte,short,int,float,double
	int compareOp)	COMPARE_OP_EQUAL : (==) 복셀값이 같은지 비교 COMPARE_OP_NOT_EQUAL : (!=) 복셀값이 다른지 비교 COMPARE_OP_GREATER_THAN : (>) 큐브 1 의 복셀값이 큰지 비교 COMPARE_OP_LESS_THAN : (<) 큐브 1 의 복셀값이 작은지 비교 COMPARE_OP_GREATER_THAN_EQUAL : (>=) 큐브 1 의 복셀값이 크거나같은지 비교 COMPARE_OP_LESS_THAN_EQUAL : (<=) 큐브 1 의 복셀값이 작거나같은지 비교
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: bit 입력 큐브의 데이터블록코드가 'S'일때: 입력큐브의 복셀값이 sentinel 인경우 반환큐브의 복셀값은 false 로 함 float,double 입력큐브 복셀값이 NaN 이면 반환큐브의 복셀값은 false 로 함 입력 큐브가 'V'이고, float 및 double 타입 일때 복셀값이 NaN 이면 예외처리 반환 큐브(bit)의 데이터블록코드는 'V'이며, 센티널은 0 임 (참고: 이후에 나오는 bit 타입의 반환 큐브도 이와같은 방식으로 처리)
		V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-094-CMP(int 두 필드 큼)의 입력 파일이 있는 폴더에서 실행.

```
TS-094-CMP>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
```

```
TS-094-CMP>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpComparison;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube cube1 = new BasicDataCube("cube1.bdc");
         BasicDataCube cube2 = new BasicDataCube("cube2.bdc");
         BasicDataCube result = VoxelOpComparison.compare(cube1, cube2,
VoxelOpComparison.COMPARE_OP_GREATER_THAN);
         System.out.println(result);
         result.writeToFile("result.bdc");
         System.out.println("saved: result.bdc");
     }


 }
```

Parameters:

`cube1` - 비교의 왼쪽 큐브.

레스터 데이터를 가지는 단일 큐브로, bit, byte, short, int, float, double 중 하나여야 함

`cube2` - 비교의 오른쪽 큐브.

`cube1` 과 level, `(col, row, floor)`, resolution 이 같아야 하고, bit, byte, short, int, float, double 중 하나여야 함

`compareOp` - `COMPARE_OP_EQUAL`, `COMPARE_OP_NOT_EQUAL`, `COMPARE_OP_GREATER_THAN`, `COMPARE_OP_LESS_THAN`, `COMPARE_OP_GREATER_THAN_EQUAL`, `COMPARE_OP_LESS_THAN_EQUAL` 중 하나

Returns:

비교 결과를 담은 새 BIT 큐브.

`true` 는 비교 조건이 성립한 자리, `false` 는 성립하지 않았거나 센티넬이라 비교하지 않은 자리임

Throws:

`NullPointerException` - `cube1` 또는 `cube2` 가 `null` 일 때

`IllegalArgumentException` - 두 큐브의 level, `(col, row, floor)`, resolution 이 다를 때, 지원하지 않는 큐브 타입일 때, `compareOp` 가 여섯 상수 중 하나가 아닐 때

`ArithmeticException` - `'V'` 큐브의 float/double 복셀값이 NaN 일 때

See Also:

- `compare(BasicDataCube, Object, int)`

compare(BasicDataCube cube, Object number, int compareOp)

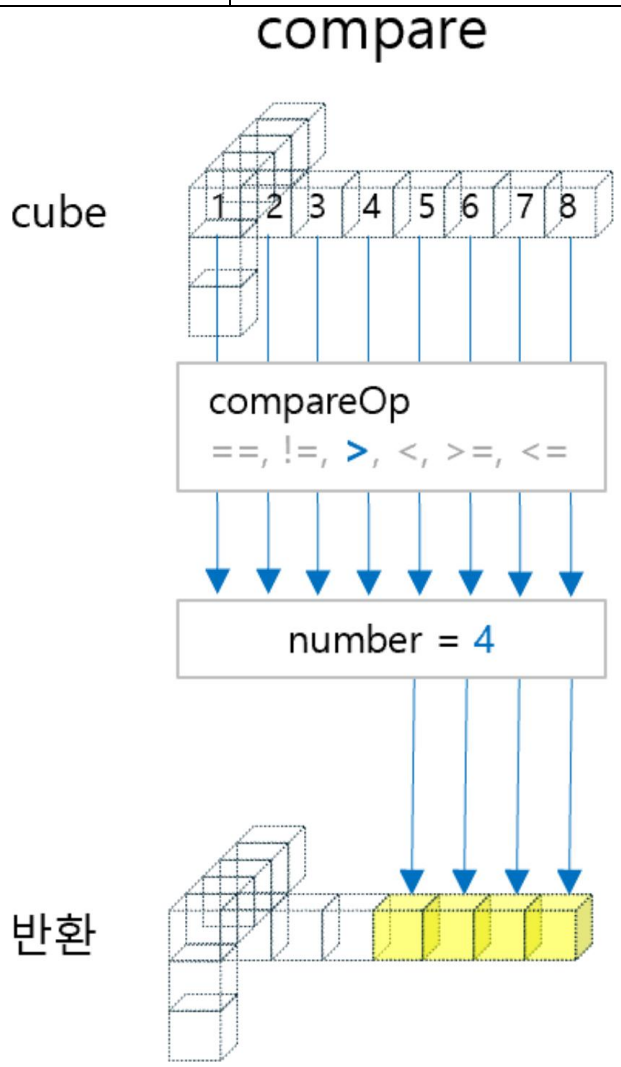
```
public static BasicDataCube compare(BasicDataCube cube,
    Object number,
    int compareOp)
```

큐브의 복셀값마다 같은 상수와 비교함.

compare

메서드	파라미터 / 반환	설명
compare	(	입력된 레스터 데이터 큐브내 모든 복셀값들을 number 값과 비교 연산. 입력되는 큐브와 상수가 서로 다른 타입이어도 됨. (이후 연산도 마찬가지 임) 입력 상수가 Boolean 일경우 false=0, true=1 로 수치화 하여 연산. (참고: 이후에 나오는 Boolean 상수의 연산도 이와 같은 방식으로 처리)
	BasicDataCube cube	레스터 데이터를 가지는 단일 큐브. 지원타입: bit,byte,short,int,float,double

	Object number	상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체
	int compareOp)	COMPARE_OP_EQUAL : (==) 복셀값이 같은지 비교 COMPARE_OP_NOT_EQUAL : (!=) 복셀값이 다른지 비교 COMPARE_OP_GREATER_THAN : (>) 큐브 1 의 복셀값이 큰지 비교 COMPARE_OP_LESS_THAN : (<) 큐브 1 의 복셀값이 작은지 비교 COMPARE_OP_GREATER_THAN_EQUAL : (>=) 큐브 1 의 복셀값이 크거나같은지 비교 COMPARE_OP_LESS_THAN_EQUAL : (<=) 큐브 1 의 복셀값이 작거나같은지 비교
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: bit
		V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-097-CMP(상수보다 큼)의 입력 파일이 있는 폴더에서 실행.

```
TS-097-CMP>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-097-CMP>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpComparison;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube cube = new BasicDataCube("cube.bdc");
         BasicDataCube result = VoxelOpComparison.compare(cube, 60,
VoxelOpComparison.COMPARE_OP_GREATER_THAN);
         System.out.println(result);
         result.writeToFile("result.bdc");
         System.out.println("saved: result.bdc");
     }


 }
```

Parameters:

`cube` - 레스터 데이터를 가지는 단일 큐브.

bit, byte, short, int, float, double 중 하나여야 함

`number` - 비교할 상수.

`Boolean`, `Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나여야 하고 NaN 이 아니어야 함.

`Long` 은 받지 않음

`compareOp` - `COMPARE_OP_EQUAL`, `COMPARE_OP_NOT_EQUAL`, `COMPARE_OP_GREATER_THAN`,

`COMPARE_OP_LESS_THAN`, `COMPARE_OP_GREATER_THAN_EQUAL`, `COMPARE_OP_LESS_THAN_EQUAL` 중 하나

Returns:

비교 결과를 담은 새 BIT 큐브.

`true` 는 비교 조건이 성립한 자리, `false` 는 성립하지 않았거나 센티넬이라 비교하지 않은 자리임

Throws:

`NullPointerException` - `cube` 또는 `number` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입일 때, `number` 가 위 여섯 가지 객체가 아닐 때,

`number` 가 NaN 일 때, `compareOp` 가 여섯 상수 중 하나가 아닐 때

`ArithmeticException` - `'V'` 큐브의 float/double 복셀값이 NaN 일 때

See Also:

- `compare(BasicDataCube, BasicDataCube, int)`

11.2. compareRange API

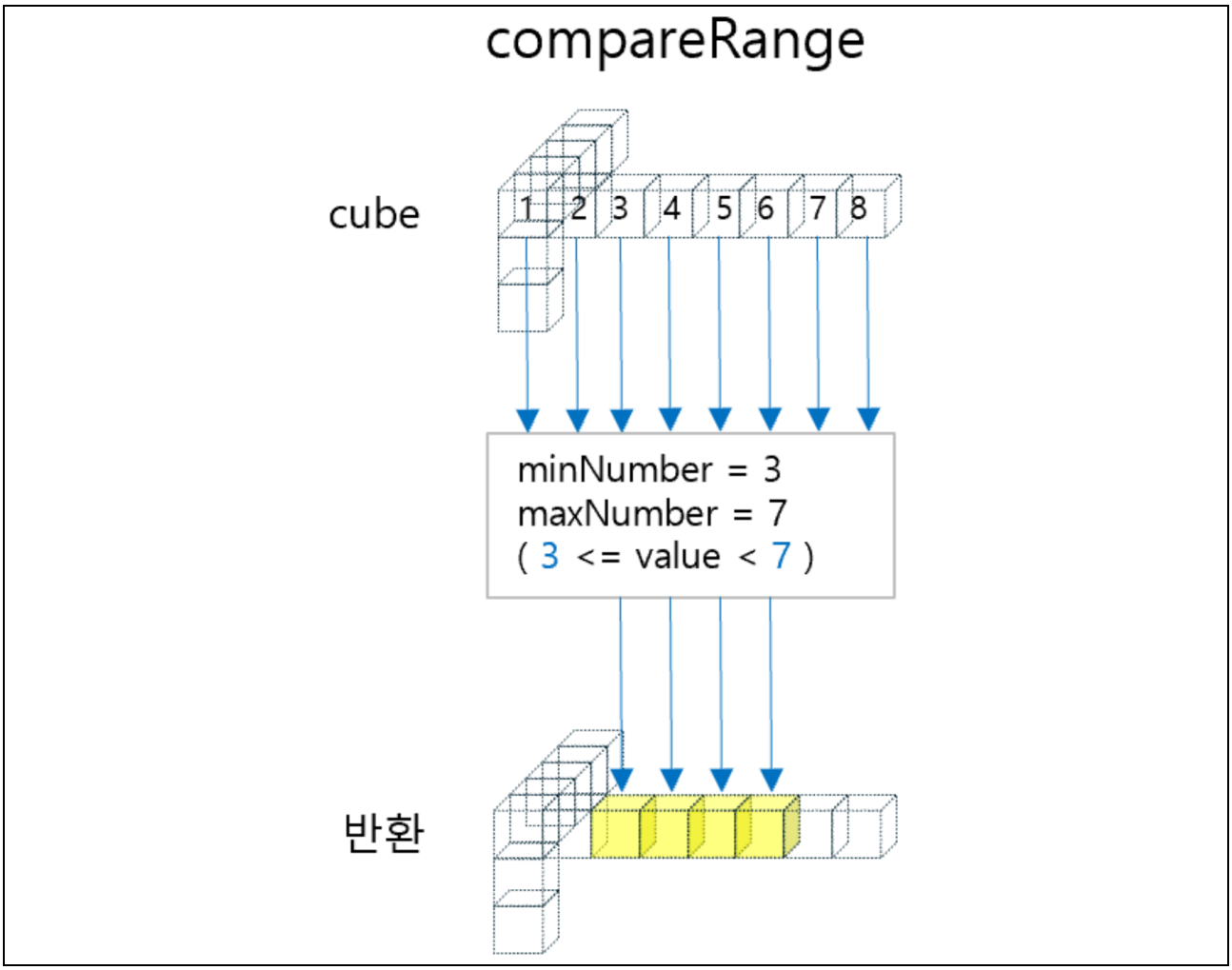
```
public static BasicDataCube compareRange(BasicDataCube cube,
    Object minNumber,
    Object maxNumber)
```

큐브의 복셀값마다 하한과 상한 사이에 드는지 봄.

하한은 포함하고 상한은 제외함 — `minNumber <= 복셀값 < maxNumber` 인 자리가 `true` 가 됨.

compareRange

메서드	파라미터 / 반환	설명
compareRange	(	입력된 레스터 데이터 큐브내 모든 복셀값들에 대해 범위(최소값 <= 복셀값 < 최대값) 비교 연산. (최소값>=최대값 인경우 예외처리) (이후 연산도 마찬가지 임) 입력되는 두 상수(min,max)가 서로 다른 타입이어도 됨. (이후 연산도 마찬가지 임)
	BasicDataCube cube	레스터 데이터를 가지는 단일 큐브. 지원타입: bit,byte,short,int,float,double
	Object minNumber	최소값 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber)	최대값 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: bit
		V2/V3/V4 타입에 대한 지원은 미정.



한쪽을 `null` 로 주면 그쪽 제한이 없다는 뜻임.

`minNumber` 가 `null` 이면 상한만, `maxNumber` 가 `null` 이면 하한만 봄.

둘 다 `null` 이면 센티넬이 아닌 모든 자리가 `true` 가 됨.

둘 다 주었는데 `minNumber >= maxNumber` 이면 드는 값이 있을 수 없으므로 큐브를 보지 않고 `IllegalArgumentException` 을 던짐.

두 상수는 서로 타입이 달라도 되고, 큐브 타입과 달라도 됨.

예제

시험 시나리오 TS-100-CMP(범위)의 입력 파일이 있는 폴더에서 실행.

```
TS-100-CMP>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-100-CMP>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpComparison;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube = new BasicDataCube("cube.bdc");
        BasicDataCube result = VoxelOpComparison.compareRange(cube, 40, 70);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
```

```
    }

}
```

Parameters:

`cube` - 레스터 데이터를 가지는 단일 큐브.

bit, byte, short, int, float, double 중 하나여야 함

`minNumber` - 하한.

이 값을 **포함**한다(`minNumber <= 복셀값`).

`Boolean`, `Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나여야 하고 NaN 이 아니어야 함.

`null` 이면 하한 제한이 없음

`maxNumber` - 상한.

이 값을 **제외**한다(`복셀값 < maxNumber`).

`Boolean`, `Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나여야 하고 NaN 이 아니어야 함.

`null` 이면 상한 제한이 없음

Returns:

범위에 드는지를 담은 새 BIT 큐브.

`true` 는 범위에 든 자리, `false` 는 들지 않았거나 센티넬이라 비교하지 않은 자리임

Throws:

`NullPointerException` - `cube` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 큐브 타입일 때, `minNumber` 또는 `maxNumber` 가 위 여섯 가지 객체가 아닐 때, 그 값이 NaN 일 때, 둘 다 주었는데 `minNumber >= maxNumber` 일 때

`ArithmeticException` - `'V'` 큐브의 float/double 복셀값이 NaN 일 때

See Also:

- `compare(BasicDataCube, Object, int)`

12. VoxelOpLogical [레스터]

클래스 설명

```
public final class VoxelOpLogical
    extends Object
```

[레스터] 논리 연산자.

인스턴스를 만들 수 없으며 정적 메서드를 제공함.

옵션 상수

옵션 값은 여섯 개가 모두 다름.

그래서 쓰임이 다른 상수를 잘못 넘기면 조용히 넘어가지 않고 `IllegalArgumentException` 으로 걸러짐.

option 상수

상수	값	쓰임
`MASK_PASS_TRUE`	`1`	마스크로 복셀값을 통과시킬 때 — `mask`
`MASK_PASS_FALSE`	`2`	
`MASK_CHANGE_TRUE`	`3`	마스크로 고른 복셀값을 범위 조건에 따라 바꿀 때 — `mask`
`MASK_CHANGE_FALSE`	`4`	
`CONDITION_PASS_TRUE`	`5`	복셀값의 범위를 참/거짓으로 환산할 때 — `and`, `or`, `xor`, `not`
`CONDITION_PASS_FALSE`	`6`	

지원 타입

레스터 큐브(`cube`, `cube1`, `cube2`)는 bit, byte, short, int, float, double 을 받음.

한 연산에 큐브가 둘 들어가도 datatype 은 서로 달라도 됨.

마스크 큐브(`mask`)는 **bit 만** 받음.

한 연산에 들어가는 큐브들은 level, `(col, row, floor)`, resolution 이 모두 같아야 함.

상수 파라미터

상수 파라미터가 받는 타입과 null 의 뜻

파라미터	받는 타입	`null` 일 때
`initNumber` `changeNumber`	`cube` 의 타입에 해당하는 객체 — bit 는 `Boolean`, byte 는 `Byte`, short 는 `Short`, int 는 `Integer`,	`cube` 의 `sentinel` 값으로 채움. `cube` 의 `dataBlockCode` 가 `'V'`이면 채울 값이 없어 `IllegalArgumentException`

	float 는 `Float`, double 은 `Double`	
`minNumber` `maxNumber`	`Boolean`, `Byte`, `Short`, `Integer`, `Float`, `Double` 중 아무거나. `cube` 의 타입과 같지 않아도 됨. NaN 은 받지 않음	그쪽 범위 제한이 없음

조건 논리연산

`and`, `or`, `xor`, `not` 은 큐브마다 `minNumber <= 복셀값 < maxNumber` 인지 보고, `option` 으로 그 판정을 참으로 읽을지 거짓으로 읽을지 정한 뒤 논리연산으로 묶음.
하한은 포함하고 상한은 포함하지 않으며, 한쪽이 `null` 이면 그쪽 제한이 없음.
결과는 bit 큐브임.

센티넬

`mask` 의 `sentinel` 설정은 보지 않음.
마스크는 비트값만으로 통과·대상 여부를 정함.

조건 논리연산에서는 입력 중 하나라도 센티넬인 자리의 결과가 `option` 과 상관없이 `false` 임.
`not` 에서도 마찬가지로 뒤집히지 않음.

결과 큐브의 헤더

마스크 연산의 결과는 level, `(col, row, floor)`, resolution, `datatype`, `dataBlockCode`, `sentinel` 을 모두 `cube` 의 것을 물려받음.

조건 논리연산의 결과는 **bit 큐브**이며 센티넬이 없음 (`dataBlockCode` 가 `'V'`).
level, `(col, row, floor)`, resolution 은 `cube1`(또는 `cube`)의 것을 물려받음.

입력 큐브는 어느 것도 변경하지 않음.

스레드 안전성

모든 메서드가 상태를 갖지 않아 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

Field Summary

Fields

Modifier and Type	Field	Description
`static final int`	`CONDITION_PASS_FALSE`	`option` 값 — 범위를 만족하는 복셀을 **`false`** 로 인식함.
`static final int`	`CONDITION_PASS_TRUE`	`option` 값 — 범위를 만족하는 복셀을 **`true`** 로 인식함.
`static final int`	`MASK_CHANGE_FALSE`	`option` 값 — 마스크값이 **`false`** 인 자리를 값

		변경의 대상으로 삼음.
static final int	MASK_CHANGE_TRUE	option 값 — 마스크값이 **true** 인 자리를 값 변경의 대상으로 삼음.
static final int	MASK_PASS_FALSE	option 값 — 마스크값이 **false** 인 자리만 복셀값이 그대로 살아남음.
static final int	MASK_PASS_TRUE	option 값 — 마스크값이 **true** 인 자리만 복셀값이 그대로 살아남음.

Method Summary

Modifier and Type	Method	Description
static BasicDataCube	and(BasicDataCube cube1, Object minNumber1, Object maxNumber1, int option1, BasicDataCube cube2, Object minNumber2, Object maxNumber2, int option2)	두 큐브의 범위 조건을 각각 계산한 뒤 AND 로 묶음.
static BasicDataCube	mask(BasicDataCube cube, BasicDataCube mask, int option, Object initNumber)	마스크가 통과시킨 복셀값만 남기고 막힌 자리는 초기값으로 채움.
static BasicDataCube	mask(BasicDataCube cube, BasicDataCube mask, int option, Object minNumber, Object maxNumber, Object changeNumber)	마스크로 고른 복셀 중 범위 조건에 드는 것만 새 값으로 바꿈.
static BasicDataCube	not(BasicDataCube cube, Object minNumber, Object maxNumber, int option)	한 큐브의 범위 조건을 계산한 뒤 뒤집음.
static BasicDataCube	or(BasicDataCube cube1, Object minNumber1, Object maxNumber1, int option1, BasicDataCube cube2, Object minNumber2, Object maxNumber2, int option2)	두 큐브의 범위 조건을 각각 계산한 뒤 OR 로 묶음.
static BasicDataCube	xor(BasicDataCube cube1, Object minNumber1, Object maxNumber1, int option1, BasicDataCube cube2, Object minNumber2, Object maxNumber2, int option2)	두 큐브의 범위 조건을 각각 계산한 뒤 XOR 로 묶음.

Field Details

MASK_PASS_TRUE

```
public static final int MASK_PASS_TRUE
```

option 값 — 마스크값이 **true** 인 자리만 복셀값이 그대로 살아남음.

값은 1.

See Also:

- Constant Field Values

MASK_PASS_FALSE

```
public static final int MASK_PASS_FALSE
```

option 값 — 마스크값이 **false** 인 자리만 복셀값이 그대로 살아남음.

값은 2.

See Also:

- Constant Field Values

MASK_CHANGE_TRUE

```
public static final int MASK_CHANGE_TRUE
```

option 값 — 마스크값이 **true** 인 자리를 값 변경의 대상으로 삼음.

값은 3.

See Also:

- Constant Field Values

MASK_CHANGE_FALSE

```
public static final int MASK_CHANGE_FALSE
```

option 값 — 마스크값이 **false** 인 자리를 값 변경의 대상으로 삼음.

값은 4.

See Also:

- Constant Field Values

CONDITION_PASS_TRUE

```
public static final int CONDITION_PASS_TRUE
```

option 값 — 범위를 만족하는 복셀을 **true** 로 인식함.

값은 5.

See Also:

- Constant Field Values

CONDITION_PASS_FALSE

```
public static final int CONDITION_PASS_FALSE
```

option 값 — 범위를 만족하는 복셀을 **false** 로 인식함.

값은 6.

See Also:

- Constant Field Values

12.1. mask API

mask(BasicDataCube cube, BasicDataCube mask, int option, Object initNumber)

```
public static BasicDataCube mask(BasicDataCube cube,
    BasicDataCube mask,
```

```
    int option,
    Object initNumber)
```

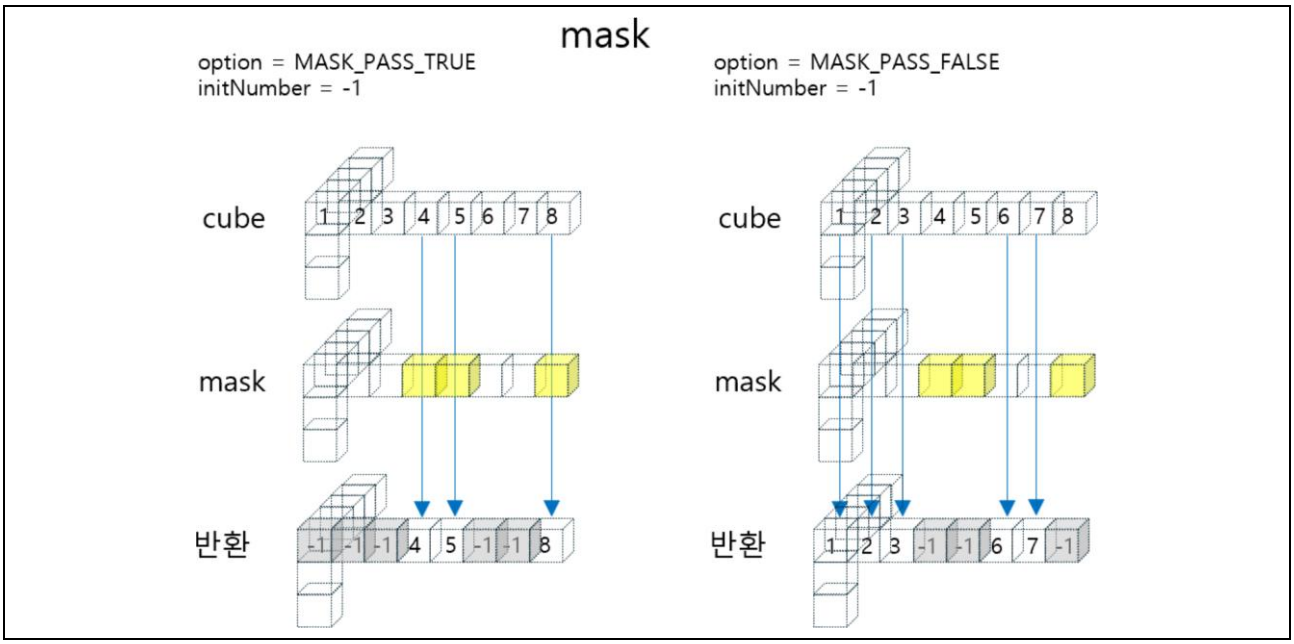
마스크가 통과시킨 복셀값만 남기고 막힌 자리는 초기값으로 채움.

`option` 으로 마스크값이 `true` 인 자리를 통과시킬지 `false` 인 자리를 통과시킬지 고름.

통과한 자리는 `cube` 의 복셀값이 그대로 남고, **막힌 자리는 `initNumber` 로 채움.**

mask

메서드	파라미터 / 반환	설명
mask	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 두 큐브에서, 레스터 큐브에 마스크 큐브를 적용하여 특정 복셀 데이터만 추출.
	BasicDataCube cube	레스터 데이터를 가지는 단일 큐브. 지원타입: bit,byte,short,int,float,double
	BasicDataCube mask	마스크 데이터를 가지는 단일 큐브. 지원타입: bit
	int option	MASK_PASS_TRUE : 마스크값이 1 인경우만 통과되어 복셀값이 그대로 살아남음 MASK_PASS_FALSE : 마스크값이 0 인경우만 통과되어 복셀값이 그대로 살아남음
	Object initNumber)	마스크에 의해 차단된 복셀들을 어떤값으로 초기화 할지 정하는 초기값 초기값은 cube 가 가지는 데이터 타입에 따라 아래와 같은 객체를 사용 이 값이 null 일때: cube 가 'V'이면 예외처리, 'S'이면 cube 의 sentinel 값으로 초기화 초기값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: cube 의 데이터 타입에 해당하는 타입으로 반환.
		V2/V3/V4 타입에 대한 지원은 미정.




```
// terrain: float 레스터 큐브
BasicDataCube high = VoxelOpComparison.compare(
        terrain, 100.0f, VoxelOpComparison.COMPARE_OP_GREATER_THAN);
BasicDataCube result = VoxelOpLogical.mask(
        terrain, high, VoxelOpLogical.MASK_PASS_TRUE, 0.0f);
```

`cube` 의 센티넬 자리도 다른 복셀과 똑같이 다룸 — 통과하면 센티넬 값이 그대로 옮겨지고, 막히면 `initNumber` 로 채움.

NaN 과 무한대도 값 그대로 옮김.

예제

시험 시나리오 TS-103-LOG(토러스 통과)의 입력 파일이 있는 폴더에서 실행.

```
TS-103-LOG>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-103-LOG>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpLogical;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube = new BasicDataCube("cube.bdc");
        BasicDataCube mask = new BasicDataCube("mask.bdc");
        BasicDataCube result = VoxelOpLogical.mask(cube, mask, VoxelOpLogical.MASK_PASS_TRUE, 0);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


}
```

Parameters:

`cube` - 레스터 데이터를 가지는 단일 큐브.

bit, byte, short, int, float, double 중 하나여야 함

`mask` - 마스크 데이터를 가지는 단일 큐브.

bit 타입이어야 하고 `cube` 와 level, `(col, row, floor)`, resolution 이 같아야 함

`option` - `MASK_PASS_TRUE`(값 `1`) 또는 `MASK_PASS_FALSE`(값 `2`) 중 하나.

`MASK_PASS_TRUE` 이면 마스크값이 `true` 인 자리가, `MASK_PASS_FALSE` 이면 `false` 인 자리가 통과함

`initNumber` - 막힌 자리를 채울 초기값.

`cube` 의 타입에 해당하는 객체여야 함.

`null` 이면 `cube` 의 `sentinel` 값으로 채움

Returns:

마스크를 적용한 새 큐브

Throws:

`NullPointerException` - `cube` 또는 `mask` 가 `null` 일 때

`IllegalArgumentException` - 다음 중 하나일 때

- 두 큐브의 level, `(col, row, floor)`, resolution 이 다름
- `cube` 가 bit, byte, short, int, float, double 이 아님
- `mask` 가 bit 가 아님
- `initNumber` 가 `cube` 의 타입에 해당하는 객체가 아님
- `initNumber` 가 `null` 인데 `cube` 의 `dataBlockCode` 가 `'V'`
- `option` 이 `MASK_PASS_TRUE`, `MASK_PASS_FALSE` 중 하나가 아님 (`MASK_CHANGE_TRUE` 처럼 쓰임이 다른 상수를 넘긴 경우 포함)

mask(BasicDataCube cube, BasicDataCube mask, int option, Object minNumber, Object maxNumber, Object changeNumber)

```
public static BasicDataCube mask(BasicDataCube cube,
    BasicDataCube mask,
    int option,
    Object minNumber,
    Object maxNumber,
    Object changeNumber)
```

마스크로 고른 복셀 중 범위 조건에 드는 것만 새 값으로 바꿈.

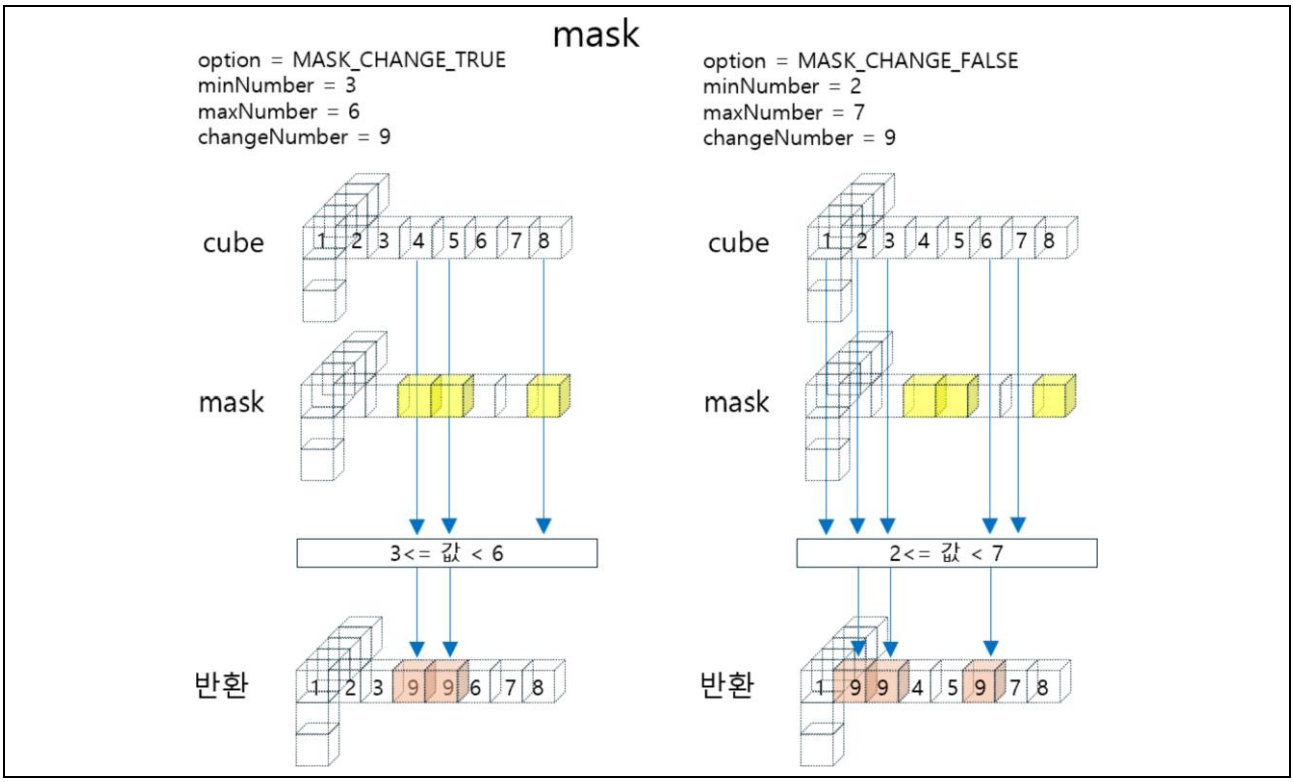
`option` 으로 마스크값이 `true` 인 자리를 대상으로 삼을지 `false` 인 자리를 대상으로 삼을지 고름.

대상이면서 범위 조건을 만족하는 자리만 `changeNumber` 로 바뀌고, 나머지는 `cube` 의 값이 그대로 남음.

통과형 `mask` 와 달리 대상이 아닌 자리는 초기값으로 덮이지 않고 원래 값이 남음.

mask

메서드	파라미터 / 반환	설명
mask	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 두 큐브에서, 레스터 큐브에 마스크 큐브를 적용하여 특정 복셀 데이터만을 대상으로 일정범위 조건에 맞으면 새로운 값으로 채움.
	BasicDataCube cube	레스터 데이터를 가지는 단일 큐브. 지원타입: bit,byte,short,int,float,double
	BasicDataCube mask	마스크 데이터를 가지는 단일 큐브. 지원타입: bit
	int option	MASK_CHANGE_TRUE : 마스크값이 1 인경우에 해당하는 복셀만을 대상으로 함 MASK_CHANGE_FALSE : 마스크값이 0 인경우에 해당하는 복셀만을 대상으로 함
	Object minNumber	최소값 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber	최대값 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	Object changeNumber)	대상이 되는 복셀들중 일정범위 조건을 만족하는 복셀들에게 적용할 변경값 변경값은 cube 가 가지는 데이터 타입에 따라 아래와 같은 객체를 사용 변경값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: cube 의 데이터 타입에 해당하는 타입으로 반환.
		V2/V3/V4 타입에 대한 지원은 미정.




```
// terrain: float 레스터 큐브, downtown: bit 마스크 큐브
BasicDataCube result = VoxelOpLogical.mask(
        terrain, downtown, VoxelOpLogical.MASK_CHANGE_TRUE,
        0.0f, 100.0f, -1.0f);   // 마스크가 1 인 자리 중 0 이상 100 미만을 -1 로
```

범위는 **하한을 포함하고 상한은 포함하지 않음** — `minNumber <= 복셀값 < maxNumber`. 한쪽이 `null` 이면 그쪽 제한이 없고, 둘 다 `null` 이면 대상이 모두 바뀜.

`cube` 의 센티넬 자리는 대상에서 빠지고 그대로 옮겨짐.

무한대는 값으로 보고 범위를 판정함.

예제

시험 시나리오 TS-106-LOG(토러스 안 범위 변경)의 입력 파일이 있는 폴더에서 실행.

```
TS-106-LOG>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-106-LOG>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpLogical;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube = new BasicDataCube("cube.bdc");
        BasicDataCube mask = new BasicDataCube("mask.bdc");
        BasicDataCube result = VoxelOpLogical.mask(cube, mask, VoxelOpLogical.MASK_CHANGE_TRUE, 40,
60, 99);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }
```

```
}
```

Parameters:

`cube` - 레스터 데이터를 가지는 단일 큐브.

bit, byte, short, int, float, double 중 하나여야 함

`mask` - 마스크 데이터를 가지는 단일 큐브.

bit 타입이어야 하고 `cube` 와 level, `(col, row, floor)`, resolution 이 같아야 함

`option` - `MASK_CHANGE_TRUE`(값 `3`) 또는 `MASK_CHANGE_FALSE`(값 `4`) 중 하나.

`MASK_CHANGE_TRUE` 이면 마스크값이 `true` 인 자리가, `MASK_CHANGE_FALSE` 이면 `false` 인 자리가 대상이 됨

`minNumber` - 범위의 하한.

이 값을 **포함**함 (`minNumber <= 복셀값`).

`cube` 의 타입과 달라도 됨.

`null` 이면 하한 제한이 없음

`maxNumber` - 범위의 상한.

이 값을 **포함하지 않음** (`복셀값 < maxNumber`).

`cube` 의 타입과 달라도 됨.

`null` 이면 상한 제한이 없음

`changeNumber` - 조건을 만족한 자리에 넣을 값.

`cube` 의 타입에 해당하는 객체여야 함.

`null` 이면 `cube` 의 `sentinel` 값으로 바꿈

Returns:

조건을 만족한 자리만 바꾼 새 큐브

Throws:

`NullPointerException` - `cube` 또는 `mask` 가 `null` 일 때

`IllegalArgumentException` - 다음 중 하나일 때

- 두 큐브의 level, `(col, row, floor)`, resolution 이 다름
- `cube` 가 bit, byte, short, int, float, double 이 아님
- `mask` 가 bit 가 아님
- `changeNumber` 가 `cube` 의 타입에 해당하는 객체가 아님
- `changeNumber` 가 `null` 인데 `cube` 의 `dataBlockCode` 가 `'V'`
- `minNumber` 나 `maxNumber` 가 `Boolean`, `Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나가 아니거나 NaN
- 둘 다 준 경우 `minNumber` 가 `maxNumber` 보다 작지 않음

- `option` 이 `MASK_CHANGE_TRUE`, `MASK_CHANGE_FALSE` 중 하나가 아님 (`MASK_PASS_TRUE` 처럼 쓰임이 다른 상수를 넘긴 경우 포함)

  `ArithmeticException` - `dataBlockCode` 가 `'V'`인 큐브의 대상 복셀값이 NaN 이어서 범위를 판정할 수 없을 때

12.2. and API

```
public static BasicDataCube and(BasicDataCube cube1,
    Object minNumber1,
    Object maxNumber1,
    int option1,
    BasicDataCube cube2,
    Object minNumber2,
    Object maxNumber2,
    int option2)
```

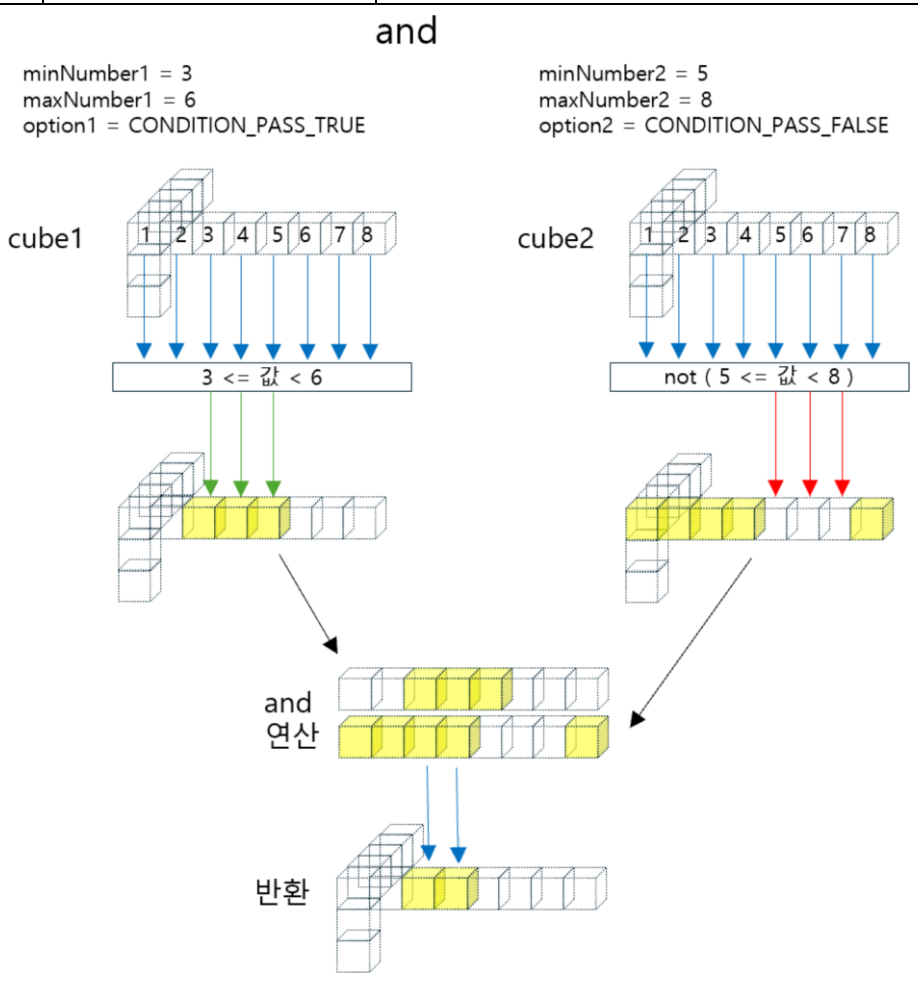
두 큐브의 범위 조건을 각각 계산한 뒤 AND 로 묶음.

두 조건이 모두 참인 자리만 결과가 `true` 가 됨.

and

메서드	파라미터 / 반환	설명
and	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 두 레스터 큐브에서, 첫번째로 각 큐브의 참/거짓 조건을 계산하고, 두번째로 계산된 결과를 이용하여 AND 논리연산을 적용.
	BasicDataCube cube1	레스터 데이터를 가지는 단일 큐브 1. 지원타입: bit,byte,short,int,float,double
	Object minNumber1	최소값 1 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber1	최대값 1 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	int option1	CONDITION_PASS_TRUE : 최소값 1~최대값 1 을 만족하는 복셀을 TRUE 로 인식 CONDITION_PASS_FALSE : 최소값 1~최대값 1 을 만족하는 복셀을 FALSE 로 인식
	BasicDataCube cube2	레스터 데이터를 가지는 단일 큐브 2. 지원타입: bit,byte,short,int,float,double

	Object minNumber2	최소값 2 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber2	최대값 2 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	int option2)	CONDITION_PASS_TRUE : 최소값 2~최대값 2 을 만족하는 복셀을 TRUE 로 인식 CONDITION_PASS_FALSE : 최소값 2~최대값 2 을 만족하는 복셀을 FALSE 로 인식
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: bit
		V2/V3/V4 타입에 대한 지원은 미정.




```
// terrain: float 큐브, density: int 큐브
BasicDataCube result = VoxelOpLogical.and(
        terrain, 0.0f, 100.0f, VoxelOpLogical.CONDITION_PASS_TRUE,
        density, 10, null, VoxelOpLogical.CONDITION_PASS_TRUE);
```

```
```

예제

시험 시나리오 TS-109-LOG(두 범위 AND)의 입력 파일이 있는 폴더에서 실행.

```
TS-109-LOG>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-109-LOG>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpLogical;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube cube1 = new BasicDataCube("cube1.bdc");
         BasicDataCube cube2 = new BasicDataCube("cube2.bdc");
         BasicDataCube result = VoxelOpLogical.and(cube1, 40, 70,
VoxelOpLogical.CONDITION_PASS_TRUE, cube2, 30, 50, VoxelOpLogical.CONDITION_PASS_TRUE);
         System.out.println(result);
         result.writeToFile("result.bdc");
         System.out.println("saved: result.bdc");
     }


 }
```

Parameters:

`cube1` - 레스터 데이터를 가지는 단일 큐브.

bit, byte, short, int, float, double 중 하나여야 함

`minNumber1` - `cube1` 범위의 하한.

이 값을 **포함**함.

`null` 이면 하한 제한이 없음

`maxNumber1` - `cube1` 범위의 상한.

이 값을 **포함하지 않음**.

`null` 이면 상한 제한이 없음

`option1` - `CONDITION_PASS_TRUE`(값 `5`) 또는 `CONDITION_PASS_FALSE`(값 `6`) 중 하나.

`CONDITION_PASS_TRUE` 이면 범위에 든 자리가 참, `CONDITION_PASS_FALSE` 이면 범위에 **들지 않은** 자리가 참임

`cube2` - `cube1` 과 level, `(col, row, floor)`, resolution 이 같아야 함.

datatype 은 달라도 됨

`minNumber2` - `cube2` 범위의 하한.

`minNumber1` 과 같은 규칙임

`maxNumber2` - `cube2` 범위의 상한.

`maxNumber1` 과 같은 규칙임

`option2` - `option1` 과 같은 규칙이며, `cube2` 에 적용됨

Returns:

두 조건을 AND 로 묶은 새 bit 큐브

Throws:

`NullPointerException` - `cube1` 또는 `cube2` 가 `null` 일 때

`IllegalArgumentException` - 다음 중 하나일 때

- 두 큐브의 level, `(col, row, floor)`, resolution 이 다름
- `cube1` 이나 `cube2` 가 bit, byte, short, int, float, double 이 아님
- `minNumber1`, `maxNumber1`, `minNumber2`, `maxNumber2` 중 하나가 `Boolean`, `Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나가 아니거나 NaN
- 한 큐브의 하한과 상한을 둘 다 준 경우 하한이 상한보다 작지 않음
- `option1` 이나 `option2` 가 `CONDITION_PASS_TRUE`, `CONDITION_PASS_FALSE` 중 하나가 아님 (`MASK_PASS_TRUE` 처럼 쓰임이 다른 상수를 넘긴 경우 포함)

`ArithmeticException` - `dataBlockCode` 가 `'V'`인 큐브의 복셀값이 NaN 이어서 범위를 판정할 수 없을 때

See Also:

- `or(BasicDataCube, Object, Object, int, BasicDataCube, Object, Object, int)`
- `xor(BasicDataCube, Object, Object, int, BasicDataCube, Object, Object, int)`
- `not(BasicDataCube, Object, Object, int)`

12.3. or API

```
public static BasicDataCube or(BasicDataCube cube1,
    Object minNumber1,
    Object maxNumber1,
    int option1,
    BasicDataCube cube2,
    Object minNumber2,
    Object maxNumber2,
    int option2)
```

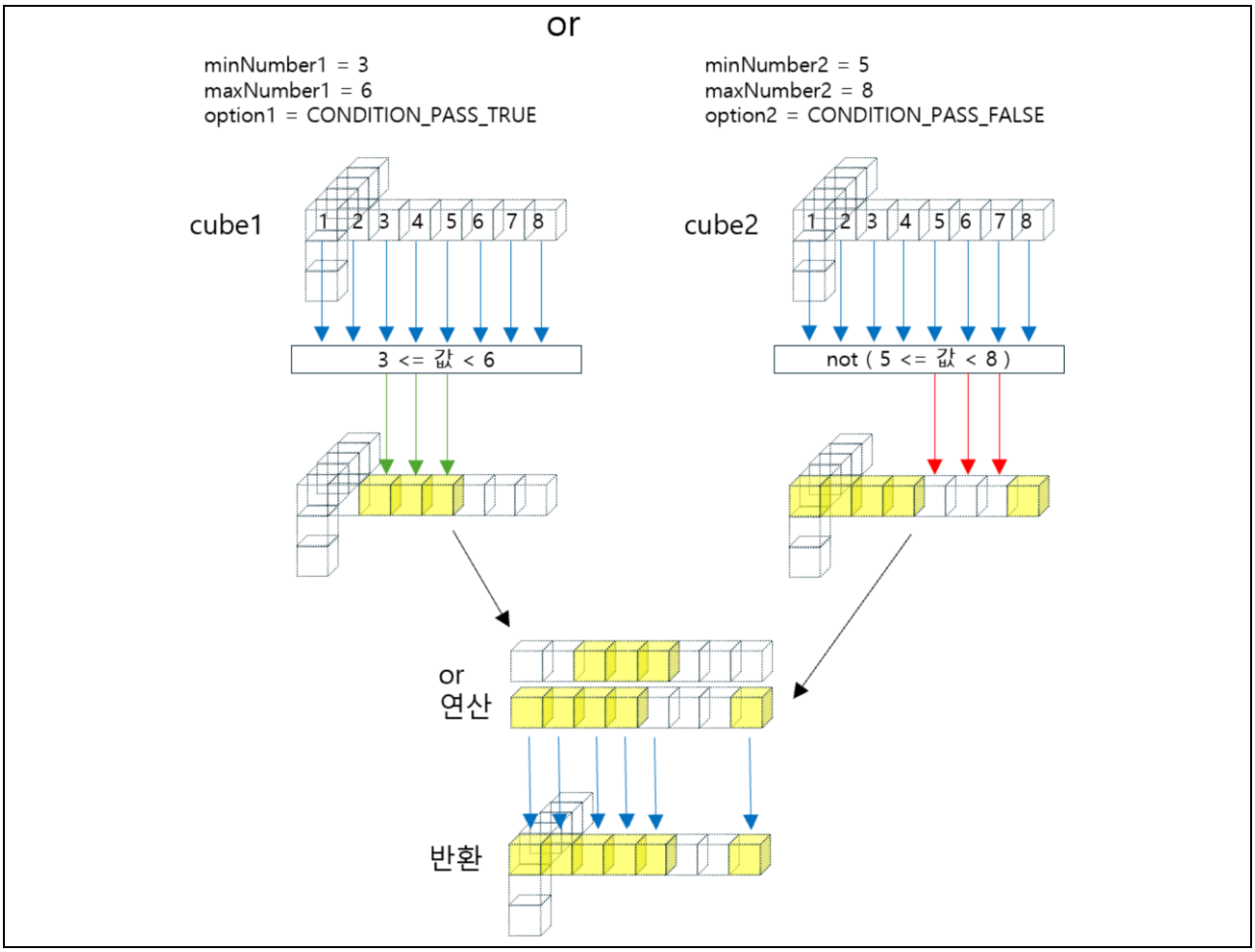
두 큐브의 범위 조건을 각각 계산한 뒤 OR 로 묶음.

두 조건 중 하나라도 참이면 결과가 `true` 가 됨.

or

메서드	파라미터 / 반환	설명
or	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 두 레스터 큐브에서, 첫번째로 각 큐브의 참/거짓 조건을 계산하고, 두번째로 계산된 결과를 이용하여 OR 논리연산을 적용.

	BasicDataCube cube1	레스터 데이터를 가지는 단일 큐브 1. 지원타입: bit,byte,short,int,float,double
	Object minNumber1	최소값 1 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber1	최대값 1 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	int option1	CONDITION_PASS_TRUE : 최소값 1~최대값 1 을 만족하는 복셀을 TRUE 로 인식 CONDITION_PASS_FALSE : 최소값 1~최대값 1 을 만족하는 복셀을 FALSE 로 인식
	BasicDataCube cube2	레스터 데이터를 가지는 단일 큐브 2. 지원타입: bit,byte,short,int,float,double
	Object minNumber2	최소값 2 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber2	최대값 2 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	int option2)	CONDITION_PASS_TRUE : 최소값 2~최대값 2 을 만족하는 복셀을 TRUE 로 인식 CONDITION_PASS_FALSE : 최소값 2~최대값 2 을 만족하는 복셀을 FALSE 로 인식
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: bit
		V2/V3/V4 타입에 대한 지원은 미정.




```
// 둘 중 하나라도 조건에 맞는 자리를 고름
BasicDataCube result = VoxelOpLogical.or(
        terrain, 0.0f, 100.0f, VoxelOpLogical.CONDITION_PASS_TRUE,
        density, 10, null, VoxelOpLogical.CONDITION_PASS_TRUE);
```

예제

시험 시나리오 TS-112-LOG(두 범위 OR)의 입력 파일이 있는 폴더에서 실행.

```
TS-112-LOG>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
```

```
TS-112-LOG>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpLogical;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube1 = new BasicDataCube("cube1.bdc");
        BasicDataCube cube2 = new BasicDataCube("cube2.bdc");
        BasicDataCube result = VoxelOpLogical.or(cube1, 40, 70, VoxelOpLogical.CONDITION_PASS_TRUE,
cube2, 30, 50, VoxelOpLogical.CONDITION_PASS_TRUE);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


}
```

Parameters:

`cube1` - 레스터 데이터를 가지는 단일 큐브.

bit, byte, short, int, float, double 중 하나여야 함

`minNumber1` - `cube1` 범위의 하한.

이 값을 **포함**함.

`null` 이면 하한 제한이 없음

`maxNumber1` - `cube1` 범위의 상한.

이 값을 **포함하지 않음**.

`null` 이면 상한 제한이 없음

`option1` - `CONDITION_PASS_TRUE`(값 `5`) 또는 `CONDITION_PASS_FALSE`(값 `6`) 중 하나.

`CONDITION_PASS_TRUE` 이면 범위에 든 자리가 참, `CONDITION_PASS_FALSE` 이면 범위에 **들지 않은** 자리가 참임

`cube2` - `cube1` 과 level, `(col, row, floor)`, resolution 이 같아야 함.

datatype 은 달라도 됨

`minNumber2` - `cube2` 범위의 하한.

`minNumber1` 과 같은 규칙임

`maxNumber2` - `cube2` 범위의 상한.

`maxNumber1` 과 같은 규칙임

`option2` - `option1` 과 같은 규칙이며, `cube2` 에 적용됨

Returns:

두 조건을 OR 로 묶은 새 bit 큐브

Throws:

`NullPointerException` - `cube1` 또는 `cube2` 가 `null` 일 때

`IllegalArgumentException` - 다음 중 하나일 때

- 두 큐브의 level, `(col, row, floor)`, resolution 이 다름
- `cube1` 이나 `cube2` 가 bit, byte, short, int, float, double 이 아님
- `minNumber1`, `maxNumber1`, `minNumber2`, `maxNumber2` 중 하나가 `Boolean`, `Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나가 아니거나 NaN
- 한 큐브의 하한과 상한을 둘 다 준 경우 하한이 상한보다 작지 않음
- `option1` 이나 `option2` 가 `CONDITION_PASS_TRUE`, `CONDITION_PASS_FALSE` 중 하나가 아님 (`MASK_PASS_TRUE` 처럼 쓰임이 다른 상수를 넘긴 경우 포함)

`ArithmeticException` - `dataBlockCode` 가 `'V'`인 큐브의 복셀값이 NaN 이어서 범위를 판정할 수 없을 때

See Also:

- `and(BasicDataCube, Object, Object, int, BasicDataCube, Object, Object, int)`
- `xor(BasicDataCube, Object, Object, int, BasicDataCube, Object, Object, int)`
- `not(BasicDataCube, Object, Object, int)`

12.4. xor API

```
public static BasicDataCube xor(BasicDataCube cube1,
    Object minNumber1,
    Object maxNumber1,
    int option1,
    BasicDataCube cube2,
    Object minNumber2,
    Object maxNumber2,
    int option2)
```

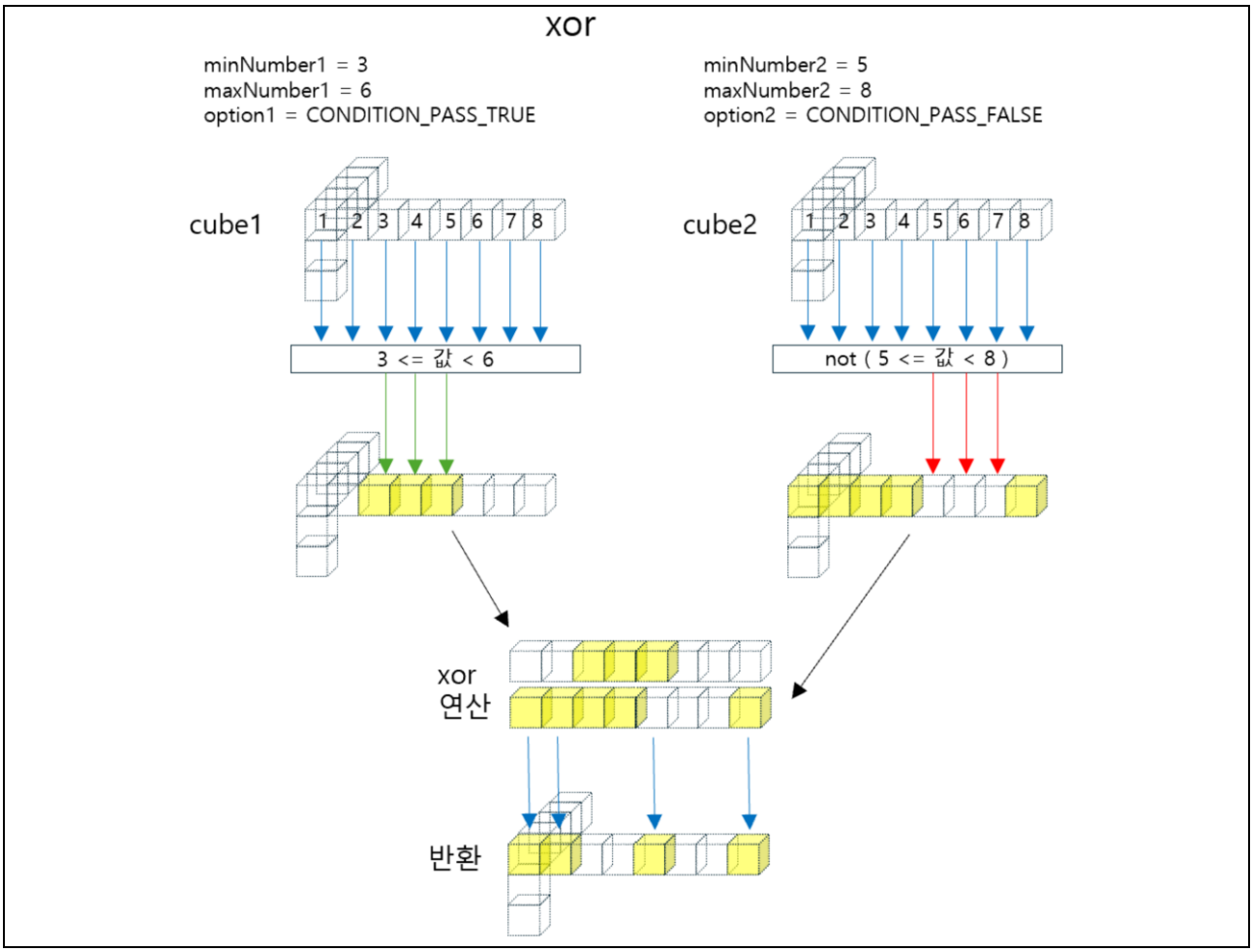
두 큐브의 범위 조건을 각각 계산한 뒤 XOR 로 묶음.

두 조건이 서로 다를 때만 결과가 `true` 가 됨.

xor

메서드	파라미터 / 반환	설명
xor	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 두 레스터 큐브에서, 첫번째로 각 큐브의 참/거짓 조건을 계산하고, 두번째로 계산된 결과를 이용하여 XOR 논리연산을 적용.
	BasicDataCube cube1	레스터 데이터를 가지는 단일 큐브 1. 지원타입: bit,byte,short,int,float,double
	Object minNumber1	최소값 1 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber1	최대값 1 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	int option1	CONDITION_PASS_TRUE : 최소값 1~최대값 1 을 만족하는 복셀을 TRUE 로 인식 CONDITION_PASS_FALSE : 최소값 1~최대값 1 을 만족하는 복셀을 FALSE 로 인식
	BasicDataCube cube2	레스터 데이터를 가지는 단일 큐브 2.

		지원타입: bit,byte,short,int,float,double
	Object minNumber2	최소값 2 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 최소값 2 은 cube2 가 가지는 데이터 타입에 따라 아래와 같은 객체를 사용 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber2	최대값 2 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 최대값 2 은 cube2 가 가지는 데이터 타입에 따라 아래와 같은 객체를 사용 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	int option2)	CONDITION_PASS_TRUE : 최소값 2~최대값 2 을 만족하는 복셀을 TRUE 로 인식 CONDITION_PASS_FALSE : 최소값 2~최대값 2 을 만족하는 복셀을 FALSE 로 인식
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: bit
		V2/V3/V4 타입에 대한 지원은 미정.




```
// 한쪽 조건에만 맞는 자리를 고름. 두 조건이 겹치는 곳은 빠짐
BasicDataCube result = VoxelOpLogical.xor(
        terrain, 0.0f, 100.0f, VoxelOpLogical.CONDITION_PASS_TRUE,
        density, 10, null, VoxelOpLogical.CONDITION_PASS_TRUE);
```

예제

시험 시나리오 TS-115-LOG(두 범위 XOR)의 입력 파일이 있는 폴더에서 실행.

```
TS-115-LOG>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
```

```
TS-115-LOG>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpLogical;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube1 = new BasicDataCube("cube1.bdc");
        BasicDataCube cube2 = new BasicDataCube("cube2.bdc");
        BasicDataCube result = VoxelOpLogical.xor(cube1, 40, 70,
VoxelOpLogical.CONDITION_PASS_TRUE, cube2, 30, 50, VoxelOpLogical.CONDITION_PASS_TRUE);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


}
```

Parameters:

`cube1` - 레스터 데이터를 가지는 단일 큐브.

bit, byte, short, int, float, double 중 하나여야 함

`minNumber1` - `cube1` 범위의 하한.

이 값을 **포함**함.

`null` 이면 하한 제한이 없음

`maxNumber1` - `cube1` 범위의 상한.

이 값을 **포함하지 않음**.

`null` 이면 상한 제한이 없음

`option1` - `CONDITION_PASS_TRUE`(값 `5`) 또는 `CONDITION_PASS_FALSE`(값 `6`) 중 하나.

`CONDITION_PASS_TRUE` 이면 범위에 든 자리가 참, `CONDITION_PASS_FALSE` 이면 범위에 **들지 않은** 자리가 참임

`cube2` - `cube1` 과 level, `(col, row, floor)`, resolution 이 같아야 함.

datatype 은 달라도 됨

`minNumber2` - `cube2` 범위의 하한.

`minNumber1` 과 같은 규칙임

`maxNumber2` - `cube2` 범위의 상한.

`maxNumber1` 과 같은 규칙임

`option2` - `option1` 과 같은 규칙이며, `cube2` 에 적용됨

Returns:

두 조건을 XOR 로 묶은 새 bit 큐브

Throws:

`NullPointerException` - `cube1` 또는 `cube2` 가 `null` 일 때

`IllegalArgumentException` - 다음 중 하나일 때

- 두 큐브의 level, `(col, row, floor)`, resolution 이 다름
- `cube1` 이나 `cube2` 가 bit, byte, short, int, float, double 이 아님
- `minNumber1`, `maxNumber1`, `minNumber2`, `maxNumber2` 중 하나가 `Boolean`, `Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나가 아니거나 NaN
- 한 큐브의 하한과 상한을 둘 다 준 경우 하한이 상한보다 작지 않음
- `option1` 이나 `option2` 가 `CONDITION_PASS_TRUE`, `CONDITION_PASS_FALSE` 중 하나가 아님 (`MASK_PASS_TRUE` 처럼 쓰임이 다른 상수를 넘긴 경우 포함)

`ArithmeticException` - `dataBlockCode` 가 `'V'`인 큐브의 복셀값이 NaN 이어서 범위를 판정할 수 없을 때

See Also:

- `and(BasicDataCube, Object, Object, int, BasicDataCube, Object, Object, int)`
- `or(BasicDataCube, Object, Object, int, BasicDataCube, Object, Object, int)`
- `not(BasicDataCube, Object, Object, int)`

12.5. not API

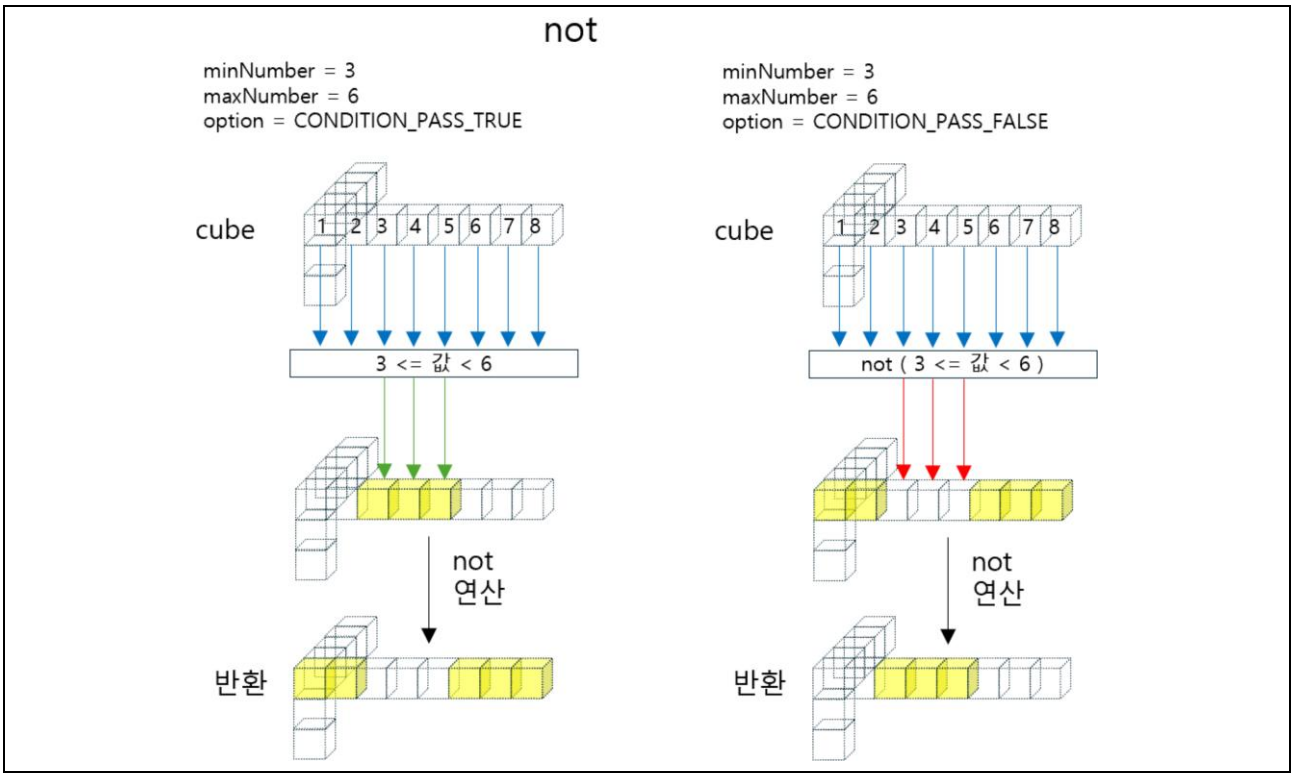
```
public static BasicDataCube not(BasicDataCube cube,
    Object minNumber,
    Object maxNumber,
    int option)
```

한 큐브의 범위 조건을 계산한 뒤 뒤집음.

`minNumber <= 복셀값 < maxNumber` 인지 보고 `option` 으로 그 판정을 참으로 읽을지 거짓으로 읽을지 정한 다음, **그 결과를 뒤집어서 결과 큐브에 넣음.** 즉 `CONDITION_PASS_TRUE` 를 주면 범위에 **들지 않은** 자리가 `true` 가 됨.

not

메서드	파라미터 / 반환	설명
not	(	입력으로 들어온 레스터 큐브에서, 첫번째로 큐브의 참/거짓 조건을 계산하고, 두번째로 계산된 결과를 이용하여 NOT 논리연산을 적용.
	BasicDataCube cube	레스터 데이터를 가지는 단일 큐브. 지원타입: bit,byte,short,int,float,double
	Object minNumber	최소값 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber	최대값 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	int option)	CONDITION_PASS_TRUE : 최소값~최대값을 만족하는 복셀을 TRUE 로 인식 CONDITION_PASS_FALSE : 최소값~최대값을 만족하는 복셀을 FALSE 로 인식
	BasicDataCube	연산후 결과를 단일 큐브로 반환. 반환타입: bit
		V2/V3/V4 타입에 대한 지원은 미정.




```
// terrain: float 큐브. 0 이상 100 미만이 아닌 자리를 고름
BasicDataCube result = VoxelOpLogical.not(
        terrain, 0.0f, 100.0f, VoxelOpLogical.CONDITION_PASS_TRUE);
```

예제

시험 시나리오 TS-118-LOG(범위 반전)의 입력 파일이 있는 폴더에서 실행.

```
TS-118-LOG>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
```

```
TS-118-LOG>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpLogical;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube cube = new BasicDataCube("cube.bdc");
        BasicDataCube result = VoxelOpLogical.not(cube, 40, 70,
VoxelOpLogical.CONDITION_PASS_TRUE);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


}
```

Parameters:

cube - 레스터 데이터를 가지는 단일 큐브.

bit, byte, short, int, float, double 중 하나여야 함

`minNumber` - 범위의 하한.

이 값을 **포함**함.

`null` 이면 하한 제한이 없음

`maxNumber` - 범위의 상한.

이 값을 **포함하지 않음**.

`null` 이면 상한 제한이 없음

`option` - `CONDITION_PASS_TRUE`(값 `5`) 또는 `CONDITION_PASS_FALSE`(값 `6`) 중 하나.

뒤집기 전의 판정을 정함 — `CONDITION_PASS_TRUE` 이면 범위에 든 자리가 참이고, 뒤집은 뒤에는 범위에 들지 않은 자리가 `true` 가 됨

Returns:

조건을 뒤집은 새 bit 큐브

Throws:

`NullPointerException` - `cube` 가 `null` 일 때

`IllegalArgumentException` - 다음 중 하나일 때

- `cube` 가 bit, byte, short, int, float, double 이 아님
- `minNumber` 나 `maxNumber` 가 `Boolean`, `Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나가 아니거나 NaN
- 둘 다 준 경우 `minNumber` 가 `maxNumber` 보다 작지 않음
- `option` 이 `CONDITION_PASS_TRUE`, `CONDITION_PASS_FALSE` 중 하나가 아님 (`MASK_PASS_TRUE` 처럼 쓰임이 다른 상수를 넘긴 경우 포함)

`ArithmeticException` - `dataBlockCode` 가 `'V'`인 큐브의 복셀값이 NaN 이어서 범위를 판정할 수 없을 때

See Also:

- `and(BasicDataCube, Object, Object, int, BasicDataCube, Object, Object, int)`
- `or(BasicDataCube, Object, Object, int, BasicDataCube, Object, Object, int)`
- `xor(BasicDataCube, Object, Object, int, BasicDataCube, Object, Object, int)`

13. VoxelOpFocal [레스터]

클래스 설명

```
public final class VoxelOpFocal
    extends Object
```

[레스터] 초점 연산자.

복셀마다 자기 자신과 주변 이웃의 값을 모아 하나의 통계값으로 바꿈.

인스턴스를 만들 수 없으며 연산마다 정적 메서드를 제공함.

모든 연산은 **큐브 리스트**를 받아 큐브마다 같은 연산을 적용하며, **맞닿은 큐브의 복셀도 이웃으로 봄**.

큐브 하나만 계산하려면 원소가 하나인 리스트를 넘기면 됨.

```
VoxelOpFocal.mean(cubes, Neighbor.N26, 1);                        // 큐브마다 복셀의 3x3x3 창 평균
VoxelOpFocal.sum(new BasicDataCube[] {cube}, Neighbor.N6, 2)[0]; // 큐브 하나
```

입력 큐브는 변경하지 않고 언제나 새 큐브를 만들어 돌려줌.

모두 상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

연산 목록

부를 수 있는 메서드

메서드	계산	반환
`mean(cubes, neighbor, distance)`	큐브마다 창 안 값의 평균	모두 double
`sum(cubes, neighbor, distance)`	큐브마다 창 안 값의 합	큐브마다 그 큐브의 타입에서 시작해 넘치면 승격
`min(cubes, neighbor, distance)`	큐브마다 창 안 값의 최소	큐브마다 그 큐브의 타입 (센티넬 자리와 겹칠 때만 승격)
`max(cubes, neighbor, distance)`	큐브마다 창 안 값의 최대	큐브마다 그 큐브의 타입 (센티넬 자리와 겹칠 때만 승격)
`standardDeviation(cubes, neighbor, distance)`	큐브마다 창 안 값의 표준편차 (모집단, `n` 으로 나눔)	모두 double
`variance(cubes, neighbor, distance)`	큐브마다 창 안 값의 분산 (모집단, `n` 으로 나눔)	모두 double
`majority(cubes, neighbor, distance)`	큐브마다 창 안에서 가장 많이 나온 값. 여럿이면 그 값들의 평균	큐브마다 그 큐브의 타입. 정수가 아닌 결과가 하나라도 있으면 byte/short 는 float, int 는 double
`median(cubes, neighbor, distance)`	큐브마다 창 안 값을 정렬한 가운데 값. 짝수 개면 가운데 두 값의 평균	큐브마다 그 큐브의 타입. 정수가 아닌 결과가 하나라도 있으면

	(두 값이 같으면 그 값)	byte/short 는 float, int 는 double
`count(cubes, neighbor, distance, minNumber, maxNumber)`	큐브마다 창 안에서 `minNumber <= 값 < maxNumber` 인 값의 개수. `null` 인 쪽은 제한 없음	모두 int, 언제나 `'V'`

모든 연산은 입력과 길이·순서가 같은 큐브 배열을 돌려줌.

창(window)

초점 복셀 `(x, y, z)`에 대해 `NeighborOffsetTable` 이 주는 `neighbor·distance` 조합의 오프셋을 더한 자리가 이웃이고, **초점 복셀 자신도 창에 포함됨**.

창의 모양 (d = distance)

neighbor	포함 조건	d = 1 일 때 창 크기 (자신 포함)
`Neighbor.N6`	`|dx| + |dy| + |dz| <= d` (맨해튼)	7
`Neighbor.N18`	`|dx| + |dy| + |dz| <= 2d`, 각 축 `<= d`	19
`Neighbor.N26`	각 축 `<= d` (체비셰프) — `(2d+1)^3` 정육면체	27

큐브 경계

창이 큐브 밖으로 나가면 `MultiCubeCoordMgr` 로 **그 자리의 큐브를 찾아 이어서 봄**.
리스트에 큐브가 하나뿐이면 이어 볼 큐브가 없으므로 큐브 안의 이웃만 씀.

큐브들을 **level, resolution, datatype 이 같은 것끼리 묶어** 묶음마다 좌표계 하나를 만듦.
종류가 다른 큐브는 격자가 맞지 않아 이웃이 될 수 없으므로 서로 들여다보지 않음.
그래서 리스트 안에 종류가 섞여 있어도 되고, 위치가 떨어져 있어도 됨.
같은 묶음 안에서 `(col, row, floor)`가 겹치면 어느 쪽을 봐야 할지 정할 수 없어 `IllegalArgumentException` 을 던짐.

창에서 빠지는 자리

다음 자리는 창에 넣지 않고 **나머지 값만으로** 계산함.
초점 복셀도 창의 구성원 하나일 뿐이라 같은 규칙을 따름.

- 좌표계 밖 — 그 자리를 담은 큐브가 리스트에 없음. 리스트 가장자리에 붙은 복셀은 이웃이 적음
- `'S'` 큐브의 센티넬 자리 (float/double 은 NaN 도 센티넬로 봄). 이웃 큐브의 복셀은 **그 큐브의** 센티넬 규칙으로 판정함

그래서 초점이 센티넬이어도 이웃에 값이 있으면 그 값들로 결과를 정하고, 초점만 값이 있고 이웃이 모두 센티넬이면 초점 값 하나로 결과를 정함.
초점과 이웃이 모두 센티넬일 때만 그 자리의 결과가 센티넬임 — 통계값을 낼 것이 없으므로.

개수를 세는 `count` 만은 셀 것이 없는 것도 답이라 센티넬이 아니라 0 임.

`'V'` 큐브는 센티넬이 없으므로 창이 비는 일이 없음.

던지는 예외

예외와 원인

예외	언제
`NullPointerException`	큐브, 큐브 리스트, 리스트의 원소, `neighbor` 가 `null` 일 때
`IllegalArgumentException`	큐브 타입이 bit 나 벡터일 때, `distance` 가 1 보다 작을 때, 리스트에서 level·resolution·datatype 이 같은 큐브끼리 `(col, row, floor)`가 겹칠 때, `count` 의 하한·상한이 Byte/Short/Integer/Float/Double 이 아니거나 NaN 일 때, 둘 다 주었는데 `minNumber >= maxNumber` 일 때
`ArithmeticException`	`'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 계산 결과가 double 을 넘을 때 (double 큐브의 합이나 분산이 넘치는 경우)

지원 타입

`BasicDataCube.DataType.BYTE`, `BasicDataCube.DataType.SHORT`, `BasicDataCube.DataType.INT`, `BasicDataCube.DataType.FLOAT`, `BasicDataCube.DataType.DOUBLE`.

결과 큐브의 헤더

입력에서 물려받는 것과 다시 정하는 것

헤더 필드	결과값
`level`, `col`, `row`, `floor`, `resolution`	입력 큐브의 것을 그대로 물려받음
`datatype`	연산마다 정해짐 — 언제나 double 인 연산, 언제나 int 인 연산(`count`), 입력 타입에서 시작해 넘치면 승격하는 연산, 평균으로 소수가 나오면 실수 타입이 되는 연산(`majority`, `median`)이 있음. 위 <연산 목록>의 반환 열과 아래 <반환 타입과 승격> 참고
`dataBlockCode`	입력과 같음 — `'V'`이면 `'V'`, `'S'`이면 `'S'`. `count` 만 입력과 무관하게 `'V'`
`sentinel`	`'S'`이면 byte/short/int 는 반환 타입의 최대값, float/double 은 NaN. `'V'`이면 없음(`null`)

반환 타입과 승격

입력 타입을 따르는 연산(`sum`, `min`, `max`)은 큐브의 타입에서 시작해, 계산 결과가 그 타입에 담기지 않으면 아래 순서로 승격함.

```
byte -> short -> int -> double
float -> double
```

반환 타입은 **큐브 단위로 하나**임.

복셀 하나만 넘쳐도 큐브 전체가 승격됨.

double 에서도 넘치면 `ArithmeticException` 을 던짐.

결과가 `'S'`이고 정수 타입이면 최대값은 센티넬 자리로 예약되어 값으로 쓸 수 없음.

계산 결과가 그 값이면 넘친 것으로 보고 승격함.

예를 들어 byte 창의 합이 127 이면 `Byte.MAX_VALUE` 와 같으므로 결과는 short 가 됨.

`'V'`이면 예약하지 않음.

입력값 중 하나를 고르는 연산(`min`, `max`)은 계산으로 넘치는 일이 없으므로, 이 예약에 걸리는 경우 — `'S'` 정수 큐브의 결과로 그 타입의 최대값이 뽑힌 경우 — 에만 승격함.

float, double 은 센티넬이 NaN 이라 최대값을 예약하지 않음.

`majority` 와 `median` 은 보통 입력값 중 하나를 고르므로 `min`, `max` 와 같이 다루지만, 평균으로 처리하는 자리(majority 는 다수결값이 여럿일 때, median 은 짝수 개일 때)에서 **정수가 아닌 결과가 그 큐브에 하나라도 나오면** 시작 타입이 바뀜.

byte, short 는 float, int 는 double. float, double 은 그대로임.

평균이 쓰였어도 결과가 모두 정수이면 — `{2, 4}`의 평균 3 처럼 — 입력 타입을 유지함.

```
결과가 모두 정수:   byte -> byte,  short -> short,  int -> int
소수 결과가 있음:   byte -> float, short -> float, int -> double
```

그러므로 결과 큐브의 값을 읽기 전에 `datatype` 을 확인해야 함.

센티넬과 NaN

복셀 하나하나를 볼 때:

- `'V'`이고 float/double 복셀값이 NaN 이면 `ArithmeticException`
- 복셀값이 무한대이면 `ArithmeticException` — 어느 통계도 유한한 값으로 만들지 못함

Method Summary

Modifier and Type	Method	Description
`static BasicDataCube[]`	`count(BasicDataCube[] cubes, Neighbor neighbor, int distance, Object minNumber, Object maxNumber)`	큐브마다 복셀의 창 안에서 `minNumber <= 값 < maxNumber` 인 값이 몇 개인지 셈.
`static BasicDataCube[]`	`majority(BasicDataCube[] cubes,`	큐브마다 복셀의 창 안에서 가장 많이 나온

	`Neighbor neighbor, int distance)`	값(다수결값)을 구함.
`static BasicDataCube[]`	`max(BasicDataCube[] cubes, Neighbor neighbor, int distance)`	큐브마다 복셀의 창 안 값의 최대값을 구함.
`static BasicDataCube[]`	`mean(BasicDataCube[] cubes, Neighbor neighbor, int distance)`	큐브마다 복셀의 창 안 값의 평균을 구함.
`static BasicDataCube[]`	`median(BasicDataCube[] cubes, Neighbor neighbor, int distance)`	큐브마다 복셀의 창 안 값을 크기순으로 정렬해 가운데 값(중앙값)을 구함.
`static BasicDataCube[]`	`min(BasicDataCube[] cubes, Neighbor neighbor, int distance)`	큐브마다 복셀의 창 안 값의 최소값을 구함.
`static BasicDataCube[]`	`standardDeviation(BasicDataCube[] cubes, Neighbor neighbor, int distance)`	큐브마다 복셀의 창 안 값의 표준편차를 구함.
`static BasicDataCube[]`	`sum(BasicDataCube[] cubes, Neighbor neighbor, int distance)`	큐브마다 복셀의 창 안 값의 합을 구함.
`static BasicDataCube[]`	`variance(BasicDataCube[] cubes, Neighbor neighbor, int distance)`	큐브마다 복셀의 창 안 값의 분산을 구함.

13.1. mean API

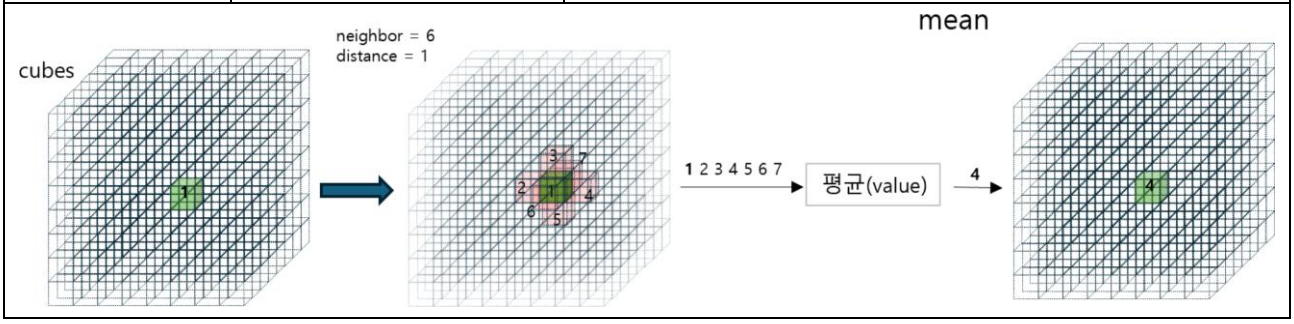
```
public static BasicDataCube[] mean(BasicDataCube[] cubes,
    Neighbor neighbor,
    int distance)
```

큐브마다 복셀의 창 안 값의 평균을 구함.

mean

메서드	파라미터 / 반환	설명
mean	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 임의 복셀을 초점으로하여 거리만큼 이웃선택(자기자신복셀포함)하여 선택된 이웃들의 평균값을 초점복셀의 값으로 취하는 연산. (큐브들간 공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음) (만약 공간크기(level)가 같은 큐브들이 인접해 있을경우 경계면 근처에서 이웃선택은 경계면을 넘어서는 인접큐브에서도 찾음) (찾은 이웃들(자신포함)중 찾은 복셀값이 sentinel 일때: 이값은 평균계산에서 제외시키고 연산)
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	Neighbor neighbor	이웃선택 방법: N6, N18, N26 중 하나. (초점연산에서 이웃선택에 박스(정육면체)형식도 포함할것: N26 으로 찾으면 정육면체 박스임)
	int distance)	이웃선택 거리: 1,2,3,... 중 하나. (복셀단위)

	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 큐브별로 구해진 결과를 cubes 의 순서와 일치시켜 반환 반환타입: double
		bit/V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-121-FOC(W4 창 평균)의 입력 파일이 있는 폴더에서 실행.

```
TS-121-FOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-121-FOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.Neighbor;
import com.voxel.operator.VoxelOpFocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpFocal.mean(cubes, Neighbor.N6, 7);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
```

```
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }

}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

모두 byte, short, int, float, double 중 하나여야 함

`neighbor` - 이웃선택 방법.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

`distance` - 이웃선택 거리 (복셀 단위).

1 이상

Returns:

큐브마다 평균을 담은 새 double 큐브.

`cubes` 와 길이·순서가 같음.

빈 리스트면 빈 배열

Throws:

`NullPointerException` - `cubes`, 그 원소, `neighbor` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 타입이 섞였을 때, `distance` 가 1 보다 작을 때,

level·resolution·타입이 같은 큐브끼리 `(col, row, floor)`가 겹칠 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때

13.2. sum API

```
public static BasicDataCube[] sum(BasicDataCube[] cubes,
    Neighbor neighbor,
    int distance)
```

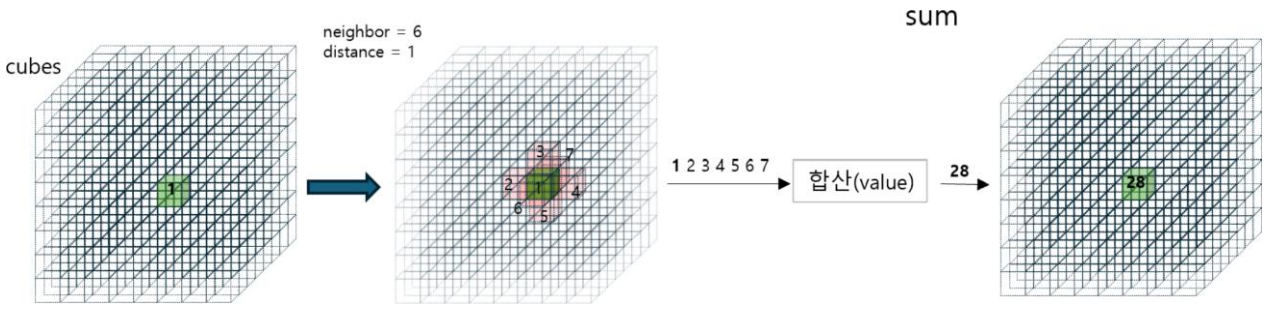
큐브마다 복셀의 창 안 값의 합을 구함.

합은 double 로 구하고 정해진 반환 타입으로 저장함.

sum

메서드	파라미터 / 반환	설명
sum	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 임의 복셀을 초점으로하여 거리만큼 이웃선택(자기자신복셀포함)하여 선택된 이웃들의 합산값을 초점복셀의 값으로 취하는 연산.

		(큐브들간 공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	Neighbor neighbor	이웃선택 방법: N6, N18, N26 중 하나.
	int distance)	이웃선택 거리: 1,2,3,... 중 하나. (복셀단위)
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 큐브별로 구해진 결과를 cubes 의 순서와 일치시켜 반환 반환타입: cubes 의 각 데이터 타입에 해당하는 타입으로 반환.
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환. (byte->short, short->int, int->double, float->double, double->예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-124-FOC(W4 창 합)의 입력 파일이 있는 폴더에서 실행.

```
TS-124-FOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-124-FOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.Neighbor;
import com.voxel.operator.VoxelOpFocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpFocal.sum(cubes, Neighbor.N26, 1);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }
```

```
    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }

}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

모두 byte, short, int, float, double 중 하나여야 함

`neighbor` - 이웃선택 방법.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

`distance` - 이웃선택 거리 (복셀 단위).

1 이상

Returns:

큐브마다 합을 담은 새 큐브.

`cubes` 와 길이·순서가 같음.

빈 리스트면 빈 배열

Throws:

`NullPointerException` - `cubes`, 그 원소, `neighbor` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 타입이 섞였을 때, `distance` 가 1 보다 작을 때, level·resolution·타입이 같은 큐브끼리 `(col, row, floor)`가 겹칠 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 창의 합이 double 을 넘을 때

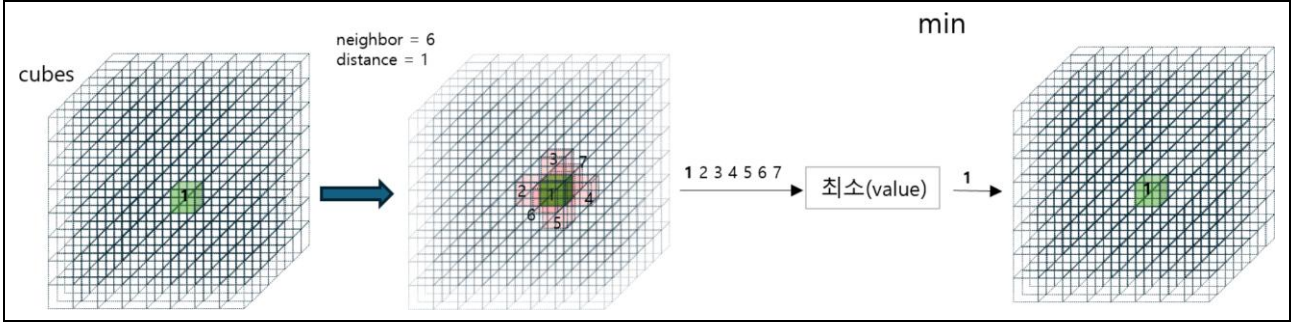
13.3. min API

```
public static BasicDataCube[] min(BasicDataCube[] cubes,
    Neighbor neighbor,
    int distance)
```

큐브마다 복셀의 창 안 값의 최소값을 구함.

min

메서드	파라미터 / 반환	설명
min	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 임의 복셀을 초점으로하여 거리만큼 이웃선택(자기자신복셀포함)하여 선택된 이웃들의 최소값을 초점복셀의 값으로 취하는 연산. (큐브들간 공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	Neighbor neighbor	이웃선택 방법: N6, N18, N26 중 하나.
	int distance)	이웃선택 거리: 1,2,3,... 중 하나. (복셀단위)
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 큐브별로 구해진 결과를 cubes 의 순서와 일치시켜 반환 반환타입: cubes 의 각 데이터 타입에 해당하는 타입으로 반환.
		bit/V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-127-FOC(W4 창 최소)의 입력 파일이 있는 폴더에서 실행.

```
TS-127-FOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
```

```
TS-127-FOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.Neighbor;
import com.voxel.operator.VoxelOpFocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpFocal.min(cubes, Neighbor.N6, 3);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

모두 byte, short, int, float, double 중 하나여야 함

`neighbor` - 이웃선택 방법.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

`distance` - 이웃선택 거리 (복셀 단위).

1 이상

Returns:

큐브마다 최소값을 담은 새 큐브.

`cubes` 와 길이·순서가 같음.

빈 리스트면 빈 배열

Throws:

`NullPointerException` - `cubes`, 그 원소, `neighbor` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 타입이 섞였을 때, `distance` 가 1 보다 작을 때, level·resolution·타입이 같은 큐브끼리 `(col, row, floor)`가 겹칠 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때

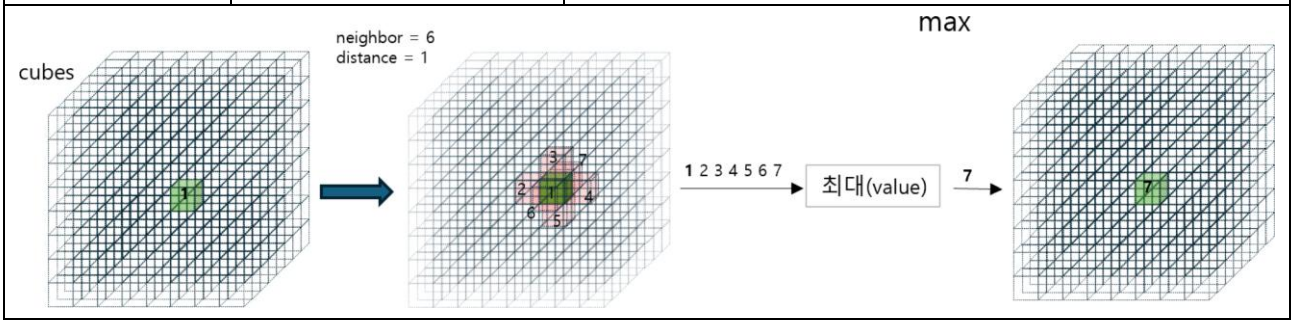
13.4. max API

```
public static BasicDataCube[] max(BasicDataCube[] cubes,
    Neighbor neighbor,
    int distance)
```

큐브마다 복셀의 창 안 값의 최대값을 구함.

max

메서드	파라미터 / 반환	설명
max	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 임의 복셀을 초점으로하여 거리만큼 이웃선택(자기자신복셀포함)하여 선택된 이웃들의 최대값을 초점복셀의 값으로 취하는 연산. (큐브들간 공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	Neighbor neighbor	이웃선택 방법: N6, N18, N26 중 하나.
	int distance)	이웃선택 거리: 1,2,3,... 중 하나. (복셀단위)
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 큐브별로 구해진 결과를 cubes 의 순서와 일치시켜 반환 반환타입: cubes 의 각 데이터 타입에 해당하는 타입으로 반환.
		bit/V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-130-FOC(W4 창 최대)의 입력 파일이 있는 폴더에서 실행.

```
TS-130-FOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-130-FOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.Neighbor;
import com.voxel.operator.VoxelOpFocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpFocal.max(cubes, Neighbor.N6, 3);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

모두 byte, short, int, float, double 중 하나여야 함

`neighbor` - 이웃선택 방법.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

`distance` - 이웃선택 거리 (복셀 단위).

1 이상

Returns:

큐브마다 최대값을 담은 새 큐브.

`cubes` 와 길이·순서가 같음.

빈 리스트면 빈 배열

Throws:

`NullPointerException` - `cubes`, 그 원소, `neighbor` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 타입이 섞였을 때, `distance` 가 1 보다 작을 때, level·resolution·타입이 같은 큐브끼리 `(col, row, floor)`가 겹칠 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때

13.5. standardDeviation API

```
public static BasicDataCube[] standardDeviation(BasicDataCube[] cubes,
    Neighbor neighbor,
    int distance)
```

큐브마다 복셀의 창 안 값의 표준편차를 구함.

모집단 표준편차임 — 편차 제곱의 합을 창에서 찾은 값의 개수 `n` (센티넬을 뺀 이웃 + 자기 자신)으로 나눈 뒤 제곱근을 취함.

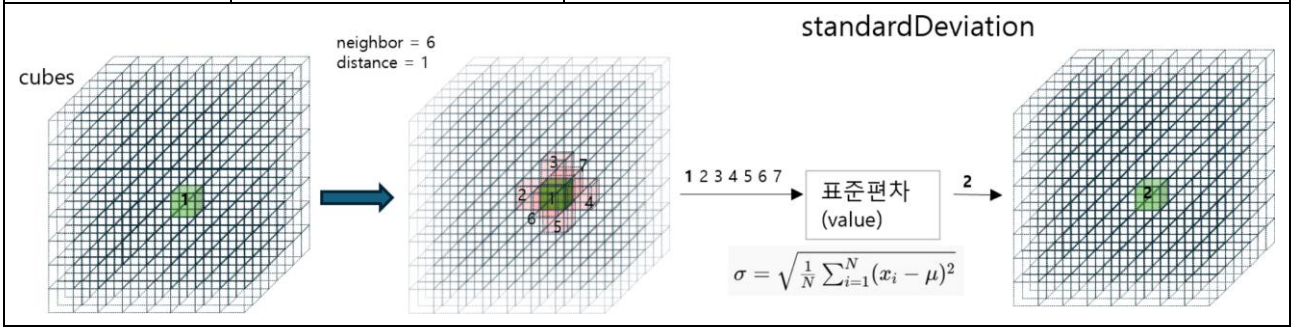
`n - 1` 로 나누는 표본 표준편차가 아님.

창에 값이 하나뿐이면 0.

standardDeviation

메서드	파라미터 / 반환	설명
standardDeviation	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 임의 복셀을 초점으로하여 거리만큼 이웃선택(자기자신복셀포함)하여 선택된 이웃들의 표준편차값을 초점복셀의 값으로 취하는 연산. (큐브들간 공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	Neighbor neighbor	이웃선택 방법: N6, N18, N26 중 하나.
	int distance)	이웃선택 거리: 1,2,3,... 중 하나. (복셀단위)

	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 큐브별로 구해진 결과를 cubes 의 순서와 일치시켜 반환 반환타입: double
		bit/V2/V3/V4 타입에 대한 지원은 미정.



값을 2 의 거듭제곱으로 축소해 계산한 뒤 복원하므로, 복셀값이 double 의 끝에 가까워도 중간 계산이 넘치지 않음.

표준편차는 값의 범위를 넘지 않으므로 결과는 언제나 유한함.

예제

시험 시나리오 TS-133-FOC(W4 창 표준편차)의 입력 파일이 있는 폴더에서 실행.

```
TS-133-FOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-133-FOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.Neighbor;
import com.voxel.operator.VoxelOpFocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpFocal.standardDeviation(cubes, Neighbor.N6, 1);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }
```

```
    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

모두 byte, short, int, float, double 중 하나여야 함

`neighbor` - 이웃선택 방법.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

`distance` - 이웃선택 거리 (복셀 단위).

1 이상

Returns:

큐브마다 표준편차를 담은 새 double 큐브.

`cubes` 와 길이·순서가 같음.

`'S'`이면 센티넬은 `Double.NaN`. 빈 리스트면 빈 배열

Throws:

`NullPointerException` - `cubes`, 그 원소, `neighbor` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 타입이 섞였을 때, `distance` 가 1 보다 작을 때,

level·resolution·타입이 같은 큐브끼리 `(col, row, floor)`가 겹칠 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때

13.6. variance API

```
public static BasicDataCube[] variance(BasicDataCube[] cubes,
    Neighbor neighbor,
    int distance)
```

큐브마다 복셀의 창 안 값의 분산을 구함.

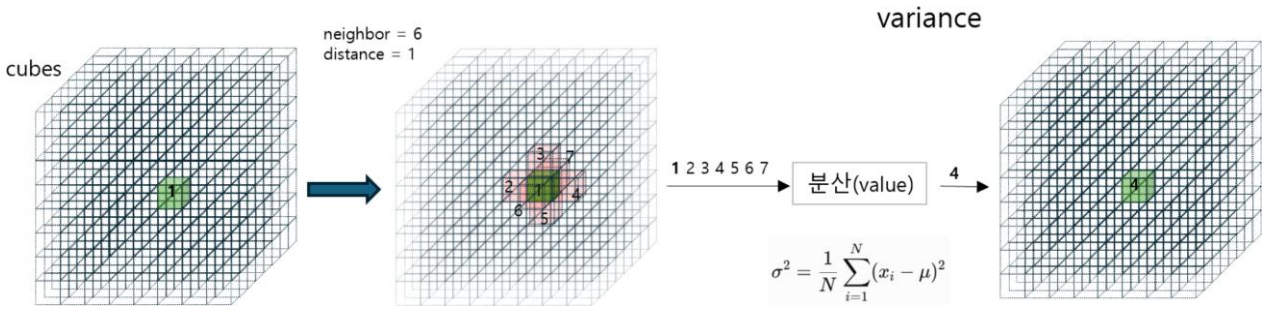
모집단 분산임 — 편차 제곱의 합을 창에서 찾은 값의 개수 `n` (센티넬을 뺀 이웃 + 자기 자신)으로 나눔.

`n - 1` 로 나누는 표본 분산이 아님.

창에 값이 하나뿐이면 0. `standardDeviation` 의 제곱과 같음.

variance

메서드	파라미터 / 반환	설명
variance	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 임의 복셀을 초점으로하여 거리만큼 이웃선택(자기자신복셀포함)하여 선택된 이웃들의 분산값을 초점복셀의 값으로 취하는 연산. (큐브들간 공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	Neighbor neighbor	이웃선택 방법: N6, N18, N26 중 하나.
	int distance)	이웃선택 거리: 1,2,3,... 중 하나. (복셀단위)
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 큐브별로 구해진 결과를 cubes 의 순서와 일치시켜 반환 반환타입: double
		bit/V2/V3/V4 타입에 대한 지원은 미정.



값을 2 의 거듭제곱으로 축소해 계산한 뒤 복원하므로 중간 계산은 넘치지 않음.

다만 분산은 값의 제곱 규모라 결과 자체가 double 을 넘을 수 있고, 그러면 `ArithmeticException` 을 던짐 (double 큐브의 값이 약 1e154 를 넘는 경우).

예제

시험 시나리오 TS-136-FOC(W4 창 분산)의 입력 파일이 있는 폴더에서 실행.

```
TS-136-FOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-136-FOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.Neighbor;
```

```
import com.voxel.operator.VoxelOpFocal;

public class Example {

    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpFocal.variance(cubes, Neighbor.N6, 1);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }

    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }

    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }

}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

모두 byte, short, int, float, double 중 하나여야 함

`neighbor` - 이웃선택 방법.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

`distance` - 이웃선택 거리 (복셀 단위).

1 이상

Returns:

큐브마다 분산을 담은 새 double 큐브.

`cubes` 와 길이·순서가 같음.

`'S'`이면 센티넬은 `Double.NaN`. 빈 리스트면 빈 배열

Throws:

`NullPointerException` - `cubes`, 그 원소, `neighbor` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 타입이 섞였을 때, `distance` 가 1 보다 작을 때, level·resolution·타입이 같은 큐브끼리 `(col, row, floor)`가 겹칠 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 분산이 double 을 넘을 때

13.7. majority API

```
public static BasicDataCube[] majority(BasicDataCube[] cubes,
    Neighbor neighbor,
    int distance)
```

큐브마다 복셀의 창 안에서 가장 많이 나온 값(다수결값)을 구함.

창의 값을 빈도로 세어 가장 많이 나온 값을 고름.

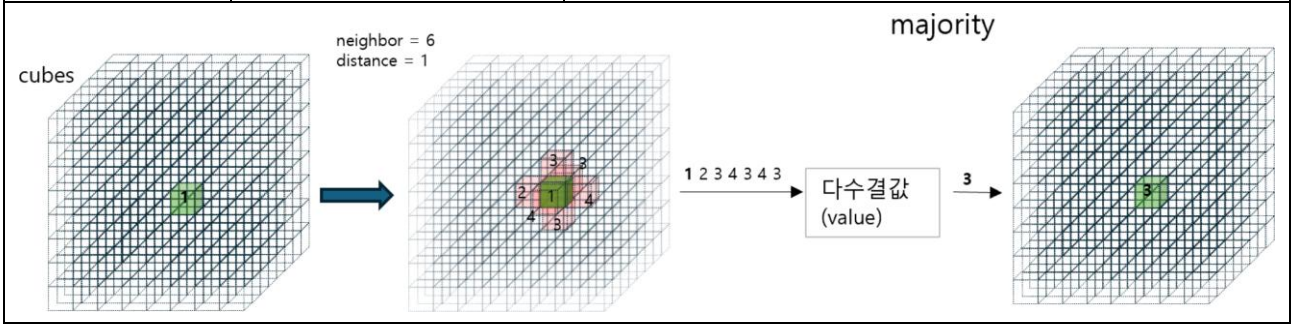
값이 같은지는 `==`로 비교함 (`-0.0` 과 `0.0` 은 같은 값).

- 가장 많이 나온 값이 하나면 그 값
- 가장 많이 나온 값이 여럿이면 그 값들의 평균 — 창의 값이 모두 다른 경우도 전부 빈도 1 로 같으므로 여기 들어가 창 전체의 평균이 됨
- 창에 값이 하나뿐이면 그 값 (다수결값 하나)

majority

메서드	파라미터 / 반환	설명
majority	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 임의 복셀을 초점으로하여 거리만큼 이웃선택(자기자신복셀포함)하여 선택된 이웃들의 다수결값을 초점복셀의 값으로 취하는 연산. (큐브들간 공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음) 다수결값을 찾을수 없는 경우: 선택된 이웃들의 값이 모두 다른 경우 -> 전체 평균값으로 처리 다수결값이 여러 개인 경우 -> 다수결값들의 평균값으로 처리 입력 큐브가 byte,short 일때 반환타입은 float 로 처리 입력 큐브가 int 일때 반환타입은 double 로 처리
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	Neighbor neighbor	이웃선택 방법: N6, N18, N26 중 하나.
	int distance)	이웃선택 거리: 1,2,3,... 중 하나. (복셀단위)
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 큐브별로 구해진 결과를 cubes 의 순서와 일치시켜 반환 반환타입: cubes 의 각 데이터 타입에 해당하는 타입으로 반환. (평균값으로 처리시: byte,short->float. int->double)

		bit/V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-139-FOC(W4 창 최빈값)의 입력 파일이 있는 폴더에서 실행.

```
TS-139-FOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-139-FOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.Neighbor;
import com.voxel.operator.VoxelOpFocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpFocal.majority(cubes, Neighbor.N6, 2);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
```

```
        }
    }

}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

모두 byte, short, int, float, double 중 하나여야 함

`neighbor` - 이웃선택 방법.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

`distance` - 이웃선택 거리 (복셀 단위).

1 이상

Returns:

큐브마다 다수결값을 담은 새 큐브.

`cubes` 와 길이·순서가 같음.

빈 리스트면 빈 배열

Throws:

`NullPointerException` - `cubes`, 그 원소, `neighbor` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 타입이 섞였을 때, `distance` 가 1 보다 작을 때, level·resolution·타입이 같은 큐브끼리 `(col, row, floor)`가 겹칠 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때

13.8. median API

```
public static BasicDataCube[] median(BasicDataCube[] cubes,
    Neighbor neighbor,
    int distance)
```

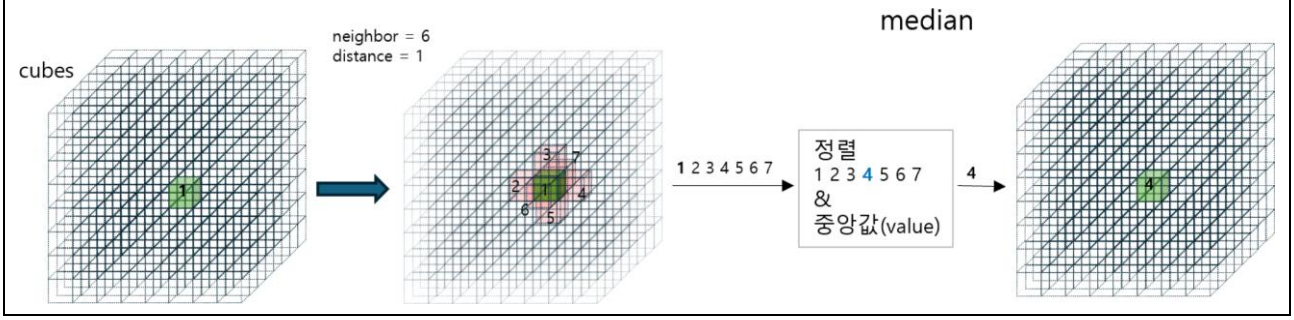
큐브마다 복셀의 창 안 값을 크기순으로 정렬해 가운데 값(중앙값)을 구함.

- 창의 값이 홀수 개면 정렬한 가운데 값
- 짝수 개면 가운데 두 값의 평균
- 창에 값이 하나뿐이면 그 값

median

메서드	파라미터 / 반환	설명
median	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 임의 복셀을 초점으로하여 거리만큼 이웃선택(자기자신복셀포함)하여 선택된 이웃들의 값을 크기순으로 정렬하여 중앙에 위치하는 값을 초점복셀의 값으로 취하는 연산.

		(큐브들간 공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음) 중앙값을 찾을수 없는 경우: 선택된 이웃들의 개수가 짝수개인 경우 -> 중앙 두개 값의 평균값으로 처리 입력 큐브가 byte,short 일때 반환타입은 float 로 처리 입력 큐브가 int 일때 반환타입은 double 로 처리
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	Neighbor neighbor	이웃선택 방법: N6, N18, N26 중 하나.
	int distance)	이웃선택 거리: 1,2,3,... 중 하나. (복셀단위)
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 큐브별로 구해진 결과를 cubes 의 순서와 일치시켜 반환 반환타입: cubes 의 각 데이터 타입에 해당하는 타입으로 반환. (평균값으로 처리시: byte,short->float. int->double)
		bit/V2/V3/V4 타입에 대한 지원은 미정.



큐브 면의 복셀은 그쪽에 맞닿은 큐브가 없으면 창의 값이 짝수 개가 되므로 (N6 은 6 개, N18 은 14 개, N26 은 18 개), 옆 큐브 없이 홀로 들어온 정수 타입 'V' 큐브는 면 어딘가에서 가운데 두 값의 합이 홀수이면 실수 타입으로 나옴.

예제

시험 시나리오 TS-142-FOC(W4 창 중앙값)의 입력 파일이 있는 폴더에서 실행.

```
TS-142-FOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-142-FOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.Neighbor;
import com.voxel.operator.VoxelOpFocal;


public class Example {
```

```
    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpFocal.median(cubes, Neighbor.N6, 1);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }

}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

모두 byte, short, int, float, double 중 하나여야 함

`neighbor` - 이웃선택 방법.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

`distance` - 이웃선택 거리 (복셀 단위).

1 이상

Returns:

큐브마다 중앙값을 담은 새 큐브.

`cubes` 와 길이·순서가 같음.

빈 리스트면 빈 배열

Throws:

`NullPointerException` - `cubes`, 그 원소, `neighbor` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 타입이 섞였을 때, `distance` 가 1 보다 작을 때, level·resolution·타입이 같은 큐브끼리 `(col, row, floor)`가 겹칠 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때

13.9. count API

```
public static BasicDataCube[] count(BasicDataCube[] cubes,
    Neighbor neighbor,
    int distance,
    Object minNumber,
    Object maxNumber)
```

큐브마다 복셀의 창 안에서 `minNumber <= 값 < maxNumber` 인 값이 몇 개인지 셈.

하한은 **포함**하고 상한은 **제외**함.

한쪽을 `null` 로 주면 그쪽 제한이 없고, 둘 다 `null` 이면 창에서 찾은 값의 개수(센티넬을 뺀 이웃 + 자기 자신)가 됨.

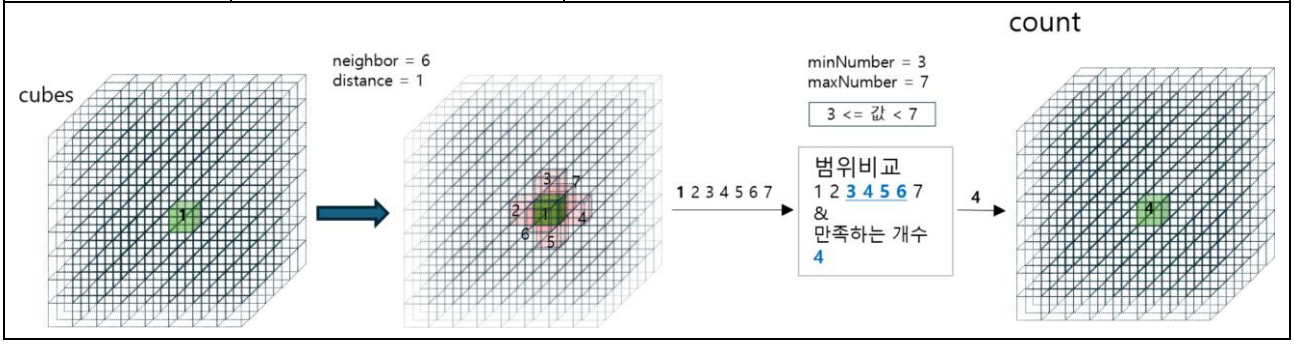
두 경계는 서로 타입이 달라도 되고 큐브 타입과 달라도 되며, 값을 double 로 바꿔 비교함.

무한대는 값으로 비교함 (`-Infinity` 하한은 제한 없음과 같음).

count

메서드	파라미터 / 반환	설명
count	(	입력된 레스터 데이터 큐브들에 대해서 각 큐브별로 큐브내 임의 복셀을 초점으로하여 거리만큼 이웃선택(자기자신복셀포함)하여 선택된 이웃들의 값을 비교하여 조건에 해당하는 복셀이 몇 개인지 세는 연산. (큐브들간 공간크기(level), 공간위치(col,row,floor), 해상도(resolution) 제약없음)
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. 지원타입: byte,short,int,float,double
	Neighbor neighbor	이웃선택 방법: N6, N18, N26 중 하나.
	int distance	이웃선택 거리: 1,2,3,... 중 하나. (복셀단위)
	Object minNumber	최소값 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber)	최대값 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 큐브별로 구해진 결과를 cubes 의 순서와 일치시켜 반환 반환타입: int (데이터블록코드는 'V'이며, 자신을 포함한 찾은 모든 이웃이

		sentinel 이면 0 값임)
		bit/V2/V3/V4 타입에 대한 지원은 미정.



센티넬은 창에서 빠지므로 세지 않음.

초점과 이웃이 모두 센티넬이면 셀 것이 없으므로 0 임 — 다른 연산과 달리 센티넬이 아님.

개수는 언제나 답이 있으므로 결과 큐브는 입력이 `'S'`여도 `'V'`이고 센티넬이 없음.

반환 타입은 입력과 무관하게 언제나 int 임.

예제

시험 시나리오 TS-145-FOC(W4 창 개수)의 입력 파일이 있는 폴더에서 실행.

```
TS-145-FOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-145-FOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.Neighbor;
import com.voxel.operator.VoxelOpFocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpFocal.count(cubes, Neighbor.N26, 1, 50, null);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }
```

```
    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }

}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

모두 byte, short, int, float, double 중 하나여야 함

`neighbor` - 이웃선택 방법.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

`distance` - 이웃선택 거리 (복셀 단위).

1 이상

`minNumber` - 하한.

이 값을 **포함**함(`minNumber <= 값`).

`Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나여야 하고 NaN 이 아니어야 함.

`null` 이면 하한 제한이 없음

`maxNumber` - 상한.

이 값을 **제외**함(`값 < maxNumber`).

`Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나여야 하고 NaN 이 아니어야 함.

`null` 이면 상한 제한이 없음

Returns:

큐브마다 개수를 담은 새 `'V'` int 큐브.

`cubes` 와 길이·순서가 같음.

빈 리스트면 빈 배열

Throws:

`NullPointerException` - `cubes`, 그 원소, `neighbor` 가 `null` 일 때

`IllegalArgumentException` - 지원하지 않는 타입이 섞였을 때, `distance` 가 1 보다 작을 때, `minNumber` 또는 `maxNumber` 가 위 다섯 가지 객체가 아니거나 NaN 일 때, 둘 다 주었는데 `minNumber >= maxNumber` 일 때, level·resolution·타입이 같은 큐브끼리 `(col, row, floor)`가 겹칠 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때

14. VoxelOpLocal [레스터]

클래스 설명

```
public final class VoxelOpLocal
    extends Object
```

[레스터] 지역 연산자.

같은 자리에 겹쳐 있는 여러 큐브의 같은 복셀 위치 값을 모아 하나의 값으로 바꿈.

입력은 **같은 level, 같은 `(col, row, floor)`, 같은 resolution** 을 가진 큐브 리스트이며 타입은 서로 달라도 됨.

복셀 위치 `(x, y, z)` 하나가 "지역"이고, 그 자리의 큐브별 값 (큐브마다 가중치와 바이어스를 적용한 `값 * weight + bias`)이 지역의 구성원임.

큐브 타입은 byte, short, int, float, double 만 받음.

```
VoxelOpLocal.mean(cubes, null, null);                                    // 큐브들의 같은 자리 평균
VoxelOpLocal.mean(cubes, new Object[] {0.5, 0.3, 0.2}, null);            // 가중 평균
VoxelOpLocal.mean(new BasicDataCube[] {a, b}, null, new Object[] {null, 10}); // b 에만 +10
```

인스턴스를 만들 수 없으며 연산마다 정적 메서드를 제공함.

결과는 언제나 단일 큐브임.

입력 큐브는 변경하지 않고 새 큐브를 만들어 돌려줌.

모두 상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

연산 목록

부를 수 있는 메서드

메서드	계산	반환
`mean(cubes, weights, biases)`	지역 값의 평균	double 큐브 하나
`sum(cubes, weights, biases)`	지역 값의 합	큐브 중 큰 타입에서 시작해 넘치면 승격한 큐브 하나
`min(cubes, weights, biases)`	지역 값의 최소	큐브 중 큰 타입 (가중치·바이어스로 벗어날 때만 승격)
`max(cubes, weights, biases)`	지역 값의 최대	큐브 중 큰 타입 (가중치·바이어스로 벗어날 때만 승격)
`standardDeviation(cubes, weights, biases)`	지역 값의 표준편차 (모집단, `n` 으로 나눔)	double 큐브 하나
`variance(cubes, weights, biases)`	지역 값의 분산 (모집단, `n` 으로 나눔)	double 큐브 하나

`majority(cubes, weights, biases)`	지역 값의 다수결값 (동률이면 그 값들의 평균)	큐브 중 큰 타입 (동률 평균이 소수이면 byte·short 는 float, int 는 double)
`median(cubes, weights, biases)`	지역 값의 중앙값 (짝수 개면 가운데 두 값의 평균)	큐브 중 큰 타입 (가운데 두 값의 평균이 소수이면 byte·short 는 float, int 는 double)
`pow(cubes, weights, biases, option)`	지역 값을 각각 제곱한 뒤 `option` 에 따라 평균 또는 다수결값	큐브 중 큰 타입에서 시작해 넘치면 승격한 큐브 하나 (평균이 소수이면 double)
`sqrt(cubes, weights, biases, option)`	지역 값의 제곱근을 각각 구한 뒤 `option` 에 따라 평균 또는 다수결값	double 큐브 하나
`log(cubes, weights, biases, option)`	지역 값의 자연로그를 각각 구한 뒤 `option` 에 따라 평균 또는 다수결값	double 큐브 하나
`sin(cubes, weights, biases, option1, option2)`	지역 값의 사인을 각각 구한 뒤 (`option1`: 라디안/각도) `option2` 에 따라 평균 또는 다수결값	double 큐브 하나
`cos(cubes, weights, biases, option1, option2)`	지역 값의 코사인을 각각 구한 뒤 (`option1`: 라디안/각도) `option2` 에 따라 평균 또는 다수결값	double 큐브 하나
`tan(cubes, weights, biases, option1, option2)`	지역 값의 탄젠트를 각각 구한 뒤 (`option1`: 라디안/각도) `option2` 에 따라 평균 또는 다수결값	double 큐브 하나
`countCompare(cubes, weights, biases, compareNumber, compareOp)`	지역 값 중 `compareNumber` 와의 비교(`compareOp`)가 참인 값의 **개수**	int 큐브 하나 (언제나 `'V'`, 모든 큐브가 센티넬인 자리는 0)
`classifyCompare(cubes, weights, biases, option, compareNumber, compareOp, trueLevel, falseLevel)`	지역 값의 평균 또는 다수결값(`option`)이 `compareNumber` 와의 비교(`compareOp`)에 해당하면 `trueLevel`, 아니면 `falseLevel`	`trueLevel` 과 `falseLevel` 중 큰 타입의 큐브 하나 (값은 두 분류값뿐)
`classifyRange(cubes, weights, biases, option, arrayRangeLevel, otherLevel)`	지역 값의 평균 또는 다수결값(`option`)이 `[start, end, level]` 범위 중 처음으로 `start <= 값 < end` 인 것의 `level`, 어디에도 없으면 `otherLevel`	`level` 들과 `otherLevel` 중 큰 타입의 큐브 하나 (값은 분류값뿐)
`classifyScore(cubes, arrayRangeScore, arrayRangeLevel, otherLevel)`	큐브마다 자기 점수표(`arrayRangeScore[k]`)로 값을 점수로 바꿔 지역에서 **합산**한 뒤, 그 합을 `arrayRangeLevel` 로 분류 (가중치·바이어스 없음)	`level` 들과 `otherLevel` 중 큰 타입의 큐브 하나 (값은 분류값뿐)

가중치와 바이어스

`weights[k]`, `biases[k]`는 `cubes[k]`에 적용되며 지역에 들어가는 값은 `복셀값 * weights[k] + biases[k]`임.
계산은 double 로 함.

- `weights` 가 `null` 이면 모든 큐브가 가중치 없음(1), `biases` 가 `null` 이면 모든 큐브가 바이어스 없음(0)
- 배열을 주면 길이가 `cubes` 와 같아야 하고, 원소가 `null` 이면 그 큐브만 미적용
- 원소는 `Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나이며 유한한 값이어야 함. 타입은 큐브 타입과 달라도 되고 결과 타입에 영향을 주지 않음
- 곱하고 더한 값이 double 을 넘으면 `ArithmeticException`

`classifyScore` 만 가중치·바이어스를 받지 않음 — 큐브마다 주는 점수 표가 그 자리를 대신함.

옵션

옵션은 모두 `int` 상수이고 **이 클래스에서 선언**하며, 정해진 값이 아니면 `IllegalArgumentException` 을 던짐.
열 개의 값이 전부 다르므로 int 인자를 서로 바꿔 넣으면 (예: `option1` 과 `option2`, `option` 과 `compareOp`) 언제나 `IllegalArgumentException` 이 남.

옵션 상수

상수	값	어느 인자	뜻
`TRIG_FUNC_RADIAN`	1	`option1` (sin, cos, tan)	가중치·바이어스를 적용한 값을 **라디안**으로 봄
`TRIG_FUNC_DEGREE`	2	`option1` (sin, cos, tan)	**각도**로 봄 (90 도의 배수에서 정확한 0, ±1)
`LOCAL_OPTION_MEAN`	3	`option` (pow, sqrt, log, classifyCompare, classifyRange) / `option2` (sin, cos, tan)	큐브마다 구한 값들을 지역에서 **평균**으로 줄임
`LOCAL_OPTION_MAJORITY`	4	`option` (pow, sqrt, log, classifyCompare, classifyRange) / `option2` (sin, cos, tan)	**다수결값**으로 줄임 — `majority` 와 같은 규칙 (동률이면 그 값들의 평균, 모두 다르면 전체 평균)
`LOCAL_OP_EQUAL`	5	`compareOp` (countCompare, classifyCompare)	`연산값 == compareNumber`
`LOCAL_OP_NOT_EQUAL`	6	`compareOp` (countCompare, classifyCompare)	`연산값 != compareNumber`

LOCAL_OP_GREATER_THAN	7	compareOp (countCompare, classifyCompare)	연산값 > compareNumber
LOCAL_OP_LESS_THAN	8	compareOp (countCompare, classifyCompare)	연산값 < compareNumber
LOCAL_OP_GREATER_THAN_EQUAL	9	compareOp (countCompare, classifyCompare)	연산값 >= compareNumber
LOCAL_OP_LESS_THAN_EQUAL	10	compareOp (countCompare, classifyCompare)	연산값 <= compareNumber

TRIG_FUNC_*는 산술연산자(VoxelOpArithmetic)의 같은 이름 상수와 이름·값이 같고, LOCAL_OP_*는 비교연산자(VoxelOpComparison)의 COMPARE_OP_*와 뜻은 같지만 **값이 다름**(5~10).
어느 쪽이든 다른 클래스의 상수가 아니라 이 클래스의 상수를 넘겨야 함.
비교는 연산값(countCompare 는 지역 값 하나하나, classifyCompare 는 지역의 평균 또는 다수결값)을 왼쪽, compareNumber 를 오른쪽에 두고 함.

지역에서 빠지는 자리

'S' 큐브의 센티넬 자리(float/double 은 NaN 도 센티넬)는 지역에 넣지 않고 **나머지 큐브의 값만으로** 계산함.
지역에 값이 하나뿐이면 그 값으로 결과를 정함.
모든 큐브가 그 자리에서 센티넬일 때만 결과가 센티넬임 (개수를 세는 countCompare 는 센티넬 대신 0).
classifyScore 에서도 센티넬 자리는 점수 0 이 아니라 합산에서 빠지는 것이며, 결과는 같음.
'V' 큐브가 하나라도 있으면 어느 지역도 비지 않음.

던지는 예외

예외와 원인

예외	언제
NullPointerException	cubes 나 그 원소가 null 일 때, compareNumber·분류값·arrayRangeLevel·arrayRangeScore 와 그 원소·start·end·score 가 null 일 때
IllegalArgumentException	cubes 가 비었을 때, 큐브 타입이 bit 나 벡터일 때, 큐브끼리 level·(col, row, floor)·resolution 이 다를 때, weights/biases 의 길이가 cubes 와 다를 때, 그 원소가 Byte/Short/Integer/Float/Double 이 아니거나 NaN·무한대일 때, option 이 LOCAL_OPTION_MEAN·LOCAL_OPTION_MAJORITY 중 하나가

	아닐 때, 삼각함수의 `option1` 이 `TRIG_FUNC_RADIAN`·`TRIG_FUNC_DEGREE` 중 하나가 아닐 때, `compareNumber` 가 Byte/Short/Integer/Float/Double 이 아니거나 NaN 일 때, `compareOp` 가 `LOCAL_OP_*` 여섯 중 하나가 아닐 때, 분류값이 Byte/Short/Integer/Float/Double 이 아니거나 NaN·무한대일 때, `arrayRangeLevel`·`arrayRangeScore` 표의 원소가 세 개짜리가 아닐 때, `start`·`end` 가 Byte/Short/Integer/Float/Double 이 아니거나 NaN 일 때, `score` 가 Byte/Short/Integer/Float/Double 이 아니거나 NaN·무한대일 때, `arrayRangeScore` 의 길이가 `cubes` 와 다를 때
`ArithmeticException`	`'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이나 계산 결과가 double 을 넘을 때 (double 큐브의 합·분산·제곱, `classifyScore` 의 점수 합이 넘치는 경우), 가중치·바이어스를 적용한 값이 연산의 정의역 밖일 때 (`sqrt` 의 음수, `log` 의 0 이하, 각도 모드 `tan` 의 90 + 180k 도)

결과 큐브의 헤더

입력에서 물려받는 것과 다시 정하는 것

헤더 필드	결과값
`level`, `col`, `row`, `floor`, `resolution`	입력 큐브들의 것 (모두 같음)
`datatype`	연산마다 정해짐 — 언제나 double 인 연산, 개수를 세어 언제나 int 인 연산(`countCompare`), 분류값 중 큰 타입인 연산(`classifyCompare`, `classifyRange`, `classifyScore`), 큐브 중 큰 타입에서 시작해 넘치면 승격하는 연산이 있음. 위 <연산 목록>의 반환 열과 아래 <반환 타입과 승격> 참고
`dataBlockCode`	입력 중 `'V'`가 하나라도 있으면 `'V'` — 그 큐브가 모든 자리에 값을 가지므로 어느 지역도 비지 않음. **모두 `'S'`일 때만**`'S'`. `countCompare` 는 빈 지역을 0 으로 세므로 입력과 무관하게 언제나 `'V'`
`sentinel`	`'S'`이면 byte/short/int 는 반환 타입의 최대값, float/double 은 NaN (입력의 센티넬 값은 쓰지 않음). `'V'`이면 없음(`null`)

반환 타입과 승격

입력 타입을 따르는 연산(`sum`, `min`, `max`, `majority`, `median`, `pow`)은 **큐브 중 가장 큰 타입**(byte < short < int < float < double)에서 시작해, 계산 결과가 그 타입에 담기지 않으면 아래 순서로 승격함.

```
byte -> short -> int -> double
float -> double
```

반환 타입은 **큐브 단위로 하나**임.

복셀 하나만 넘쳐도 큐브 전체가 승격됨.

double 에서도 넘치면 `ArithmeticException` 을 던짐.

가중치·바이어스는 객체가 `Integer` 인지 `Double` 인지가 아니라 **곱하고 더해서 나온 값**으로 판단함 — 정수 타입 큐브라도 그 값이 정수가 아니면 (예: int 큐브에 가중치 0.5) 그 큐브는 double 로 치므로 시작 타입이 double 이 됨.

값이 정수이면 (예: 가중치 2, 또는 `2.0`) 타입을 바꾸지 않고, 넘칠 때만 사다리로 승격함.

결과가 `'S'`(모든 입력이 `'S'`)이고 정수 타입이면 최대값은 센티넬 자리로 예약되어 값으로 쓸 수 없음.

계산 결과가 그 값이면 넘친 것으로 보고 승격함.

예를 들어 byte 큐브들의 합이 127 이면 `Byte.MAX_VALUE` 와 같으므로 결과는 short 가 됨.

`'V'`이면 예약하지 않음.

float, double 은 센티넬이 NaN 이라 최대값을 예약하지 않음.

입력값 중 하나를 고르는 연산(`min`, `max`)은 가중치·바이어스가 없으면 계산으로 넘치는 일이 없으므로, 승격은 가중치·바이어스로 값이 범위를 벗어나거나 소수가 될 때, 그리고 이 예약에 걸릴 때만 일어남.

`majority` 와 `median` 은 입력값이 모두 정수인데 평균 처리(다수결 동률, 짝수 개의 가운데 두 값)로 소수가 나온 자리가 있으면 double 이 아니라 **byte, short 는 float, int 는 double** 로 감.

그런 자리가 하나도 없으면 입력 타입을 유지함.

`pow` 는 이 규칙이 없어서 제곱의 평균(또는 동률 평균)이 소수이면 double 임.

분류 연산(`classifyCompare`, `classifyRange`, `classifyScore`)의 결과에는 분류값만 들어가므로 큐브 타입은 보지 않고 **분류값 중 큰 타입**(객체 타입 기준, byte < short < int < float < double)이 반환 타입임 — `classifyCompare` 는 `trueLevel` 과 `falseLevel`, `classifyRange`·`classifyScore` 는 모든 `level` 과 결과에 같이 담기는 `otherLevel` 까지 (`score` 는 결과에 안 담기므로 타입과 무관).

두 군데서만 그보다 올라감 — 결과가 `'S'`인데 분류값이 그 정수 타입의 최대값(예약된 센티넬 자리)과 같으면 위 사다리로 승격하고, `Integer` 분류값이 float 에 정확히 담기지 않으면 (224 을 넘는 값) float 대신 double 임.

분류값이 바뀌어 담기는 일은 없음.

그러므로 결과 큐브의 값을 읽기 전에 `datatype` 을 확인해야 함.

센티넬과 NaN

- `'V'` 큐브의 float/double NaN 은 값이 될 수 없으므로 `ArithmeticException`
- 복셀값이 무한대이면 `ArithmeticException`

Field Summary

Fields

Modifier and Type	Field	Description
static final int	LOCAL_OP_EQUAL	compareOp 값 — == 비교.
static final int	LOCAL_OP_GREATER_THAN	compareOp 값 — > 비교 (연산값 > compareNumber).
static final int	LOCAL_OP_GREATER_THAN_EQUAL	compareOp 값 — >= 비교 (연산값 >= compareNumber).
static final int	LOCAL_OP_LESS_THAN	compareOp 값 — < 비교 (연산값 < compareNumber).
static final int	LOCAL_OP_LESS_THAN_EQUAL	compareOp 값 — <= 비교 (연산값 <= compareNumber).
static final int	LOCAL_OP_NOT_EQUAL	compareOp 값 — != 비교.
static final int	LOCAL_OPTION_MAJORITY	option 값 — 큐브마다 구한 값들을 지역에서 **다수결값**으로 줄임.
static final int	LOCAL_OPTION_MEAN	option 값 — 큐브마다 구한 값들을 지역에서 **평균**으로 줄임.
static final int	TRIG_FUNC_DEGREE	삼각함수의 option1 값 — 가중치·바이어스를 적용한 값을 **각도**로 봄.
static final int	TRIG_FUNC_RADIAN	삼각함수의 option1 값 — 가중치·바이어스를 적용한 값을 **라디안**으로 봄.

Method Summary

Modifier and Type	Method	Description
static BasicDataCube	classifyCompare(BasicDataCube[] cubes, Object[] weights, Object[] biases, int option, Object compareNumber, int compareOp, Object trueLevel, Object falseLevel)	지역 값의 평균 또는 다수결값이 비교조건에 해당하는지로 **두 값으로 분류**.
static BasicDataCube	classifyRange(BasicDataCube[] cubes, Object[] weights, Object[] biases, int option, Object[][] arrayRangeLevel, Object otherLevel)	지역 값의 평균 또는 다수결값이 어느 범위에 드는지로 **여러 값으로 분류**.
static BasicDataCube	classifyScore(BasicDataCube[] cubes, Object[][][] arrayRangeScore, Object[][] arrayRangeLevel, Object otherLevel)	큐브마다 값을 **점수**로 바꿔 지역에서 **합산**하고, 그 합이 어느 범위에 드는지로 **여러 값으로 분류**.
static BasicDataCube	cos(BasicDataCube[] cubes, Object[] weights, Object[] biases, int option1, int option2)	큐브들의 같은 복셀 위치 값의 코사인을 각각 구한 뒤, 그 코사인값들의 평균 또는 다수결값을 구함.
static BasicDataCube	countCompare(BasicDataCube[] cubes, Object[] weights, Object[] biases, Object compareNumber, int	지역 값 중 비교조건을 만족하는 값의 **개수**.

	compareOp)	
static BasicDataCube	log(BasicDataCube[] cubes, Object[] weights, Object[] biases, int option)	큐브들의 같은 복셀 위치 값의 자연로그를 각각 구한 뒤, 그 로그값들의 평균 또는 다수결값을 구함.
static BasicDataCube	majority(BasicDataCube[] cubes, Object[] weights, Object[] biases)	큐브들의 같은 복셀 위치 값 중 가장 많이 나온 값(다수결값)을 구함.
static BasicDataCube	max(BasicDataCube[] cubes, Object[] weights, Object[] biases)	큐브들의 같은 복셀 위치 값의 최대값을 구함.
static BasicDataCube	mean(BasicDataCube[] cubes, Object[] weights, Object[] biases)	큐브들의 같은 복셀 위치 값의 평균을 구함.
static BasicDataCube	median(BasicDataCube[] cubes, Object[] weights, Object[] biases)	큐브들의 같은 복셀 위치 값을 크기순으로 늘어놓았을 때 가운데 값(중앙값)을 구함.
static BasicDataCube	min(BasicDataCube[] cubes, Object[] weights, Object[] biases)	큐브들의 같은 복셀 위치 값의 최소값을 구함.
static BasicDataCube	pow(BasicDataCube[] cubes, Object[] weights, Object[] biases, int option)	큐브들의 같은 복셀 위치 값을 각각 제곱한 뒤, 그 제곱값들의 평균 또는 다수결값을 구함.
static BasicDataCube	sin(BasicDataCube[] cubes, Object[] weights, Object[] biases, int option1, int option2)	큐브들의 같은 복셀 위치 값의 사인을 각각 구한 뒤, 그 사인값들의 평균 또는 다수결값을 구함.
static BasicDataCube	sqrt(BasicDataCube[] cubes, Object[] weights, Object[] biases, int option)	큐브들의 같은 복셀 위치 값의 제곱근을 각각 구한 뒤, 그 제곱근들의 평균 또는 다수결값을 구함.
static BasicDataCube	standardDeviation(BasicDataCube[] cubes, Object[] weights, Object[] biases)	큐브들의 같은 복셀 위치 값의 표준편차를 구함.
static BasicDataCube	sum(BasicDataCube[] cubes, Object[] weights, Object[] biases)	큐브들의 같은 복셀 위치 값의 합을 구함.
static BasicDataCube	tan(BasicDataCube[] cubes, Object[] weights, Object[] biases, int option1, int option2)	큐브들의 같은 복셀 위치 값의 탄젠트를 각각 구한 뒤, 그 탄젠트값들의 평균 또는 다수결값을 구함.
static BasicDataCube	variance(BasicDataCube[] cubes, Object[] weights, Object[] biases)	큐브들의 같은 복셀 위치 값의 분산을 구함.

Field Details

TRIG_FUNC_RADIAN

```
public static final int TRIG_FUNC_RADIAN
```

삼각함수의 `option1` 값 — 가중치·바이어스를 적용한 값을 **라디안**으로 봄.

값은 `1`.

`Math.PI / 2` 가 90 도임.

See Also:

- Constant Field Values

TRIG_FUNC_DEGREE

```
public static final int TRIG_FUNC_DEGREE
```

삼각함수의 `option1` 값 — 가중치·바이어스를 적용한 값을 **각도**로 봄.

값은 `2`.

See Also:

- Constant Field Values

LOCAL_OPTION_MEAN

```
public static final int LOCAL_OPTION_MEAN
```

`option` 값 — 큐브마다 구한 값들을 지역에서 **평균**으로 줄임.

값은 `3`.

See Also:

- Constant Field Values

LOCAL_OPTION_MAJORITY

```
public static final int LOCAL_OPTION_MAJORITY
```

`option` 값 — 큐브마다 구한 값들을 지역에서 **다수결값**으로 줄임.

값은 `4`.

`majority` 와 같은 규칙 — 가장 많이 나온 값, 동률이면 그 값들의 평균, 모두 다르면 전체 평균.

See Also:

- Constant Field Values

LOCAL_OP_EQUAL

```
public static final int LOCAL_OP_EQUAL
```

`compareOp` 값 — `==` 비교. 값은 `5`.

See Also:

- Constant Field Values

LOCAL_OP_NOT_EQUAL

```
public static final int LOCAL_OP_NOT_EQUAL
```

`compareOp` 값 — `!=` 비교. 값은 `6`.

See Also:

- Constant Field Values

LOCAL_OP_GREATER_THAN

```
public static final int LOCAL_OP_GREATER_THAN
```

compareOp 값 — > 비교 (연산값 > compareNumber). 값은 7.

See Also:

- Constant Field Values

LOCAL_OP_LESS_THAN

```
public static final int LOCAL_OP_LESS_THAN
```

compareOp 값 — < 비교 (연산값 < compareNumber). 값은 8.

See Also:

- Constant Field Values

LOCAL_OP_GREATER_THAN_EQUAL

```
public static final int LOCAL_OP_GREATER_THAN_EQUAL
```

compareOp 값 — >= 비교 (연산값 >= compareNumber). 값은 9.

See Also:

- Constant Field Values

LOCAL_OP_LESS_THAN_EQUAL

```
public static final int LOCAL_OP_LESS_THAN_EQUAL
```

compareOp 값 — <= 비교 (연산값 <= compareNumber). 값은 10.

See Also:

- Constant Field Values

14.1. mean API

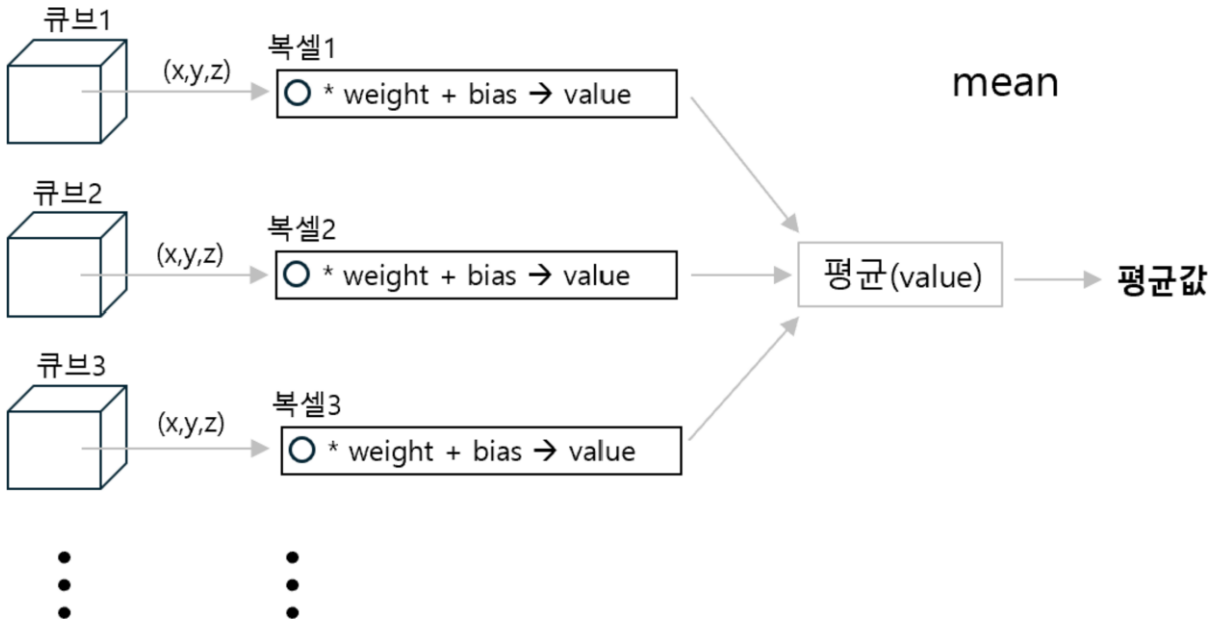
```
public static BasicDataCube mean(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases)
```

큐브들의 같은 복셀 위치 값의 평균을 구함.

mean

메서드	파라미터 / 반환	설명
mean	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 평균값을 구하는 연산.

	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases)	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: double (입력으로 들어온 cubes 가 모두 'S'일때 'S', 그렇지 않으면 'V'로 함) (입력으로 들어온 cubes 가 모두 'S'일때의 센티널: * byte,short,int 일때: 해당타입의 최대값 (주의: 오버플로 체크시 해당타입의 최대값 고려할것) (참고: int 에서 double 로 타입승격시 Double.NaN 사용) * float 일때: Float.NaN (0x7FC00000). (타입승격시 Double.NaN) * double 일때: Double.NaN (0x7FF8000000000000) (참고: 이후에 이와같은 경우 발생시 이와 같은 방식으로 처리)
		bit/V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-148-LOC(2 겹 평균)의 입력 파일이 있는 폴더에서 실행.

```
TS-148-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-148-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpLocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        Object[] weights = null;
        Object[] biases = null;
        BasicDataCube result = VoxelOpLocal.mean(cubes, weights, biases);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

Returns:

평균을 담은 새 double 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때

14.2. sum API

```
public static BasicDataCube sum(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases)
```

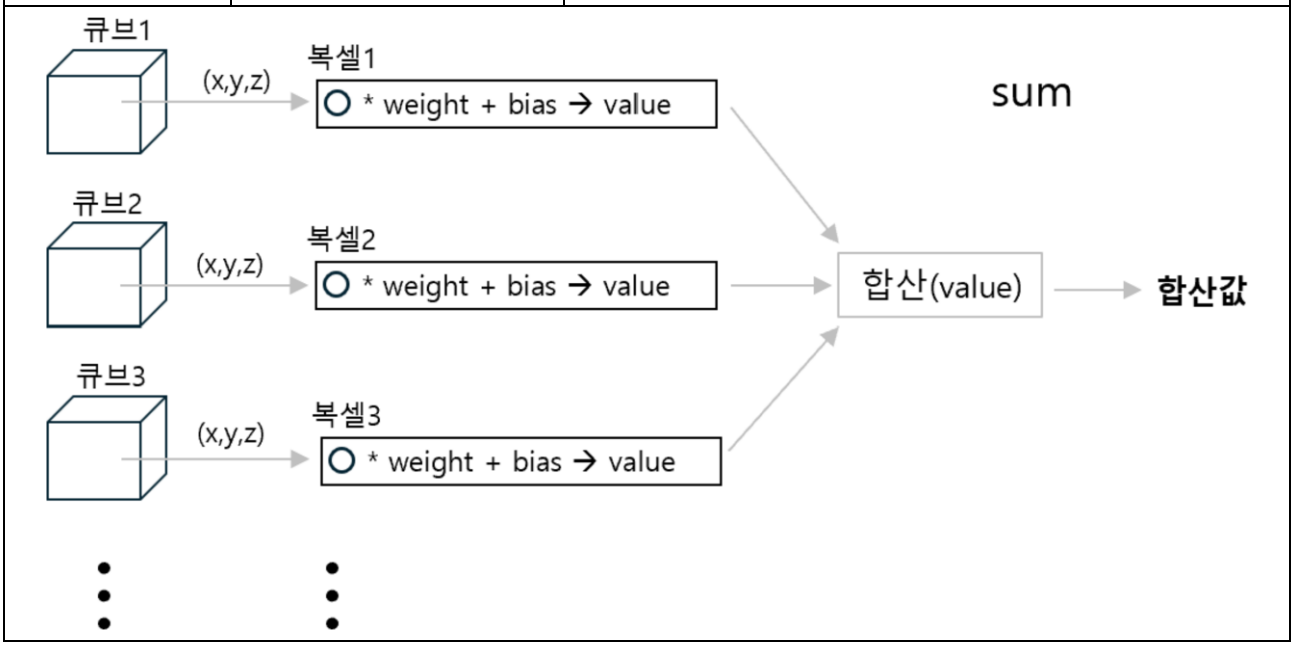
큐브들의 같은 복셀 위치 값의 합을 구함.

합은 double 로 구하고 정해진 반환 타입으로 저장함.

sum

메서드	파라미터 / 반환	설명
sum	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 합산값을 구하는 연산.
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases)	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: cubes 중에서 큰쪽의 타입을 반환.
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환.

		(byte->short, short->int, int->double, float->double, double->예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-152-LOC(2 겹 합)의 입력 파일이 있는 폴더에서 실행.

```
TS-152-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-152-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpLocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        Object[] weights = null;
        Object[] biases = null;
        BasicDataCube result = VoxelOpLocal.sum(cubes, weights, biases);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
```

```
    }

}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

Returns:

합을 담은 새 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이나 합이 double 을 넘을 때

14.3. min API

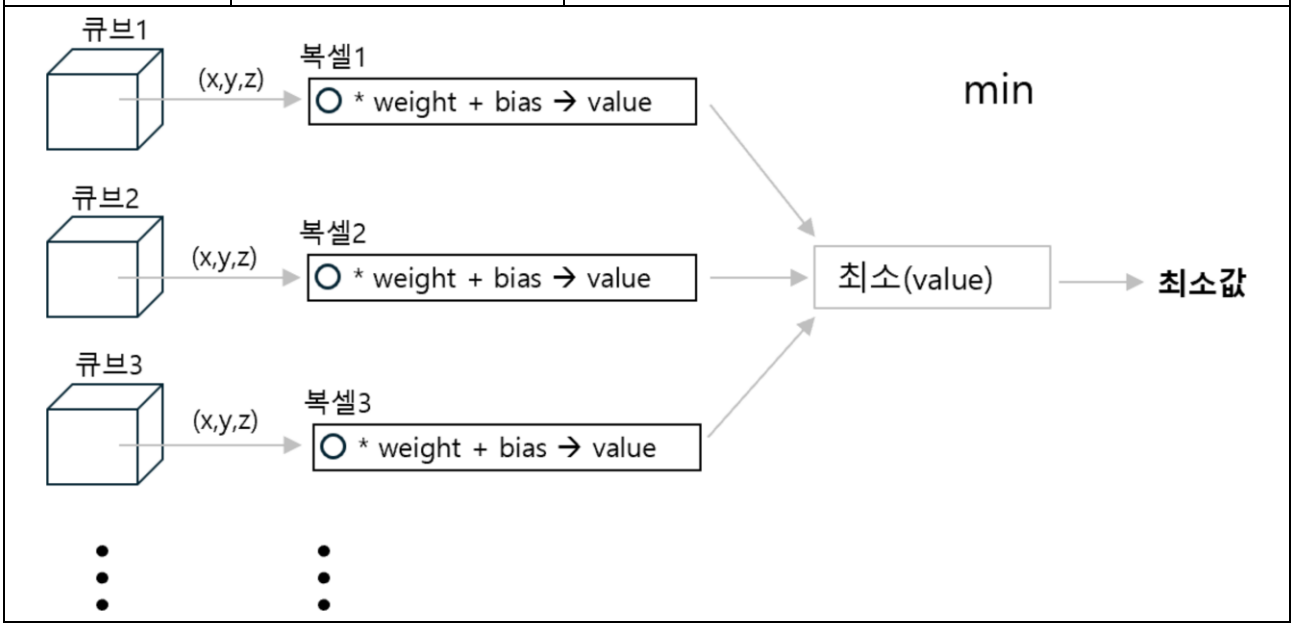
```
public static BasicDataCube min(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases)
```

큐브들의 같은 복셀 위치 값의 최소값을 구함.

min

메서드	파라미터 / 반환	설명
min	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 최소값을 구하는 연산.

	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases)	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: cubes 중에서 큰쪽의 타입을 반환.
		bit/V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-155-LOC(2 겹 최소)의 입력 파일이 있는 폴더에서 실행.

```
TS-155-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-155-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpLocal;
```

```
public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        Object[] weights = null;
        Object[] biases = null;
        BasicDataCube result = VoxelOpLocal.min(cubes, weights, biases);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

Returns:

최소값을 담은 새 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때

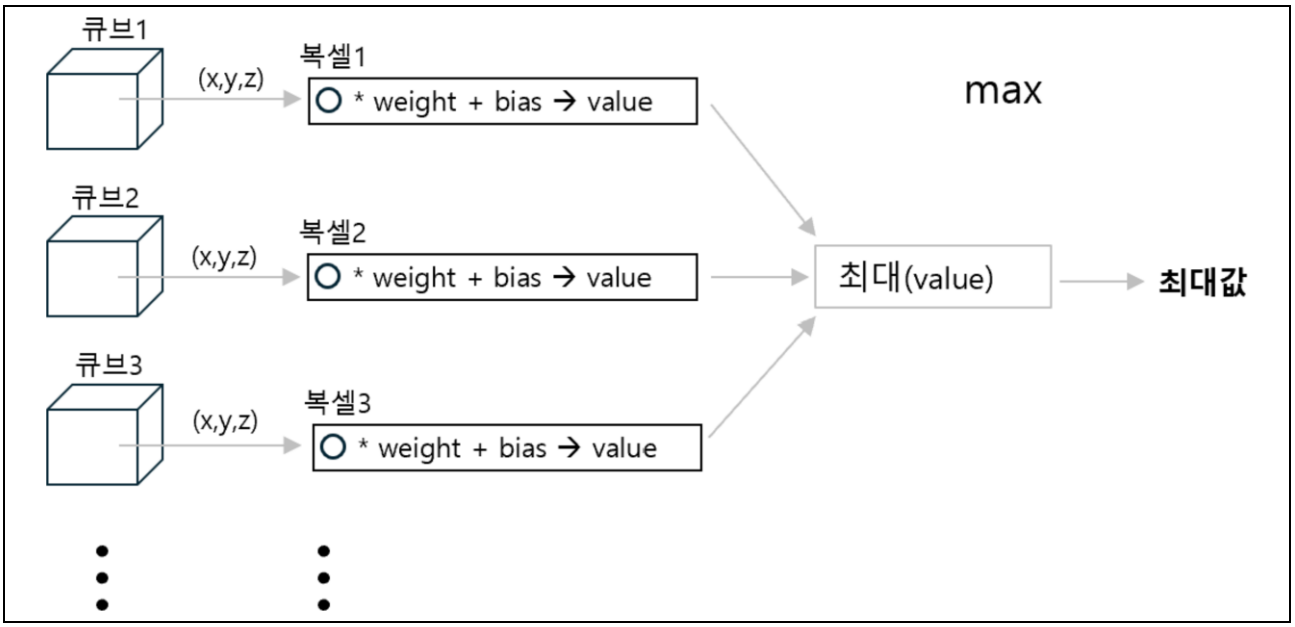
14.4. max API

```
public static BasicDataCube max(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases)
```

큐브들의 같은 복셀 위치 값의 최대값을 구함.

max

메서드	파라미터 / 반환	설명
max	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 최대값을 구하는 연산.
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases)	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: cubes 중에서 큰쪽의 타입을 반환.
		bit/V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-158-LOC(2 겹 최대)의 입력 파일이 있는 폴더에서 실행.

```
TS-158-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-158-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpLocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        Object[] weights = null;
        Object[] biases = null;
        BasicDataCube result = VoxelOpLocal.max(cubes, weights, biases);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

Returns:

최대값을 담은 새 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때

14.5. standardDeviation API

```
public static BasicDataCube standardDeviation(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases)
```

큐브들의 같은 복셀 위치 값의 표준편차를 구함.

모집단 표준편차임 — 편차 제곱의 합을 지역에서 찾은 값의 개수 `n`(센티넬을 뺀 큐브 수)으로 나누고 제곱근을 취함.

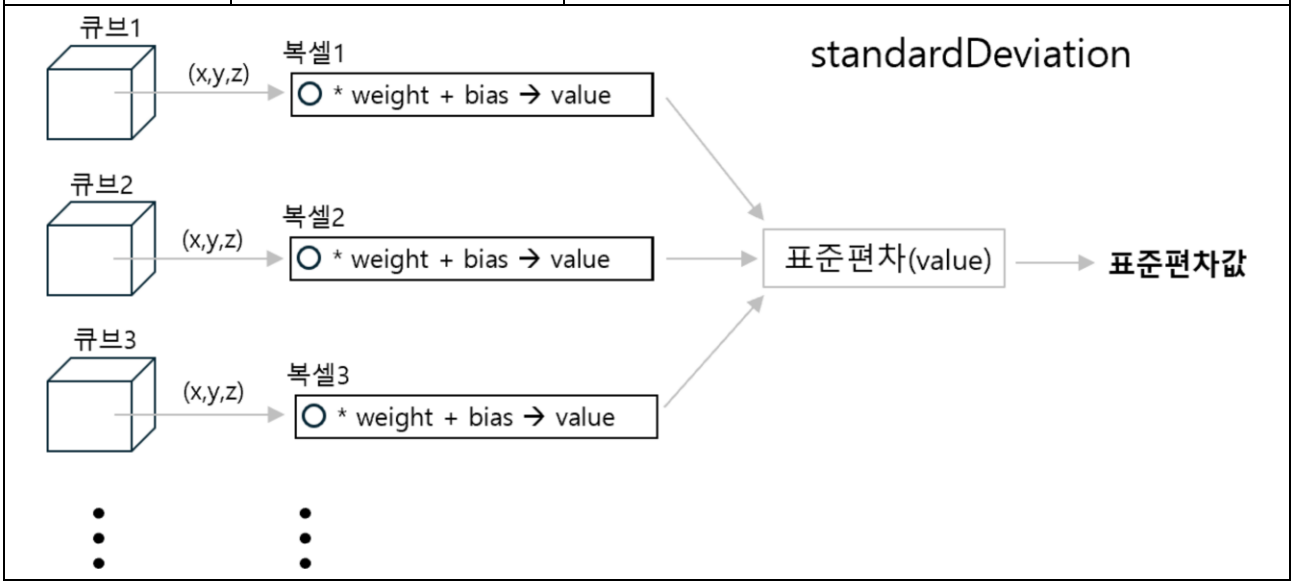
`n - 1` 로 나누는 표본 표준편차가 아님.

지역에 값이 하나뿐이면 0.

standardDeviation

메서드	파라미터 / 반환	설명
standardDeviation	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias)

		표준편차값을 구하는 연산.
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases)	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: double
		bit/V2/V3/V4 타입에 대한 지원은 미정.



값을 2 의 거듭제곱으로 축소해 계산한 뒤 복원하므로 값이 double 끝에 가까워도 넘치지 않음.

예제

시험 시나리오 TS-160-LOC(2 겹 표준편차)의 입력 파일이 있는 폴더에서 실행.

```
TS-160-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-160-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
```

```
import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpLocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        Object[] weights = null;
        Object[] biases = null;
        BasicDataCube result = VoxelOpLocal.standardDeviation(cubes, weights, biases);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

Returns:

표준편차를 담은 새 double 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때

14.6. variance API

```
public static BasicDataCube variance(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases)
```

큐브들의 같은 복셀 위치 값의 분산을 구함.

모집단 분산임 — 편차 제곱의 합을 지역에서 찾은 값의 개수 `n`(센티넬을 뺀 큐브 수)으로 나눔.

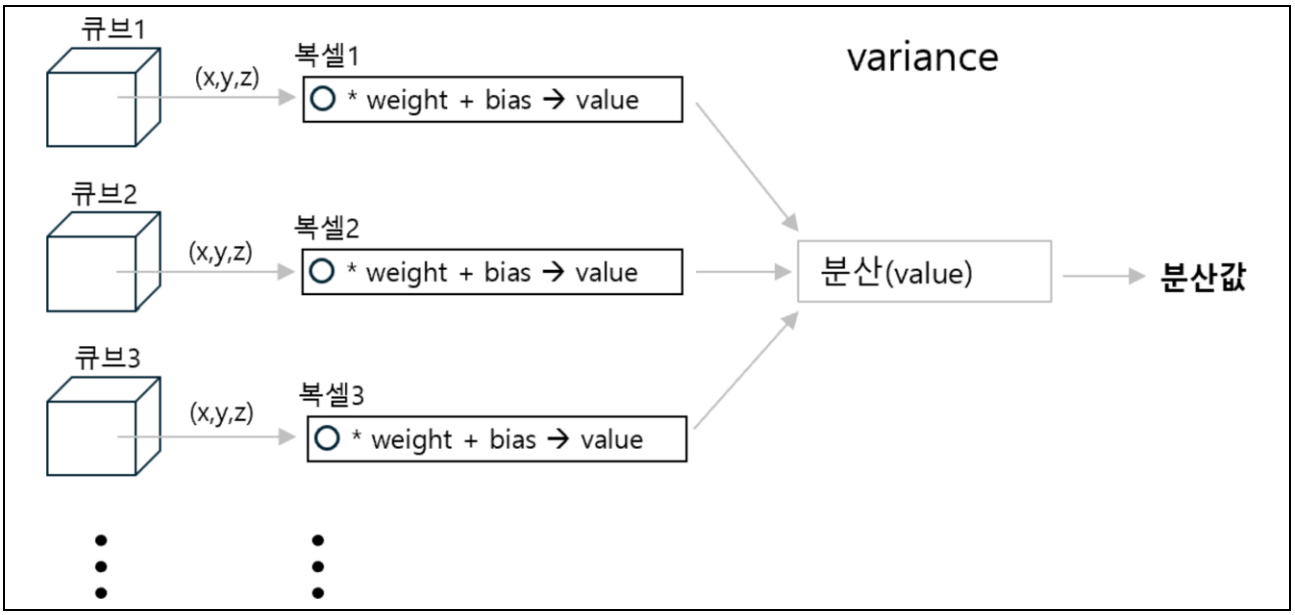
`n - 1` 로 나누는 표본 분산이 아님.

지역에 값이 하나뿐이면 0.

`standardDeviation` 의 제곱과 같음.

variance

메서드	파라미터 / 반환	설명
variance	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 분산값을 구하는 연산.
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases)	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: double
		bit/V2/V3/V4 타입에 대한 지원은 미정.



값을 2 의 거듭제곱으로 축소해 계산한 뒤 복원하므로 중간 계산은 넘치지 않음.

다만 분산은 값의 제곱 규모라 결과 자체가 double 을 넘을 수 있고, 그러면 `ArithmeticException` 을 던짐 (값이 약 1e154 를 넘는 경우).

예제

시험 시나리오 TS-163-LOC(2 겹 분산)의 입력 파일이 있는 폴더에서 실행.

```
TS-163-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-163-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpLocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        Object[] weights = null;
        Object[] biases = null;
        BasicDataCube result = VoxelOpLocal.variance(cubes, weights, biases);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
```

```
    }

}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

Returns:

분산을 담은 새 double 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때, 분산이 double 을 넘을 때

14.7. majority API

```
public static BasicDataCube majority(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases)
```

큐브들의 같은 복셀 위치 값 중 가장 많이 나온 값(다수결값)을 구함.

값을 빈도로 세어 가장 많이 나온 값을 고름.

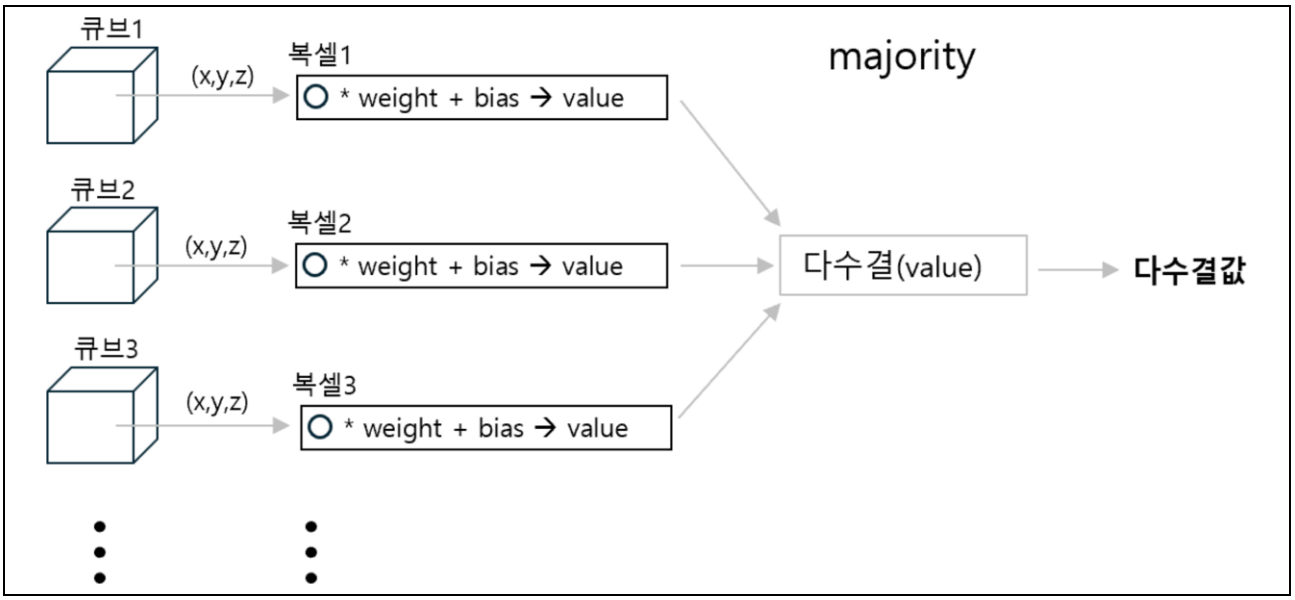
값이 같은지는 `==`로 비교함 (`-0.0` 과 `0.0` 은 같은 값).

- 가장 많이 나온 값이 하나면 그 값
- 가장 많이 나온 값이 여럿이면 그 값들의 평균 — 지역의 값이 모두 다른 경우도 전부 빈도 1 로 같으므로 여기 들어가 지역 전체의 평균이 됨
- 지역에 값이 하나뿐이면 그 값 (큐브가 하나이거나 나머지가 센티넬)

majority

메서드	파라미터 / 반환	설명

majority	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 다수결값을 구하는 연산. 다수결값을 찾을수 없는 경우: 해당지역의 복셀값이 모두 다른 경우 -> 전체 평균값으로 처리 다수결값이 여러 개인 경우 -> 다수결값들의 평균값으로 처리 cubes 중에서 큰쪽의 타입이 byte,short 일때 반환타입은 float 로 처리 cubes 중에서 큰쪽의 타입이 int 일때 반환타입은 double 로 처리
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases)	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: cubes 중에서 큰쪽의 타입을 반환. (평균값으로 처리시: byte,short->float. int->double)
		bit/V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-166-LOC(3 겹 최빈값)의 입력 파일이 있는 폴더에서 실행.

```
TS-166-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-166-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpLocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        Object[] weights = null;
        Object[] biases = null;
        BasicDataCube result = VoxelOpLocal.majority(cubes, weights, biases);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }

}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

Returns:

다수결값을 담은 새 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때

14.8. median API

```
public static BasicDataCube median(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases)
```

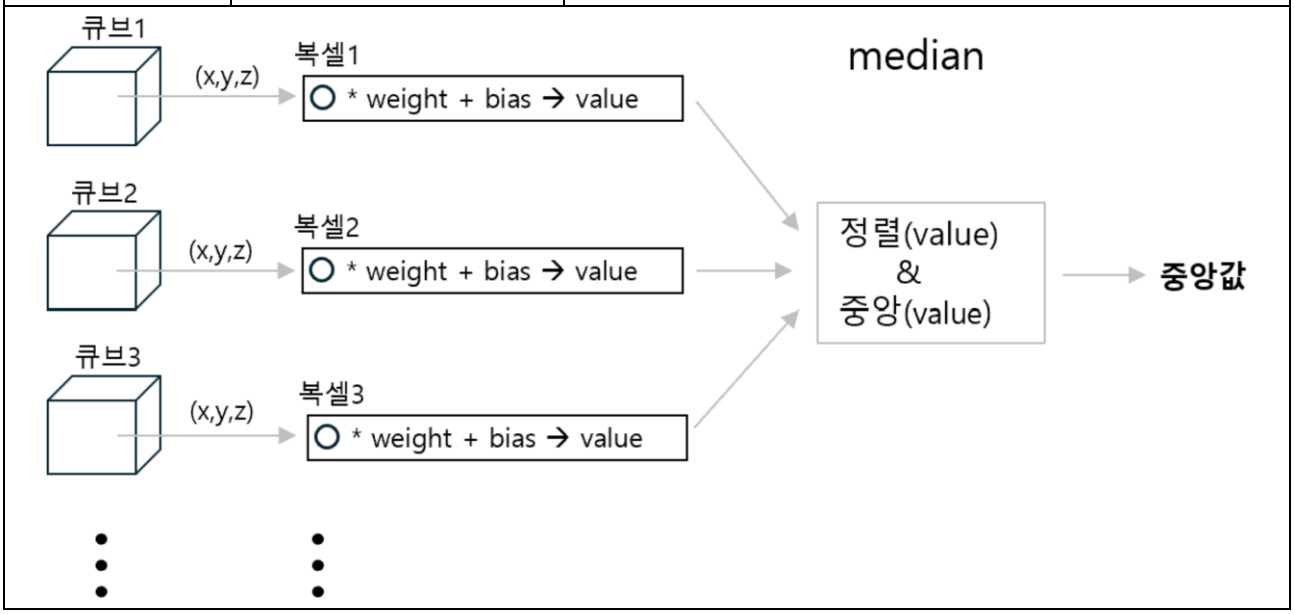
큐브들의 같은 복셀 위치 값을 크기순으로 늘어놓았을 때 가운데 값(중앙값)을 구함.

- 값이 홀수 개면 가운데 값
- 값이 짝수 개면 가운데 두 값의 평균 (두 값이 같으면 그 값)
- 지역에 값이 하나뿐이면 그 값 (큐브가 하나이거나 나머지가 센티넬)

median

메서드	파라미터 / 반환	설명
median	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들을 (voxelValue * weight + bias) 크기순으로 정렬하여 중앙에 위치하는 값을 구하는 연산. 중앙값을 찾을수 없는 경우:

		해당지역의 복셀값 개수가 짝수개인 경우 -> 중앙 두개 값의 평균값으로 처리 cubes 중에서 큰쪽의 타입이 byte,short 일때 반환타입은 float 로 처리 cubes 중에서 큰쪽의 타입이 int 일때 반환타입은 double 로 처리
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases)	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: cubes 중에서 큰쪽의 타입을 반환. (평균값으로 처리시: byte,short->float. int->double)
		bit/V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-170-LOC(2 겹 중앙값)의 입력 파일이 있는 폴더에서 실행.

```
TS-170-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-170-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpLocal;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        Object[] weights = null;
        Object[] biases = null;
        BasicDataCube result = VoxelOpLocal.median(cubes, weights, biases);
        System.out.println(result);
        result.writeToFile("result.bdc");
        System.out.println("saved: result.bdc");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

Returns:

중앙값을 담은 새 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때

14.9. pow API

```
public static BasicDataCube pow(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases,
    int option)
```

큐브들의 같은 복셀 위치 값을 각각 제곱한 뒤, 그 제곱값들의 평균 또는 다수결값을 구함.

지역의 값은 큐브마다 `(복셀값 * weights[k] + biases[k])²`임.

`option` 이 `LOCAL_OPTION_MEAN` 이면 제곱값들의 평균, `LOCAL_OPTION_MAJORITY` 이면 제곱값들의 다수결값 (`majority` 와 같은 규칙)임.

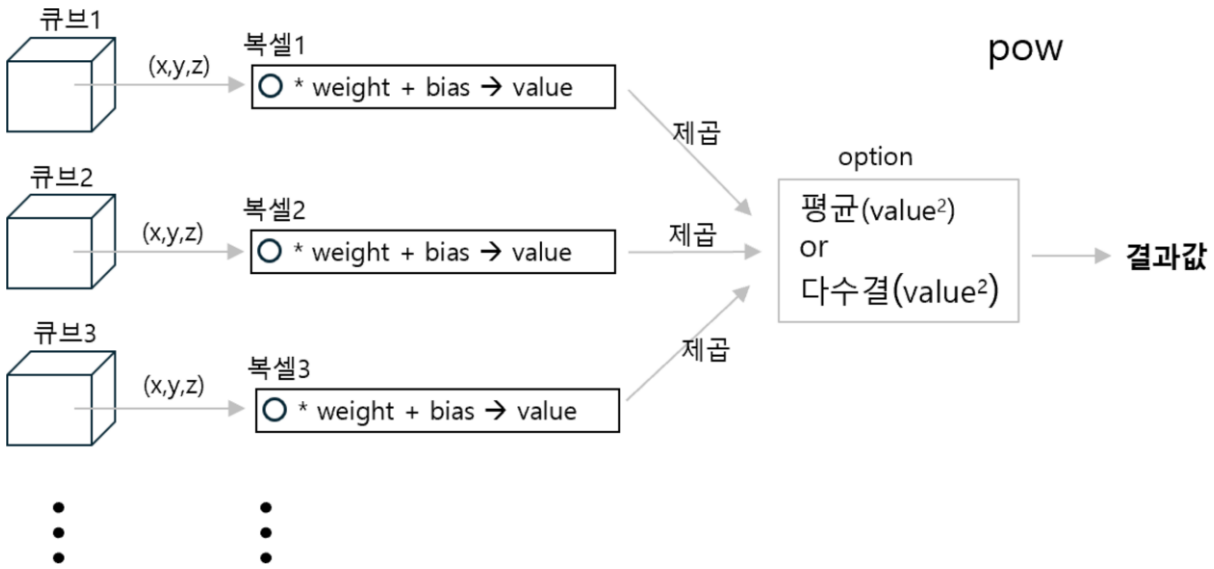
부호가 다른 같은 크기의 값은 제곱이 같으므로 다수결에서 같은 값으로 셈.

지역에 값이 하나뿐이면 그 제곱값.

pow

메서드	파라미터 / 반환	설명
pow	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 제곱을 각각 구한 후, 이들의 평균 또는 다수결값을 구하는 연산.
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	int option)	LOCAL_OPTION_MEAN : 해당지역의 제곱값들의 평균을 구함 LOCAL_OPTION_MAJORITY : 해당지역의 제곱값들의 다수결값을 구함

	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: cubes 중에서 큰쪽의 타입을 반환.
		만약 연산중 오버플로/언더플로 발생시 타입승격하여 반환. (byte->short, short->int, int->double, float->double, double->예외처리) bit/V2/V3/V4 타입에 대한 지원은 미정.



반환 타입은 **큐브 중 가장 큰 타입**에서 시작해 제곱값이나 그 평균이 담기지 않으면 승격함.

제곱은 값이 빠르게 커지므로 byte `12` 만 돼도 `144` 라 short 가 되고, int `46341` 이상이면 double 이 됨.

정수 타입 큐브라도 가중치·바이어스를 적용한 값이 정수가 아니거나 평균이 정수가 아니면 double 임.

double 에서도 제곱이 넘치면 (절대값이 약 1.34e154 이상) `ArithmeticException` 을 던짐.

예제

시험 시나리오 TS-174-LOC(1 겹 pow 평균)의 입력 파일이 있는 폴더에서 실행.

```
TS-174-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-174-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import java.io.FileInputStream;
 import java.util.ArrayList;
 import java.util.List;
 import java.util.zip.ZipEntry;
 import java.util.zip.ZipInputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpLocal;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes = readZip("cubes.zip");
         Object[] weights = {0.5};
         Object[] biases = null;
         BasicDataCube result = VoxelOpLocal.pow(cubes, weights, biases,
VoxelOpLocal.LOCAL_OPTION_MEAN);
         System.out.println(result);
         result.writeToFile("result.bdc");
```

```
        System.out.println("saved: result.bdc");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

`option` - `LOCAL_OPTION_MEAN` 또는 `LOCAL_OPTION_MAJORITY`

Returns:

제곱값의 평균 또는 다수결값을 담은 새 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때, `option` 이 둘 중 하나가 아닐 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이나 그 제곱이 double 을 넘을 때

14.10. sqrt API

```
public static BasicDataCube sqrt(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases,
    int option)
```

큐브들의 같은 복셀 위치 값의 제곱근을 각각 구한 뒤, 그 제곱근들의 평균 또는 다수결값을 구함.

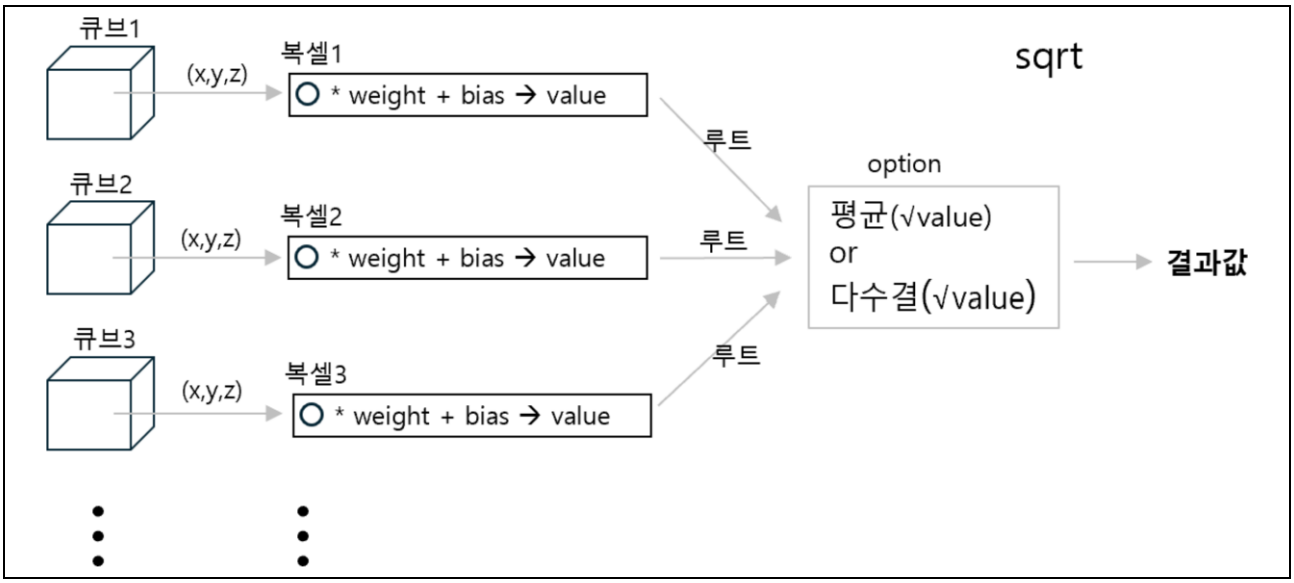
지역의 값은 큐브마다 `√(복셀값 * weights[k] + biases[k])`임.

`option` 이 `LOCAL_OPTION_MEAN` 이면 제곱근들의 평균, `LOCAL_OPTION_MAJORITY` 이면 제곱근들의 다수결값 (`majority` 와 같은 규칙)임.

지역에 값이 하나뿐이면 그 제곱근.

sqrt

메서드	파라미터 / 반환	설명
sqrt	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 제곱근을 각각 구한 후, 이들의 평균 또는 다수결값을 구하는 연산.
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	int option)	LOCAL_OPTION_MEAN : 해당지역의 제곱근값들의 평균을 구함 LOCAL_OPTION_MAJORITY : 해당지역의 제곱근값들의 다수결값을 구함
	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: double
		bit/V2/V3/V4 타입에 대한 지원은 미정.



가중치·바이어스를 적용한 값이 **음수**이면 제곱근이 정의되지 않으므로 `ArithmeticException` 을 던짐.

`'S'` 큐브의 센티넬 자리는 지역에 들어가지 않으므로 센티넬 값이 음수여도 상관없음.

`0` 과 `-0.0` 은 됨.

예제

시험 시나리오 TS-177-LOC(1 겹 sqrt 평균)의 입력 파일이 있는 폴더에서 실행.

```
TS-177-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-177-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import java.io.FileInputStream;
 import java.util.ArrayList;
 import java.util.List;
 import java.util.zip.ZipEntry;
 import java.util.zip.ZipInputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpLocal;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes = readZip("cubes.zip");
         Object[] weights = {4};
         Object[] biases = null;
         BasicDataCube result = VoxelOpLocal.sqrt(cubes, weights, biases,
VoxelOpLocal.LOCAL_OPTION_MEAN);
         System.out.println(result);
         result.writeToFile("result.bdc");
         System.out.println("saved: result.bdc");
     }


     // zip 안의 .bdc 를 저장 순서대로 읽음
     static BasicDataCube[] readZip(String path) throws Exception {
         List<BasicDataCube> list = new ArrayList<>();
         try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
             for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                 list.add(new BasicDataCube(zip.readAllBytes()));
             }
         }
         return list.toArray(new BasicDataCube[0]);
```

```
    }

}
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

`option` - `LOCAL_OPTION_MEAN` 또는 `LOCAL_OPTION_MAJORITY`

Returns:

제곱근의 평균 또는 다수결값을 담은 새 double 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때, `option` 이 둘 중 하나가 아닐 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때, 그 값이 음수일 때

14.11. log API

```
public static BasicDataCube log(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases,
    int option)
```

큐브들의 같은 복셀 위치 값의 자연로그를 각각 구한 뒤, 그 로그값들의 평균 또는 다수결값을 구함.

지역의 값은 큐브마다 `ln(복셀값 * weights[k] + biases[k])`임.

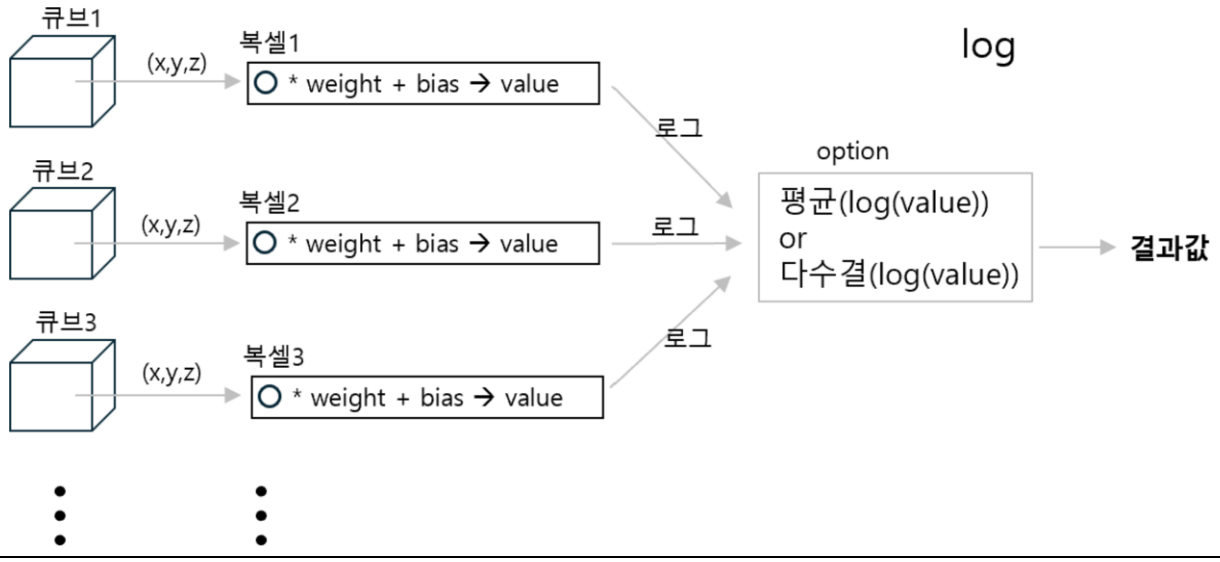
`option` 이 `LOCAL_OPTION_MEAN` 이면 로그값들의 평균, `LOCAL_OPTION_MAJORITY` 이면 로그값들의 다수결값(`majority` 와 같은 규칙)임.

지역에 값이 하나뿐이면 그 로그값.

log

메서드	파라미터 / 반환	설명

log	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 자연로그값을 각각 구한 후, 이들의 평균 또는 다수결값을 구하는 연산.
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	int option)	LOCAL_OPTION_MEAN : 해당지역의 자연로그값들의 평균을 구함 LOCAL_OPTION_MAJORITY : 해당지역의 자연로그값들의 다수결값을 구함
	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: double
		bit/V2/V3/V4 타입에 대한 지원은 미정.



가중치·바이어스를 적용한 값이 **0 이하**이면 자연로그가 정의되지 않으므로 `ArithmeticException` 을 던짐 (0 도 포함 — `ln 0` 은 음의 무한대).

`'S'` 큐브의 센티넬 자리는 지역에 들어가지 않으므로 센티넬 값이 0 이하여도 상관없음.

예제

시험 시나리오 TS-180-LOC(1 겹 log 평균)의 입력 파일이 있는 폴더에서 실행.

```
TS-180-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-180-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import java.io.FileInputStream;
 import java.util.ArrayList;
 import java.util.List;
 import java.util.zip.ZipEntry;
 import java.util.zip.ZipInputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpLocal;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes = readZip("cubes.zip");
         Object[] weights = null;
         Object[] biases = {1};
         BasicDataCube result = VoxelOpLocal.log(cubes, weights, biases,
VoxelOpLocal.LOCAL_OPTION_MEAN);
         System.out.println(result);
         result.writeToFile("result.bdc");
         System.out.println("saved: result.bdc");
     }


     // zip 안의 .bdc 를 저장 순서대로 읽음
     static BasicDataCube[] readZip(String path) throws Exception {
         List<BasicDataCube> list = new ArrayList<>();
         try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
             for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                 list.add(new BasicDataCube(zip.readAllBytes()));
             }
         }
         return list.toArray(new BasicDataCube[0]);
     }


 }
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

`option` - `LOCAL_OPTION_MEAN` 또는 `LOCAL_OPTION_MAJORITY`

Returns:

자연로그의 평균 또는 다수결값을 담은 새 double 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때, `option` 이 둘 중 하나가 아닐 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때, 그 값이 0 이하일 때

14.12. sin API

```
public static BasicDataCube sin(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases,
    int option1,
    int option2)
```

큐브들의 같은 복셀 위치 값의 사인을 각각 구한 뒤, 그 사인값들의 평균 또는 다수결값을 구함.

지역의 값은 큐브마다 `sin(복셀값 * weights[k] + biases[k])`임.

`option1` 이 `TRIG_FUNC_RADIAN` 이면 가중치·바이어스를 적용한 값을 라디안으로, `TRIG_FUNC_DEGREE` 이면 각도로 봄.

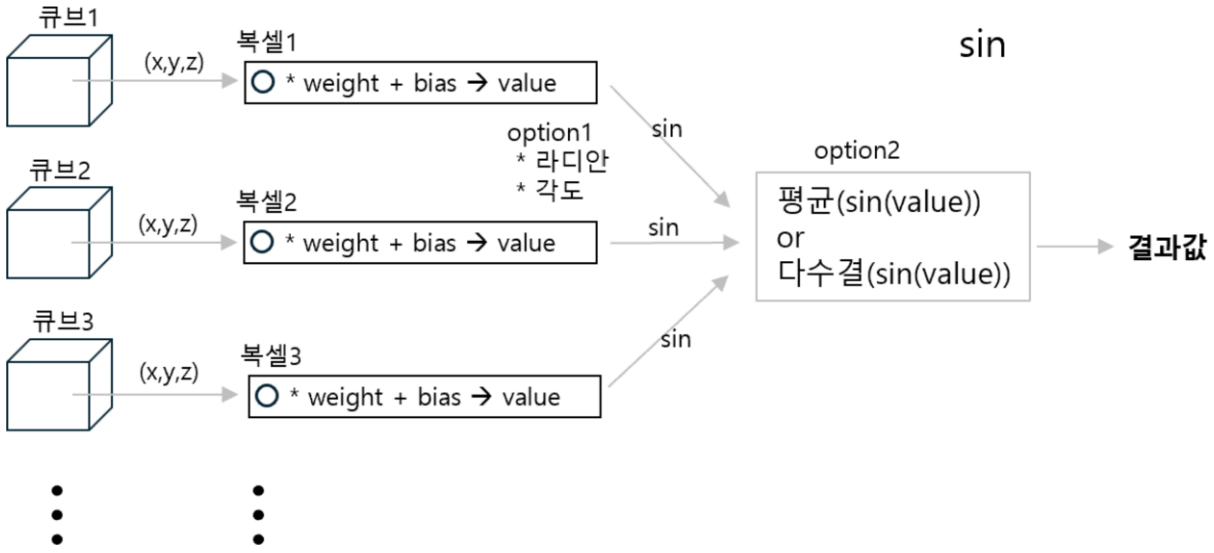
`option2` 가 `LOCAL_OPTION_MEAN` 이면 사인값들의 평균, `LOCAL_OPTION_MAJORITY` 이면 사인값들의 다수결값 (`majority` 와 같은 규칙)임.

지역에 값이 하나뿐이면 그 사인값.

sin

메서드	파라미터 / 반환	설명
sin	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 싸인값을 각각 구한 후, 이들의 평균 또는 다수결값을 구하는 연산.

	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	int option1	TRIG_FUNC_RADIAN : 복셀값(가중치와 바이어스를 적용한)을 라디안값으로 해석. TRIG_FUNC_DEGREE : 복셀값(가중치와 바이어스를 적용한)을 각도값으로 해석.
	int option2)	LOCAL_OPTION_MEAN : 해당지역의 싸인값들의 평균을 구함 LOCAL_OPTION_MAJORITY : 해당지역의 싸인값들의 다수결값을 구함
	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: double
		bit/V2/V3/V4 타입에 대한 지원은 미정.



각도는 산술연산자의 `sin` 과 같은 방법으로 계산함 — 360 도로 접은 뒤 가장 가까운 90 도 배수 기준으로 계산하므로 `180` 이나 `360` 에서 `1.2e-16` 같은 찌꺼기 없이 정확히 `0` 이 나오고, `90` 은 정확히 `1` 임.

라디안은 `Math.sin(double)` 그대로임.

사인은 모든 유한한 값에서 정의되므로 정의역 예외는 없음.

예제

시험 시나리오 TS-181-LOC(2 겹 sin 라디안 평균)의 입력 파일이 있는 폴더에서 실행.

```
TS-181-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-181-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import java.io.FileInputStream;
 import java.util.ArrayList;
 import java.util.List;
 import java.util.zip.ZipEntry;
 import java.util.zip.ZipInputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpLocal;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes = readZip("cubes.zip");
         Object[] weights = null;
         Object[] biases = null;
         BasicDataCube result = VoxelOpLocal.sin(cubes, weights, biases,
VoxelOpLocal.TRIG_FUNC_RADIAN, VoxelOpLocal.LOCAL_OPTION_MEAN);
         System.out.println(result);
         result.writeToFile("result.bdc");
         System.out.println("saved: result.bdc");
     }


     // zip 안의 .bdc 를 저장 순서대로 읽음
     static BasicDataCube[] readZip(String path) throws Exception {
         List<BasicDataCube> list = new ArrayList<>();
         try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
             for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                 list.add(new BasicDataCube(zip.readAllBytes()));
             }
         }
         return list.toArray(new BasicDataCube[0]);
     }


 }
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

`option1` - `TRIG_FUNC_RADIAN` 또는 `TRIG_FUNC_DEGREE`

`option2` - `LOCAL_OPTION_MEAN` 또는 `LOCAL_OPTION_MAJORITY`

Returns:

사인의 평균 또는 다수결값을 담은 새 double 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때, `option1` 이나 `option2` 가 각각 둘 중 하나가 아닐 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때

14.13. cos API

```
public static BasicDataCube cos(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases,
    int option1,
    int option2)
```

큐브들의 같은 복셀 위치 값의 코사인을 각각 구한 뒤, 그 코사인값들의 평균 또는 다수결값을 구함.

지역의 값은 큐브마다 `cos(복셀값 * weights[k] + biases[k])`임.

`option1` 이 `TRIG_FUNC_RADIAN` 이면 가중치·바이어스를 적용한 값을 라디안으로, `TRIG_FUNC_DEGREE` 이면 각도로 봄.

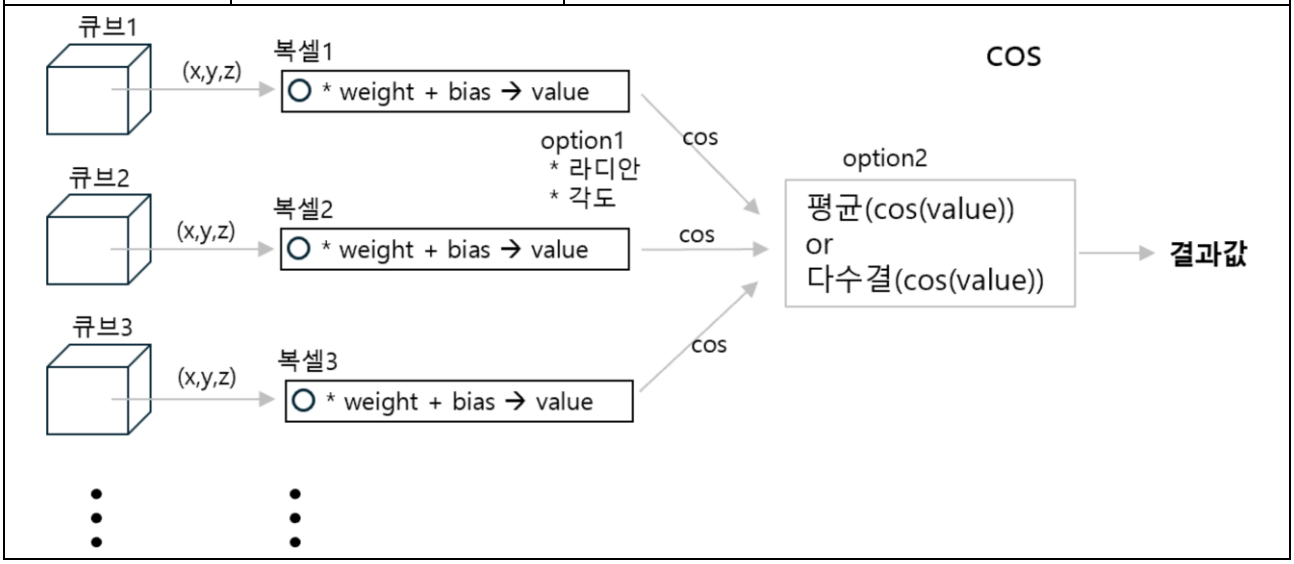
`option2` 가 `LOCAL_OPTION_MEAN` 이면 코사인값들의 평균, `LOCAL_OPTION_MAJORITY` 이면 코사인값들의 다수결값 (`majority` 와 같은 규칙)임.

지역에 값이 하나뿐이면 그 코사인값.

cos

메서드	파라미터 / 반환	설명
cos	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 코싸인값을 각각 구한 후, 이들의 평균 또는 다수결값을 구하는 연산.
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체

		float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	int option1	TRIG_FUNC_RADIAN : 복셀값(가중치와 바이어스를 적용한)을 라디안값으로 해석. TRIG_FUNC_DEGREE : 복셀값(가중치와 바이어스를 적용한)을 각도값으로 해석.
	int option2)	LOCAL_OPTION_MEAN : 해당지역의 코싸인값들의 평균을 구함 LOCAL_OPTION_MAJORITY : 해당지역의 코싸인값들의 다수결값을 구함
	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: double
		bit/V2/V3/V4 타입에 대한 지원은 미정.



각도는 산술연산자의 `cos` 와 같은 방법으로 계산함 — 360 도로 접은 뒤 가장 가까운 90 도 배수 기준으로 계산하므로 `90` 이나 `270` 에서 `6.1e-17` 같은 찌꺼기 없이 정확히 `0` 이 나오고, `180` 은 정확히 `-1` 임.

라디안은 `Math.cos(double)` 그대로임.

코사인은 모든 유한한 값에서 정의되므로 정의역 예외는 없음.

예제

시험 시나리오 TS-186-LOC(1 겹 cos 각도 평균)의 입력 파일이 있는 폴더에서 실행.

```
TS-186-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-186-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import java.io.FileInputStream;
 import java.util.ArrayList;
 import java.util.List;
 import java.util.zip.ZipEntry;
 import java.util.zip.ZipInputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpLocal;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes = readZip("cubes.zip");
         Object[] weights = {2};
         Object[] biases = null;
         BasicDataCube result = VoxelOpLocal.cos(cubes, weights, biases,
VoxelOpLocal.TRIG_FUNC_DEGREE, VoxelOpLocal.LOCAL_OPTION_MEAN);
         System.out.println(result);
         result.writeToFile("result.bdc");
         System.out.println("saved: result.bdc");
     }


     // zip 안의 .bdc 를 저장 순서대로 읽음
     static BasicDataCube[] readZip(String path) throws Exception {
         List<BasicDataCube> list = new ArrayList<>();
         try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
             for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                 list.add(new BasicDataCube(zip.readAllBytes()));
             }
         }
         return list.toArray(new BasicDataCube[0]);
     }


 }
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

`option1` - `TRIG_FUNC_RADIAN` 또는 `TRIG_FUNC_DEGREE`

`option2` - `LOCAL_OPTION_MEAN` 또는 `LOCAL_OPTION_MAJORITY`

Returns:

코사인의 평균 또는 다수결값을 담은 새 double 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때, `option1` 이나 `option2` 가 각각 둘 중 하나가 아닐 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때

14.14. tan API

```
public static BasicDataCube tan(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases,
    int option1,
    int option2)
```

큐브들의 같은 복셀 위치 값의 탄젠트를 각각 구한 뒤, 그 탄젠트값들의 평균 또는 다수결값을 구함.

지역의 값은 큐브마다 `tan(복셀값 * weights[k] + biases[k])`임.

`option1` 이 `TRIG_FUNC_RADIAN` 이면 가중치·바이어스를 적용한 값을 라디안으로, `TRIG_FUNC_DEGREE` 이면 각도로 봄.

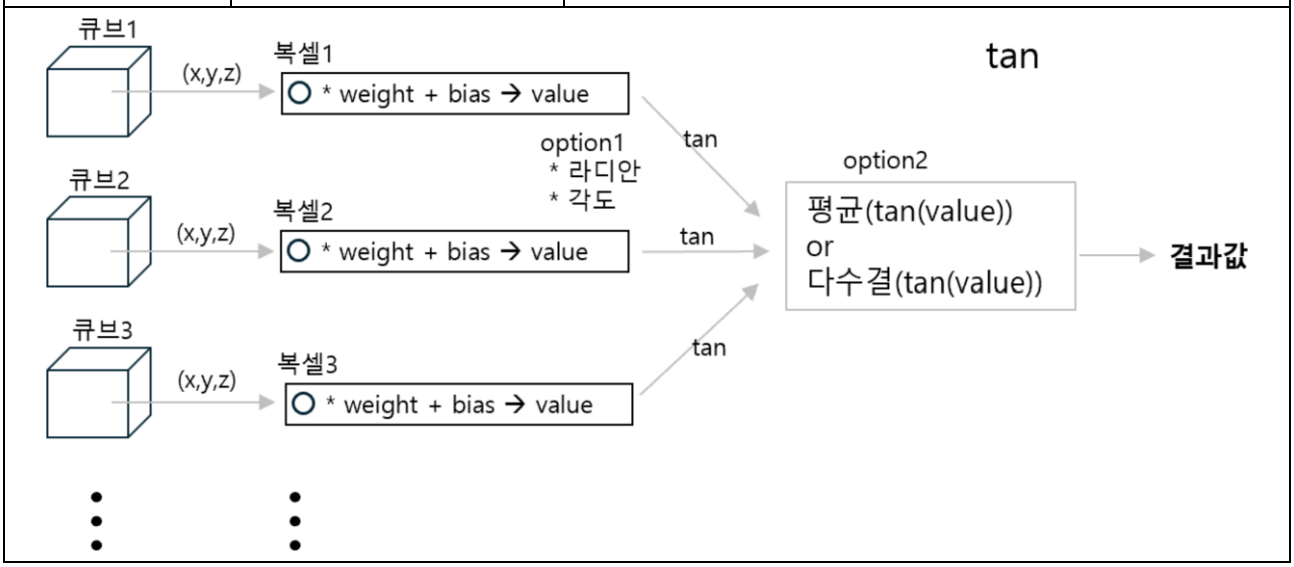
`option2` 가 `LOCAL_OPTION_MEAN` 이면 탄젠트값들의 평균, `LOCAL_OPTION_MAJORITY` 이면 탄젠트값들의 다수결값 (`majority` 와 같은 규칙)임.

지역에 값이 하나뿐이면 그 탄젠트값.

tan

메서드	파라미터 / 반환	설명
tan	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 탄젠트값을 각각 구한 후, 이들의 평균 또는 다수결값을 구하는 연산.
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체

		float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	int option1	TRIG_FUNC_RADIAN : 복셀값(가중치와 바이어스를 적용한)을 라디안값으로 해석. TRIG_FUNC_DEGREE : 복셀값(가중치와 바이어스를 적용한)을 각도값으로 해석.
	int option2)	LOCAL_OPTION_MEAN : 해당지역의 탄젠트값들의 평균을 구함 LOCAL_OPTION_MAJORITY : 해당지역의 탄젠트값들의 다수결값을 구함
	BasicDataCube	연산후 결과를 단일큐브로 반환. 반환타입: double
		bit/V2/V3/V4 타입에 대한 지원은 미정.



각도 모드에서 가중치·바이어스를 적용한 값이 **90 + 180k 도**(90, 270, -90, 450, ...)이면 탄젠트가 정의되지 않으므로 `ArithmeticException` 을 던짐.

라디안 모드에서는 π/2 가 double 로 정확히 표현되지 않아 그 근처 값은 예외 없이 아주 큰 값 (약 `1.6e16`)이 나옴.

각도 계산은 산술연산자와 같은 방법이라 `45` 는 정확히 `1`, `180` 은 정확히 `0` 임.

`'S'` 큐브의 센티넬 자리는 지역에 들어가지 않으므로 센티넬 값이 90 이어도 상관없음.

예제

시험 시나리오 TS-187-LOC(1 겹 tan 라디안 평균)의 입력 파일이 있는 폴더에서 실행.

```
TS-187-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-187-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import java.io.FileInputStream;
 import java.util.ArrayList;
 import java.util.List;
 import java.util.zip.ZipEntry;
 import java.util.zip.ZipInputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpLocal;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes = readZip("cubes.zip");
         Object[] weights = null;
         Object[] biases = null;
         BasicDataCube result = VoxelOpLocal.tan(cubes, weights, biases,
VoxelOpLocal.TRIG_FUNC_RADIAN, VoxelOpLocal.LOCAL_OPTION_MEAN);
         System.out.println(result);
         result.writeToFile("result.bdc");
         System.out.println("saved: result.bdc");
     }


     // zip 안의 .bdc 를 저장 순서대로 읽음
     static BasicDataCube[] readZip(String path) throws Exception {
         List<BasicDataCube> list = new ArrayList<>();
         try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
             for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                 list.add(new BasicDataCube(zip.readAllBytes()));
             }
         }
         return list.toArray(new BasicDataCube[0]);
     }


 }
```

Parameters:

`cubes` - 레스터 데이터를 가지는 큐브 리스트.

비어 있으면 안 되고, 모두 byte, short, int, float, double 중 하나여야 하며, level·`(col, row, floor)`·resolution 이 같아야 함

`weights` - 큐브별 가중치.

`null` 이면 모두 미적용.

배열이면 `cubes` 와 길이가 같아야 하고 원소는 `null`(미적용) 또는 Byte/Short/Integer/Float/Double

`biases` - 큐브별 바이어스.

규칙은 `weights` 와 같음

`option1` - `TRIG_FUNC_RADIAN` 또는 `TRIG_FUNC_DEGREE`

`option2` - `LOCAL_OPTION_MEAN` 또는 `LOCAL_OPTION_MAJORITY`

Returns:

탄젠트의 평균 또는 다수결값을 담은 새 double 큐브

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 지원하지 않는 타입이 섞였을 때, 큐브끼리 level·`(col, row, floor)`·resolution 이 다를 때, `weights`/`biases` 의 길이나 원소가 잘못됐을 때, `option1` 이나 `option2` 가 각각 둘 중 하나가 아닐 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때, 각도 모드에서 그 값이 90 + 180k 도일 때

14.15. countCompare API

```
public static BasicDataCube countCompare(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases,
    Object compareNumber,
    int compareOp)
```

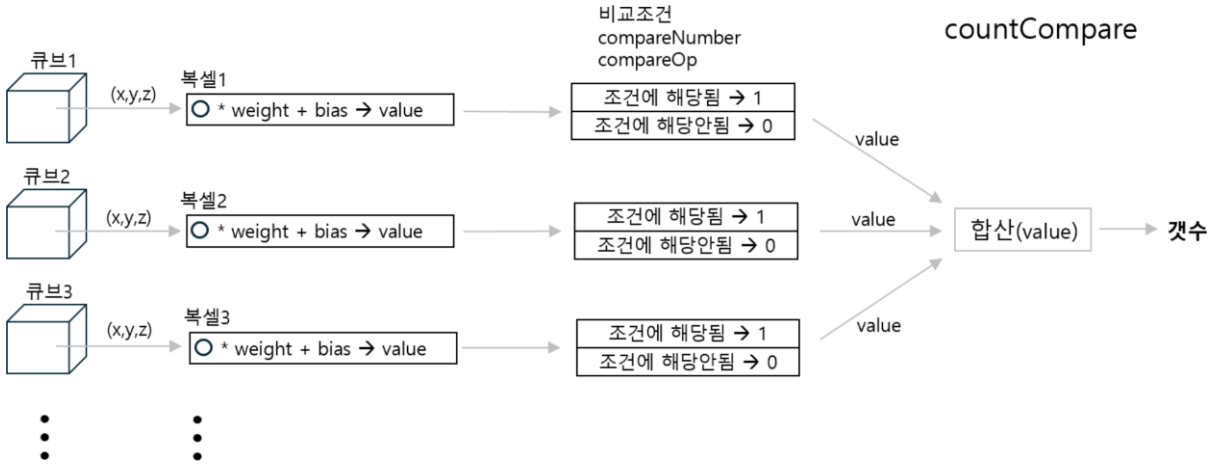
지역 값 중 비교조건을 만족하는 값의 **개수**.

같은 자리에 겹친 큐브들의 복셀 위치마다, 큐브별 가중치·바이어스를 적용한 값들 가운데 `compareNumber` 와의 `compareOp` 비교가 참인 것을 세어 int 큐브에 담아 돌려줌.

countCompare

메서드	파라미터 / 반환	설명
countCompare	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대해서 (voxelValue * weight + bias) 비교조건에 해당하는 복셀이 몇개인지 계산하는 연산.
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체

		biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	Object compareNumber	비교값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체
	int compareOp)	비교 OP LOCAL_OP_EQUAL : (==) 연산값과 비교값이 같은지 비교 LOCAL_OP_NOT_EQUAL : (!=) 연산값과 비교값이 다른지 비교 LOCAL_OP_GREATER_THAN : (>) 연산값이 비교값보다 큰지 비교 LOCAL_OP_LESS_THAN : (<) 연산값이 비교값보다 작은지 비교 LOCAL_OP_GREATER_THAN_EQUAL : (>=) 연산값이 비교값보다 크거나같은지 비교 LOCAL_OP_LESS_THAN_EQUAL : (<=) 연산값이 비교값보다 작거나같은지 비교
	BasicDataCube	갯수 결과를 단일큐브로 반환. 반환타입: int (데이터블록코드는 'V'이며, 해당지역의 모든 복셀이 sentinel 이면 0 값임)
		bit/V2/V3/V4 타입에 대한 지원은 미정.



비교는 지역 값을 왼쪽, `compareNumber` 를 오른쪽에 두고 double 로 함 — `LOCAL_OP_GREATER_THAN` 은 `지역값 > compareNumber` 를 뜻함.

지역 값과 `compareNumber` 를 모두 double 로 바꿔 견주므로, float 큐브의 값을 `Double` 상수와 `==`로 비교하면 float 가 double 로 정확히 옮겨지지 않는 값(예: `0.1f` 와 `0.1`)은 다르게 나옴.

센티넬 자리는 세지 않으므로 모든 큐브가 센티넬인 자리는 0 이고, 그래서 결과는 입력과 무관하게 언제나 `'V'`임.

반환 타입은 언제나 int 임.

예제

시험 시나리오 TS-192-LOC(2 겹 조건 개수)의 입력 파일이 있는 폴더에서 실행.

```
TS-192-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-192-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import java.io.FileInputStream;
 import java.util.ArrayList;
 import java.util.List;
 import java.util.zip.ZipEntry;
 import java.util.zip.ZipInputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpLocal;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes = readZip("cubes.zip");
         Object[] weights = {2, 1};
         Object[] biases = null;
         BasicDataCube result = VoxelOpLocal.countCompare(cubes, weights, biases, 100,
VoxelOpLocal.LOCAL_OP_GREATER_THAN_EQUAL);
         System.out.println(result);
         result.writeToFile("result.bdc");
         System.out.println("saved: result.bdc");
     }


     // zip 안의 .bdc 를 저장 순서대로 읽음
     static BasicDataCube[] readZip(String path) throws Exception {
         List<BasicDataCube> list = new ArrayList<>();
         try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
             for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                 list.add(new BasicDataCube(zip.readAllBytes()));
             }
         }
         return list.toArray(new BasicDataCube[0]);
     }


 }
```

Parameters:

`cubes` - 같은 level, 같은 `(col, row, floor)`, 같은 resolution 의 큐브 리스트.

타입은 달라도 됨

`weights` - 큐브별 가중치.

`null` 이면 모두 1, 원소가 `null` 이면 그 큐브만 1

`biases` - 큐브별 바이어스.

`null` 이면 모두 0, 원소가 `null` 이면 그 큐브만 0

`compareNumber` - 비교값.

`Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나이고 NaN 이 아니어야 함.

무한대는 값으로 비교함

`compareOp` - `LOCAL_OP_EQUAL`, `LOCAL_OP_NOT_EQUAL`, `LOCAL_OP_GREATER_THAN`, `LOCAL_OP_LESS_THAN`, `LOCAL_OP_GREATER_THAN_EQUAL`, `LOCAL_OP_LESS_THAN_EQUAL` 중 하나

Returns:

자리마다 조건을 만족하는 값의 개수를 담은 int `'V'` 큐브 하나

Throws:

`NullPointerException` - `cubes` 나 그 원소가 `null` 일 때, `compareNumber` 가 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 큐브 타입이 bit 나 벡터일 때, 큐브끼리 level·위치·resolution 이 다를 때, `weights`/`biases` 의 길이가 `cubes` 와 다르거나 원소가 Byte/Short/Integer/Float/Double 이 아니거나 NaN·무한대일 때, `compareNumber` 가 Byte/Short/Integer/Float/Double 이 아니거나 NaN 일 때, `compareOp` 가 `LOCAL_OP_*` 여섯 중 하나가 아닐 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때

14.16. classifyCompare API

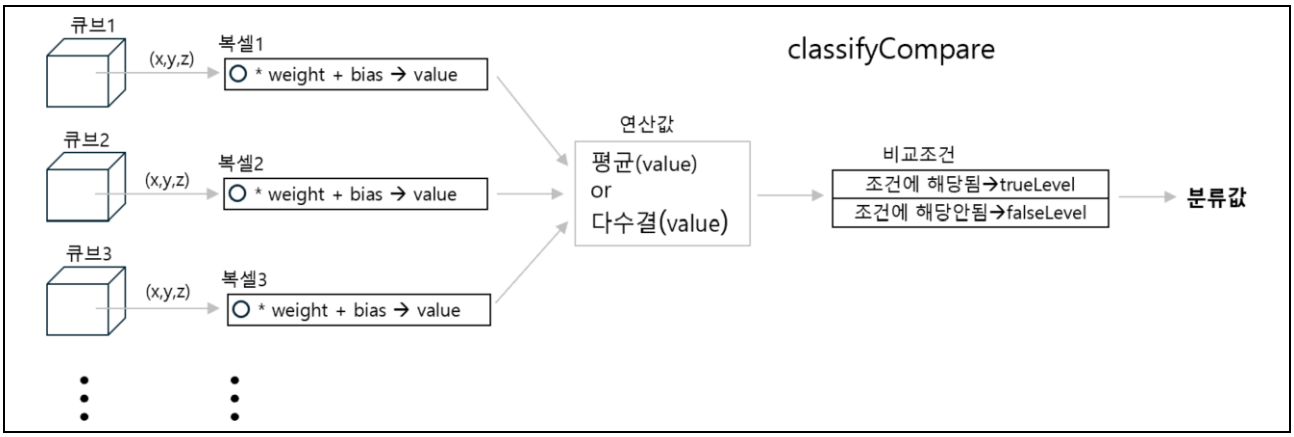
```
public static BasicDataCube classifyCompare(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases,
    int option,
    Object compareNumber,
    int compareOp,
    Object trueLevel,
    Object falseLevel)
```

지역 값의 평균 또는 다수결값이 비교조건에 해당하는지로 **두 값으로 분류**.
같은 자리에 겹친 큐브들의 복셀 위치마다, 큐브별 가중치·바이어스를 적용한 값들을 `option` 에 따라 평균 또는 다수결값으로 줄인 뒤 그 값과 `compareNumber` 의 `compareOp` 비교가 참이면 `trueLevel`, 거짓이면 `falseLevel` 을 담아 돌려줌.

classifyCompare

메서드	파라미터 / 반환	설명
classifyCompare	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 평균 또는 다수결값을 구한후, 그 연산 결과가 비교조건에 해당하는지 해당하지 않는지 두 종류로 분류하는 연산.

	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	int option	LOCAL_OPTION_MEAN : 해당지역의 복셀값들의 평균을 구함 LOCAL_OPTION_MAJORITY : 해당지역의 복셀값들의 다수결값을 구함
	Object compareNumber	비교값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체
	int compareOp	비교 OP LOCAL_OP_EQUAL : (==) 연산값과 비교값이 같은지 비교 LOCAL_OP_NOT_EQUAL : (!=) 연산값과 비교값이 다른지 비교 LOCAL_OP_GREATER_THAN : (>) 연산값이 비교값보다 큰지 비교 LOCAL_OP_LESS_THAN : (<) 연산값이 비교값보다 작은지 비교 LOCAL_OP_GREATER_THAN_EQUAL : (>=) 연산값이 비교값보다 크거나같은지 비교 LOCAL_OP_LESS_THAN_EQUAL : (<=) 연산값이 비교값보다 작거나같은지 비교
	Object trueLevel	연산값이 비교조건에 해당할 때 반환되는 분류값 분류값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체
	Object falseLevel)	연산값이 비교조건에 해당하지 않을 때 반환되는 분류값 분류값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체
	BasicDataCube	분류후 결과를 단일큐브로 반환. 반환타입: trueLevel 과 falseLevel 중에서 큰 타입으로 반환. (두 종류로 분류 : 반환되는 큐브에는 trueLevel 와 falseLevel 두 가지 값을 가짐) (cubes 가 모두 'S' 이고 해당지역의 값이 모두 sentinel 일때 sentinel 값을 가질 수 있음)
		bit/V2/V3/V4 타입에 대한 지원은 미정.



비교하는 것은 지역 값 하나하나가 아니라 **지역을 줄인 값 하나**임 — 예를 들어 지역 {1, 3}의 평균 2 는 `== 2` 에 해당하지만 지역 안에 2 는 없음.

평균과 다수결값은 `mean`· `majority` 와 같은 값(double)이고, 비교는 double 로 함.

반환 타입은 큐브 타입과 무관하게 **`trueLevel` 과 `falseLevel` 중 큰 타입**임.

두 분류값은 같아도 됨.

예제

시험 시나리오 TS-195-LOC(1 겹 평균 비교 분류)의 입력 파일이 있는 폴더에서 실행.

```
TS-195-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-195-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import java.io.FileInputStream;
 import java.util.ArrayList;
 import java.util.List;
 import java.util.zip.ZipEntry;
 import java.util.zip.ZipInputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpLocal;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes = readZip("cubes.zip");
         Object[] weights = null;
         Object[] biases = null;
         BasicDataCube result = VoxelOpLocal.classifyCompare(cubes, weights, biases,
VoxelOpLocal.LOCAL_OPTION_MEAN, 0.0, VoxelOpLocal.LOCAL_OP_LESS_THAN, 0.5, 1.5);
         System.out.println(result);
         result.writeToFile("result.bdc");
         System.out.println("saved: result.bdc");
     }


     // zip 안의 .bdc 를 저장 순서대로 읽음
     static BasicDataCube[] readZip(String path) throws Exception {
         List<BasicDataCube> list = new ArrayList<>();
         try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
             for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                 list.add(new BasicDataCube(zip.readAllBytes()));
             }
         }
         return list.toArray(new BasicDataCube[0]);
     }
```

```
}
```

Parameters:

`cubes` - 같은 level, 같은 `(col, row, floor)`, 같은 resolution 의 큐브 리스트.
타입은 달라도 됨

`weights` - 큐브별 가중치.
`null` 이면 모두 1, 원소가 `null` 이면 그 큐브만 1

`biases` - 큐브별 바이어스.
`null` 이면 모두 0, 원소가 `null` 이면 그 큐브만 0

`option` - `LOCAL_OPTION_MEAN` 또는 `LOCAL_OPTION_MAJORITY`

`compareNumber` - 비교값.
`Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나이고 NaN 이 아니어야 함.
무한대는 값으로 비교함

`compareOp` - `LOCAL_OP_EQUAL`, `LOCAL_OP_NOT_EQUAL`, `LOCAL_OP_GREATER_THAN`, `LOCAL_OP_LESS_THAN`, `LOCAL_OP_GREATER_THAN_EQUAL`, `LOCAL_OP_LESS_THAN_EQUAL` 중 하나

`trueLevel` - 비교가 참인 자리에 담는 분류값.
`Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나이고 유한해야 함

`falseLevel` - 비교가 거짓인 자리에 담는 분류값.
타입 규칙은 `trueLevel` 과 같음

Returns:

자리마다 두 분류값 중 하나를 담은 큐브 하나

Throws:

`NullPointerException` - `cubes` 나 그 원소, `compareNumber`, `trueLevel`, `falseLevel` 이 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 큐브 타입이 bit 나 벡터일 때, 큐브끼리 level·위치·resolution 이 다를 때, `weights`/`biases` 의 길이가 `cubes` 와 다르거나 원소가 Byte/Short/Integer/Float/Double 이 아니거나 NaN·무한대일 때, `option` 이 `LOCAL_OPTION_MEAN`·`LOCAL_OPTION_MAJORITY` 중 하나가 아닐 때, `compareNumber` 가 Byte/Short/Integer/Float/Double 이 아니거나 NaN 일 때, `compareOp` 가 `LOCAL_OP_*` 여섯 중 하나가 아닐 때, 분류값이 Byte/Short/Integer/Float/Double 이 아니거나 NaN·무한대일 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때

14.17. classifyRange API

```
public static BasicDataCube classifyRange(BasicDataCube[] cubes,
    Object[] weights,
    Object[] biases,
    int option,
    Object[][] arrayRangeLevel,
    Object otherLevel)
```

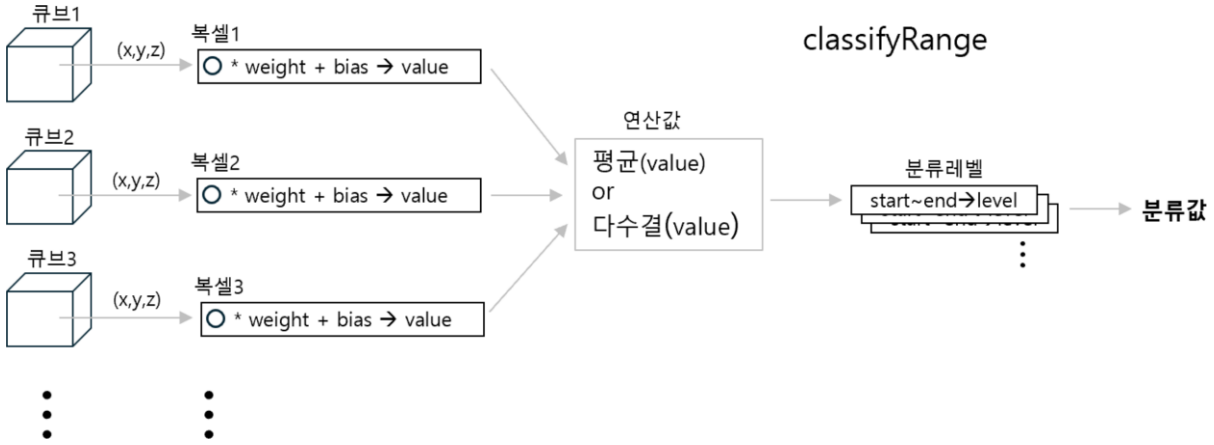
지역 값의 평균 또는 다수결값이 어느 범위에 드는지로 **여러 값으로 분류**.

같은 자리에 겹친 큐브들의 복셀 위치마다, 큐브별 가중치·바이어스를 적용한 값들을 `option` 에 따라 평균 또는 다수결값으로 줄인 뒤 `arrayRangeLevel` 의 `[start, end, level]`을 **들어 있는 순서대로** 보아 처음으로 `start <= 값 < end` 인 범위의 `level` 을, 어느 범위에도 들지 않으면 `otherLevel` 을 담아 돌려줌.

classifyRange

메서드	파라미터 / 반환	설명
classifyRange	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값(각 큐브별 가중치와 바이어스를 적용한)들에 대한 (voxelValue * weight + bias) 평균 또는 다수결값을 구한후, 그 연산 결과가 어떤범위에 해당하는지를 여러종류로 분류하는 연산.
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[] weights	각 큐브들에 대한 가중치값 리스트 (cubes 순서와 일치) 가중치값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 weights 가 null 일 경우 가중치 사용안하고 연산 weights[N]가 null 일 경우 N 번째 큐브는 가중치 사용안하고 연산
	Object[] biases	각 큐브들에 대한 바이어스값 리스트 (cubes 순서와 일치) 바이어스값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 biases 가 null 일 경우 바이어스 사용안하고 연산 biases[N]가 null 일 경우 N 번째 큐브는 바이어스 사용안하고 연산
	int option	LOCAL_OPTION_MEAN : 해당지역의 복셀값들의 평균을 구함 LOCAL_OPTION_MAJORITY : 해당지역의 복셀값들의 다수결값을 구함
	Object[][] arrayRangeLevel	연산값을 분류값으로 변환하기 위한 입력 자료구조. 하나의 범위 및 분류값을 [Object start, Object end, Object

		level]와 같이 배열로 표현하고, 이러한 배열이 여러 개로 구성된 이중 배열. start : 범위의 시작값이며, 이 값은 범위에 포함된다 end : 범위의 끝값이며, 이 값은 범위에 포함되지 않는다 level : 연산값이 start~end 범위일때 반환되는 분류값 start,end,level: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 (참고: 분류작업 수행은 arrayRangeLevel 에 들어있는 순서대로 적용해 나감. 따라서, arrayRangeLevel 자체에 대한 검증(중복,겹침,등)은 하지 않음) (참고: 하나의 범위 및 분류값이 start,end,level 3 개로 구성되었는지만 검증)
	Object otherLevel)	연산값이 어느 한가지의 분류에도 해당하지 않는 경우 이 값을 분류값으로 사용 otherLevel : byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체
	BasicDataCube	분류후 결과를 단일큐브로 반환. 반환타입: arrayRangeLevel 내의 level 중에서 큰 타입으로 반환. (반환되는 큐브에는 arrayRangeLevel 내의 level 값들과 otherLevel 값을 가짐) (cubes 가 모두 'S' 이고 해당지역의 값이 모두 sentinel 일때 sentinel 값을 가질 수 있음)
		bit/V2/V3/V4 타입에 대한 지원은 미정.



범위는 시작값을 포함하고 끝값을 포함하지 않음.

명세대로 `arrayRangeLevel` 자체는 원소가 `[start, end, level]` 세 개짜리인지만 검사하고 겹침·중복·`start >= end` 는 검사하지 않음 — 겹치면 앞의 것이 이기고, `start >= end` 인 범위는 어느 값도 들지 않으며, 빈 배열이면 모든 자리가 `otherLevel` 임.

`start`·`end` 에 무한대를 주면 한쪽이 열린 범위가 됨 (예: `[Double.NEGATIVE_INFINITY, 0, level]`은 0 미만 전부).

비교하는 것은 지역 값 하나하나가 아니라 지역을 줄인 값 하나이며, 평균과 다수결값은 `mean·majority` 와 같은 값(double)이고 범위 판정도 double 로 함.

반환 타입은 큐브 타입과 무관하게 **모든 `level` 과 `otherLevel` 중 큰 타입**임.
분류값은 서로 같아도 됨.

예제

시험 시나리오 TS-196-LOC(1 겹 평균 범위 분류)의 입력 파일이 있는 폴더에서 실행.

```
TS-196-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-196-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import java.io.FileInputStream;
 import java.util.ArrayList;
 import java.util.List;
 import java.util.zip.ZipEntry;
 import java.util.zip.ZipInputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpLocal;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes = readZip("cubes.zip");
         Object[] weights = null;
         Object[] biases = null;
         Object[][] arrayRangeLevel = {
             {0, 40, 1},
             {40, 70, 2},
             {70, 90, 3}
         };
         BasicDataCube result = VoxelOpLocal.classifyRange(cubes, weights, biases,
VoxelOpLocal.LOCAL_OPTION_MEAN, arrayRangeLevel, 9);
         System.out.println(result);
         result.writeToFile("result.bdc");
         System.out.println("saved: result.bdc");
     }


     // zip 안의 .bdc 를 저장 순서대로 읽음
     static BasicDataCube[] readZip(String path) throws Exception {
         List<BasicDataCube> list = new ArrayList<>();
         try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
             for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                 list.add(new BasicDataCube(zip.readAllBytes()));
             }
         }
         return list.toArray(new BasicDataCube[0]);
     }


 }
```

Parameters:

`cubes` - 같은 level, 같은 `(col, row, floor)`, 같은 resolution 의 큐브 리스트.
타입은 달라도 됨

`weights` - 큐브별 가중치.

`null` 이면 모두 1, 원소가 `null` 이면 그 큐브만 1

`biases` - 큐브별 바이어스.

`null` 이면 모두 0, 원소가 `null` 이면 그 큐브만 0

`option` - `LOCAL_OPTION_MEAN` 또는 `LOCAL_OPTION_MAJORITY`

`arrayRangeLevel` - `[start, end, level]` 배열의 배열.

`start·end` 는 `Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나이고 NaN 이 아니어야 함 (무한대 가능),

`level` 은 같은 다섯 타입 중 하나이고 유한해야 함.

순서대로 적용함

`otherLevel` - 어느 범위에도 들지 않는 자리에 담는 분류값.

타입 규칙은 `level` 과 같음

Returns:

자리마다 분류값 하나를 담은 큐브 하나

Throws:

`NullPointerException` - `cubes` 나 그 원소, `arrayRangeLevel` 이나 그 원소, `start·end·level·otherLevel` 이 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 큐브 타입이 bit 나 벡터일 때, 큐브끼리 level·위치·resolution 이 다를 때, `weights/biases` 의 길이가 `cubes` 와 다르거나 원소가 Byte/Short/Integer/Float/Double 이 아니거나 NaN·무한대일 때, `option` 이 `LOCAL_OPTION_MEAN·LOCAL_OPTION_MAJORITY` 중 하나가 아닐 때, `arrayRangeLevel` 의 원소가 세 개짜리가 아닐 때, `start·end` 가 Byte/Short/Integer/Float/Double 이 아니거나 NaN 일 때, `level·otherLevel` 이 Byte/Short/Integer/Float/Double 이 아니거나 NaN·무한대일 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 가중치·바이어스를 적용한 값이 double 을 넘을 때

14.18. classifyScore API

```
public static BasicDataCube classifyScore(BasicDataCube[] cubes,
    Object[][][] arrayRangeScore,
    Object[][] arrayRangeLevel,
    Object otherLevel)
```

큐브마다 값을 **점수**로 바꿔 지역에서 **합산**하고, 그 합이 어느 범위에 드는지로 **여러 값으로 분류**.

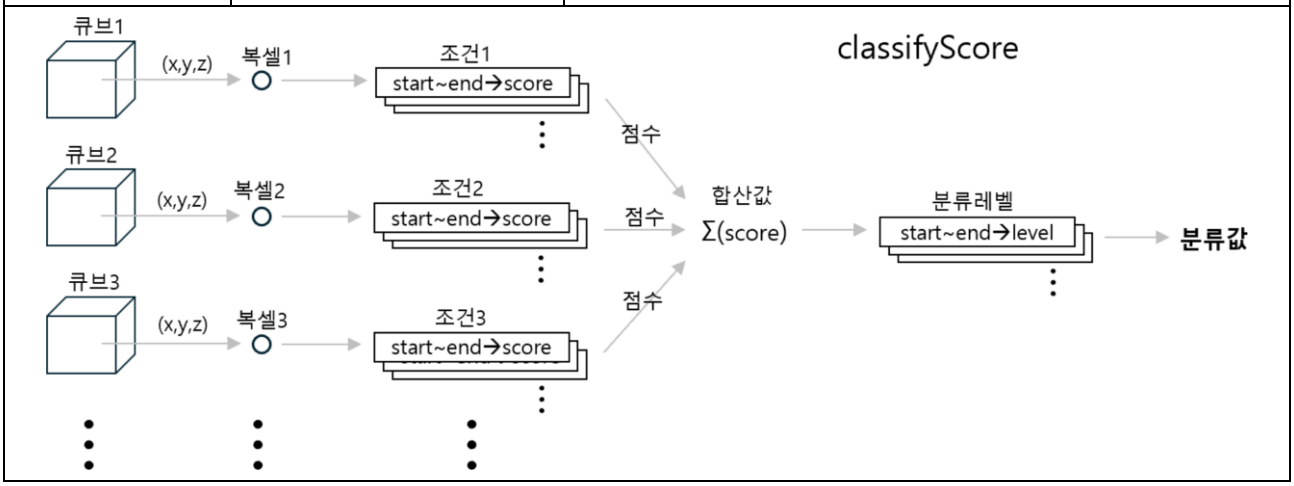
같은 자리에 겹친 큐브들의 복셀 위치마다, `cubes[k]`의 값을 `arrayRangeScore[k]`의 `[start, end, score]`로 점수로 바꾸고(어느 범위에도 안 들면 0) 큐브들의 점수를 모두 더한 뒤, 그 합을 `arrayRangeLevel` 의 `[start, end, level]`로 분류해 담아 돌려줌 (어느 범위에도 들지 않으면 `otherLevel`).

가중치·바이어스는 받지 않음.

classifyScore

메서드	파라미터 / 반환	설명
classifyScore	(	같은 공간크기(level), 같은 공간위치(col,row,floor), 같은 해상도(resolution)를 가지는 레스터 데이터 큐브들에 대해서, 입력으로 들어온 모든 큐브의 임의 복셀위치(x,y,z)를 지역으로 구분하여, 해당 지역의 복셀값들에 대해 각 큐브별로 복셀값의 점수을 구하고, 구해진 점수을 모두 합산한 후, 이 합산값이 어떤 레벨에 해당하는지를 분류하는 연산.
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 타입 이어도 됨) 지원타입: byte,short,int,float,double
	Object[][][] arrayRangeScore	복셀값을 점수로 변환하기 위한 입력 자료구조.(cubes 와 순서일치) 하나의 범위 및 점수를 [Object start, Object end, Object score]와 같이 배열로 표현하고, 이러한 배열이 여러 개로 구성된 이중 배열을 하나의 큐브조건으로 구성하며, 입력된 각 큐브별로 큐브조건들을 삼중 배열로 구성. start : 범위의 시작값이며, 이 값은 범위에 포함된다 end : 범위의 끝값이며, 이 값은 범위에 포함되지 않는다 score : 복셀값이 start~end 범위 일때의 점수 start,end,score: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 복셀값이 어떠한 점수에도 해당하지 않을 때: 점수는 0 으로 함. (참고: 점수작업 수행은 arrayRangeScore 에 들어있는 순서대로 적용해 나감. 따라서, arrayRangeScore 자체에 대한 검증(중복,겹침,등)은 하지 않음) (참고: 하나의 범위 및 점수가 start,end,score 3 개로 구성되었는지만 검증)
	Object[][] arrayRangeLevel	합산값을 분류값으로 변환하기 위한 입력 자료구조. 하나의 범위 및 분류값을 [Object start, Object end, Object level]와 같이 배열로 표현하고, 이러한 배열이 여러 개로 구성된 이중 배열. start : 범위의 시작값이며, 이 값은 범위에 포함된다 end : 범위의 끝값이며, 이 값은 범위에 포함되지 않는다 level : 연산값이 start~end 범위일때 반환되는 분류값 start,end,level: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 (참고: 분류작업 수행은 arrayRangeLevel 에 들어있는 순서대로

		적용해 나감. 따라서, arrayRangeLevel 자체에 대한 검증(중복,겹침,등)은 하지 않음) (참고: 하나의 범위 및 분류값이 start,end,level 3 개로 구성되었는지만 검증)
	Object otherLevel)	합산값이 어느 한가지의 분류에도 해당하지 않는 경우 이 값을 분류값으로 사용 otherLevel : byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체
	BasicDataCube	분류후 결과를 단일큐브로 반환. 반환타입: arrayRangeLevel 내의 level 중에서 큰 타입으로 반환. (반환되는 큐브에는 arrayRangeLevel 내의 level 값들과 otherLevel 값을 가짐) (cubes 가 모두 'S' 이고 해당지역의 값이 모두 sentinel 일때 sentinel 값을 가질 수 있음)
		bit/V2/V3/V4 타입에 대한 지원은 미정.



두 표 모두 시작값을 포함하고 끝값을 포함하지 않으며, 명세대로 원소가 세 개짜리인지만 검사하고 겹침·중복·`start >= end` 는 검사하지 않음 — **들어 있는 순서대로** 처음 맞는 범위를 씀.

빈 점수 표를 준 큐브는 언제나 점수 0 이고, 빈 `arrayRangeLevel` 이면 모든 자리가 `otherLevel` 임.

점수 합은 double 로 계산함.

`'S'` 큐브의 센티넬 자리는 점수 0 으로 더하는 것이 아니라 지역에서 빠지는 것이며 결과는 같음.

반환 타입은 큐브 타입·점수와 무관하게 **모든 `level` 과 `otherLevel` 중 큰 타입**임.

예제

시험 시나리오 TS-201-LOC(1 겹 점수 합산 분류)의 입력 파일이 있는 폴더에서 실행.

```
TS-201-LOC>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-201-LOC>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.util.ArrayList;
import java.util.List;
```

```
 import java.util.zip.ZipEntry;
 import java.util.zip.ZipInputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpLocal;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes = readZip("cubes.zip");
         Object[][][] arrayRangeScore = {
             {
                 {0.0, 4.0, 0.5},
                 {4.0, 30.0, 1.5}
             }
         };
         Object[][] arrayRangeLevel = {
             {0.0, 1.0, 1},
             {1.0, 2.0, 2}
         };
         BasicDataCube result = VoxelOpLocal.classifyScore(cubes, arrayRangeScore, arrayRangeLevel,
0);
         System.out.println(result);
         result.writeToFile("result.bdc");
         System.out.println("saved: result.bdc");
     }


     // zip 안의 .bdc 를 저장 순서대로 읽음
     static BasicDataCube[] readZip(String path) throws Exception {
         List<BasicDataCube> list = new ArrayList<>();
         try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
             for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                 list.add(new BasicDataCube(zip.readAllBytes()));
             }
         }
         return list.toArray(new BasicDataCube[0]);
     }


 }
```

Parameters:

`cubes` - 같은 level, 같은 `(col, row, floor)`, 같은 resolution 의 큐브 리스트.

타입은 달라도 됨

`arrayRangeScore` - 큐브마다 하나씩, `cubes` 와 같은 순서의 점수 표.

`arrayRangeScore[k]`는 `[start, end, score]` 배열의 배열이며 원소는 `null` 이면 안 됨 (빈 표는 됨).

`start`·`end` 는 `Byte`, `Short`, `Integer`, `Float`, `Double` 중 하나이고 NaN 이 아니어야 함 (무한대 가능),

`score` 는 같은 다섯 타입 중 하나이고 유한해야 함

`arrayRangeLevel` - 점수 합을 분류하는 `[start, end, level]` 배열의 배열.

규칙은 점수 표와 같고 `level` 은 유한해야 함

`otherLevel` - 점수 합이 어느 범위에도 들지 않는 자리에 담는 분류값.

타입 규칙은 `level` 과 같음

Returns:

자리마다 분류값 하나를 담은 큐브 하나

Throws:

`NullPointerException` - `cubes` 나 그 원소, `arrayRangeScore` 나 그 원소, `arrayRangeLevel` 이나 그 원소, `start·end·score·level·otherLevel` 이 `null` 일 때

`IllegalArgumentException` - `cubes` 가 비었을 때, 큐브 타입이 bit 나 벡터일 때, 큐브끼리 level·위치·resolution 이 다를 때, `arrayRangeScore` 의 길이가 `cubes` 와 다를 때, 두 표의 원소가 세 개짜리가 아닐 때, `start·end` 가 Byte/Short/Integer/Float/Double 이 아니거나 NaN 일 때, `score·level·otherLevel` 이 Byte/Short/Integer/Float/Double 이 아니거나 NaN·무한대일 때

`ArithmeticException` - `'V'` 큐브의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 점수 합이 double 을 넘을 때

15. VoxelOpDistance [레스터]

클래스 설명

```
public final class VoxelOpDistance
    extends Object
```

[레스터] 거리분석 연산자.

두 객체의 최소경계상자(MBB) 중심 사이의 거리를 구하고, 그 사이를 잇는 선을 그림.

진입점은 정적 메서드이며 인스턴스를 만들 수 없음.

상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

연산 정의

객체 1 의 중심을 (x1, y1, z1), 객체 2 의 중심을 (x2, y2, z2)라 할 때

```
distanceEuclidean   de = √{(x2-x1)² + (y2-y1)² + (z2-z1)²}   좌표축 방향으로 복셀 1 칸이 1.0
distanceManhattan   dm = |x2-x1| + |y2-y1| + |z2-z1|         6 이웃으로 한 번 이동이 1
distanceChebyshev   dc = max(|x2-x1|, |y2-y1|, |z2-z1|)      26 이웃으로 한 번 이동이 1
```

객체 고르기

큐브 리스트 하나가 객체 하나임.

리스트 안의 큐브들은 한 객체를 나눠 담고 있으므로 level, resolution, datatype 이 모두 같아야 하고 `(col, row, floor)`가 서로 달라야 함.

객체 둘은 서로 datatype 이 달라도 되지만 level 과 resolution 은 같아야 함.

어느 복셀이 객체에 속하는지는 범위조건과 `option` 이 정함.

```
minNumber <= 복셀값 < maxNumber            (null 을 준 쪽은 제한 없음)

DISTANCE_PASS_TRUE    범위에 드는 복셀이 객체
DISTANCE_PASS_FALSE   범위에 들지 않는 복셀이 객체
```

bit 큐브와 `Boolean` 상수의 `false`, `true` 는 각각 `0`, `1` 로 바꿔 비교함.

경계값 객체의 타입은 큐브 타입과 달라도 됨 — 둘 다 숫자로 바꿔 비교하므로 byte 큐브에 `Double` 경계를 주거나 bit 큐브에 숫자 경계를 주어도 됨.

경계값의 무한대는 값으로 쓰지만(상한을 `+∞`로 주면 상한이 없는 것과 같음), **센티넬이 아닌 복셀값이 무한대이거나 NaN 이면** `ArithmeticException` 임.

센티넬 자리는 언제나 객체에서 뺌.

범위조건을 보지 않으므로 `DISTANCE_PASS_FALSE` 로 뒤집어도 객체에 들어오지 않음.

MBB 중심

객체에 속한 복셀의 좌표를 큐브 경계 너머까지 이어 본 값으로 MBB 를 잡음 (전역좌표 = `col × 눈금수 + x`, row/floor 도 같음).

축마다 중심은 `(min + max) / 2` 이고, **소수가 나오면 내림해 복셀 하나로 확정함.**

```
x = 1,2,3 을 차지하는 객체      min=1, max=3   중심 2.0 → 복셀 2
x = 1,2,3,4 를 차지하는 객체    min=1, max=4   중심 2.5 → 복셀 2
```

반환

길이 2 인 `Object[]`임.

```
[0]  Double 거리값 (distanceEuclidean)
     또는 Integer 거리값 (distanceManhattan, distanceChebyshev)
[1]  BasicDataCube[] lineCubes — 거리를 시각화하는 bit 큐브 리스트
```

`lineCubes` 는 두 중심을 잇는 경로가 지나는 복셀만 `true` 인 bit 큐브들이며, 경로의 모양은 거리 정의를 따름.

```
distanceEuclidean   두 중심을 잇는 직선.
                    복셀이 차지하는 칸을 조금이라도 통과하면 켜고,
                    모서리나 꼭짓점에 점으로만 닿는 복셀은 통과한 것이 아니므로 켜지 않음

distanceManhattan   축을 하나씩 옮기는 ㄱ자 경로 (X를 다 옮기고 Y, 그다음 Z).
                    한 걸음이 복셀 하나이므로 켜지는 복셀 수는 거리 + 1

distanceChebyshev   26 이웃으로 한 걸음씩 가는 계단 경로.
                    걸음마다 가장 긴 축은 꼭 한 칸 가고, 나머지 축은 직선 위의 점이
                    들어 있는 복셀(좌표를 내림한 자리)로 따라 붙음.
                    한 걸음이 복셀 하나이므로 켜지는 복셀 수는 거리 + 1
```

경로는 **객체 1 의 중심에서 출발해 객체 2 의 중심에서 끝남** — 그래서 `distanceManhattan` 처럼 경로가 방향을 타는 연산은 두 객체를 바꿔 넣으면 모양이 달라짐(거리값은 같음).

경로가 실제로 지나는 `(col, row, floor)` 자리의 큐브만 새로 만들어 담으며, 입력 큐브 자리라도 경로가 지나지 않으면 담지 않음.

순서는 경로가 자리를 처음 밟는 순서임.

큐브는 입력의 level 과 resolution 을 물려받고 데이터 블록 코드는 `'V'`(센티넬 없음)임.

지원 타입

입력은 bit, byte, short, int, float, double 이며 반환하는 `lineCubes` 는 bit 임.

해상도는 `BasicDataCube.Resolution.R64`, `BasicDataCube.Resolution.R128` 둘 다 지원함.

입력 큐브는 변경하지 않음.

사용 예:

```
// 값이 1 이상인 복셀을 객체로 보고 두 객체의 중심 거리를 구함
Object[] r = VoxelOpDistance.distanceEuclidean(
        cubes1, Byte.valueOf((byte) 1), null, VoxelOpDistance.DISTANCE_PASS_TRUE,
        cubes2, Byte.valueOf((byte) 1), null, VoxelOpDistance.DISTANCE_PASS_TRUE);


double distance = (Double) r[0];
BasicDataCube[] lineCubes = (BasicDataCube[]) r[1];


// 맨해튼·체비셰프는 같은 인자에 거리값만 Integer 임 — Double 로 받으면 ClassCastException
int steps = (Integer) VoxelOpDistance.distanceChebyshev(
        cubes1, Byte.valueOf((byte) 1), null, VoxelOpDistance.DISTANCE_PASS_TRUE,
        cubes2, Byte.valueOf((byte) 1), null, VoxelOpDistance.DISTANCE_PASS_TRUE)[0];
```

Field Summary

Fields

Modifier and Type	Field	Description
`static final int`	`DISTANCE_PASS_FALSE`	범위조건을 만족하지 **않는** 복셀을 객체로 봄.
`static final int`	`DISTANCE_PASS_TRUE`	범위조건(`minNumber <= 복셀값 < maxNumber`)을 만족하는 복셀을 객체로 봄.

Method Summary

Modifier and Type	Method	Description
`static Object[]`	`distanceChebyshev(BasicDataCube[] cubes1, Object minNumber1, Object maxNumber1, int option1, BasicDataCube[] cubes2, Object minNumber2, Object maxNumber2, int option2)`	두 객체의 MBB 중심 사이 체비셰프 거리(26 이웃 거리).
`static Object[]`	`distanceEuclidean(BasicDataCube[] cubes1, Object minNumber1, Object maxNumber1, int option1, BasicDataCube[] cubes2, Object minNumber2, Object maxNumber2, int option2)`	두 객체의 MBB 중심 사이 직선거리(피타고라스 거리).
`static Object[]`	`distanceManhattan(BasicDataCube[] cubes1, Object minNumber1, Object maxNumber1, int option1, BasicDataCube[] cubes2, Object minNumber2, Object maxNumber2, int option2)`	두 객체의 MBB 중심 사이 맨해튼 거리(6 이웃 거리).

Field Details

DISTANCE_PASS_TRUE

```
public static final int DISTANCE_PASS_TRUE
```

범위조건(`minNumber <= 복셀값 < maxNumber`)을 만족하는 복셀을 객체로 봄.

See Also:

- Constant Field Values

DISTANCE_PASS_FALSE

```
public static final int DISTANCE_PASS_FALSE
```

범위조건을 만족하지 **않는** 복셀을 객체로 봄. 센티넬 자리는 여기에도 들지 않음.

See Also:

- Constant Field Values

15.1. distanceEuclidean API

```
public static Object[] distanceEuclidean(BasicDataCube[] cubes1,
    Object minNumber1,
    Object maxNumber1,
    int option1,
    BasicDataCube[] cubes2,
    Object minNumber2,
    Object maxNumber2,
    int option2)
```

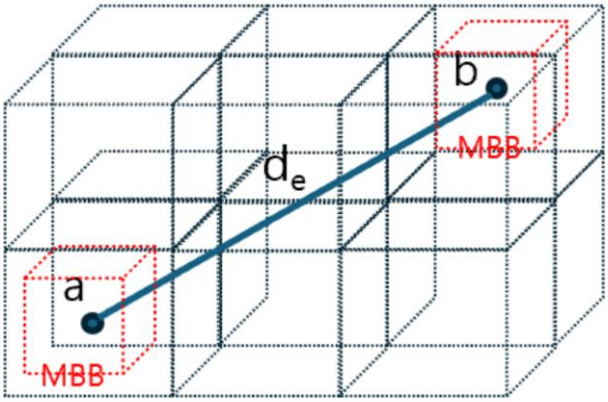
두 객체의 MBB 중심 사이 직선거리(피타고라스 거리).

distanceEuclidean

메서드	파라미터 / 반환	설명
distanceEuclidean	(	객체 1(cubes1)과 객체 2(cubes2)의 최소경계상자(Minimum Bounding Box,MBB) 중심(a 점과 b 점)에서 직선거리(피타고라스거리)를 구하는 연산. de = √{(x2-x1)2+(y2-y1)2+(z2-z1)2} [단위: 좌표축 방향으로 복셀 1 칸에 대한 거리를 1.0 으로 계산] (cubes1, cubes2 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes1	레스터 데이터를 가지는 큐브 리스트. (하나의 객체를 표현하므로 서로 같은 타입 임) 지원타입: bit,byte,short,int,float,double 입력 큐브가 bit 일경우 false=0, true=1 로 수치화 하여 연산.
	Object minNumber1	최소값 1 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber1	최대값 1 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미

	int option1	DISTANCE_PASS_TRUE : 최소값 1~최대값 1 을 만족하는 복셀을 TRUE 로 인식 DISTANCE_PASS_FALSE : 최소값 1~최대값 1 을 만족하는 복셀을 FALSE 로 인식
	BasicDataCube[] cubes2	레스터 데이터를 가지는 큐브 리스트. (하나의 객체를 표현하므로 서로 같은 타입 임) 지원타입: bit,byte,short,int,float,double 입력 큐브가 bit 일경우 false=0, true=1 로 수치화 하여 연산.
	Object minNumber2	최소값 2 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber2	최대값 2 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	int option2)	DISTANCE_PASS_TRUE : 최소값 2~최대값 2 을 만족하는 복셀을 TRUE 로 인식 DISTANCE_PASS_FALSE : 최소값 2~최대값 2 을 만족하는 복셀을 FALSE 로 인식
	[double, BasicDataCube[]]	연산후 결과를 직선거리 및 큐브리스트의 배열로 반환: double distance : 직선거리값 (double 값) BasicDataCube[] lineCubes : 직선거리를 시각화하는 큐브리스트 (bit 큐브리스트)
		V2/V3/V4 타입에 대한 지원은 미정.

distanceEuclidean



반환값: [double $distance_e$, BasicDataCube[] lineCubes]
a와 b의 직선상에서 만나는 복셀을 표현

$$d_e = \sqrt{\{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2\}}$$

예제

시험 시나리오 TS-203-DIS(두 필드의 고점 영역)의 입력 파일이 있는 폴더에서 실행.

```
TS-203-DIS>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
```

```
TS-203-DIS>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import java.io.FileInputStream;
 import java.io.FileOutputStream;
 import java.util.ArrayList;
 import java.util.List;
 import java.util.zip.ZipEntry;
 import java.util.zip.ZipInputStream;
 import java.util.zip.ZipOutputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpDistance;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes1 = readZip("cubes1.zip");
         BasicDataCube[] cubes2 = readZip("cubes2.zip");
         Object[] result = VoxelOpDistance.distanceEuclidean(cubes1, 80, null,
VoxelOpDistance.DISTANCE_PASS_TRUE, cubes2, 70, null, VoxelOpDistance.DISTANCE_PASS_TRUE);
         System.out.println("distance: " + result[0]);
         BasicDataCube[] lineCubes = (BasicDataCube[]) result[1];
         System.out.println("lineCubes cube count: " + lineCubes.length);
         writeZip("lineCubes.zip", lineCubes);
         System.out.println("saved: lineCubes.zip");
     }


     // zip 안의 .bdc 를 저장 순서대로 읽음
     static BasicDataCube[] readZip(String path) throws Exception {
         List<BasicDataCube> list = new ArrayList<>();
         try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
```

```
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }

}
```

Parameters:

`cubes1` - 객체 1 을 담은 큐브 리스트

`minNumber1` - 객체 1 범위조건의 하한(포함).

`null` 이면 제한 없음

`maxNumber1` - 객체 1 범위조건의 상한(제외).

`null` 이면 제한 없음

`option1` - `DISTANCE_PASS_TRUE` 또는 `DISTANCE_PASS_FALSE`

`cubes2` - 객체 2 를 담은 큐브 리스트

`minNumber2` - 객체 2 범위조건의 하한(포함).

`null` 이면 제한 없음

`maxNumber2` - 객체 2 범위조건의 상한(제외).

`null` 이면 제한 없음

`option2` - `DISTANCE_PASS_TRUE` 또는 `DISTANCE_PASS_FALSE`

Returns:

`[0]` 직선거리(Double), `[1]` 경로를 그린 bit 큐브 리스트

Throws:

`IllegalArgumentException` - 큐브 리스트가 `null` 이거나 비었을 때, 리스트에 `null` 원소가 있을 때, 지원하지 않는 datatype 일 때, 한 리스트 안에서 level, resolution, datatype 이 서로 다를 때, 한 리스트 안에서 `(col, row, floor)`가 중복될 때, 두 리스트의 level 또는 resolution 이 다를 때, 상한·하한이 Boolean/Byte/Short/Integer/Float/Double 이 아니거나 NaN 일 때, 둘 다 주었는데 `minNumber >=`

`maxNumber` 일 때, `option` 이 두 상수 중 하나가 아닐 때, 범위조건을 만족하는 복셀이 하나도 없어 객체가 비었을 때

`ArithmeticException` - 센티넬이 없는 큐브(`'V'`)의 복셀값이 NaN 일 때, 복셀값이 무한대일 때

15.2. distanceManhattan API

```
public static Object[] distanceManhattan(BasicDataCube[] cubes1,
    Object minNumber1,
    Object maxNumber1,
    int option1,
    BasicDataCube[] cubes2,
    Object minNumber2,
    Object maxNumber2,
    int option2)
```

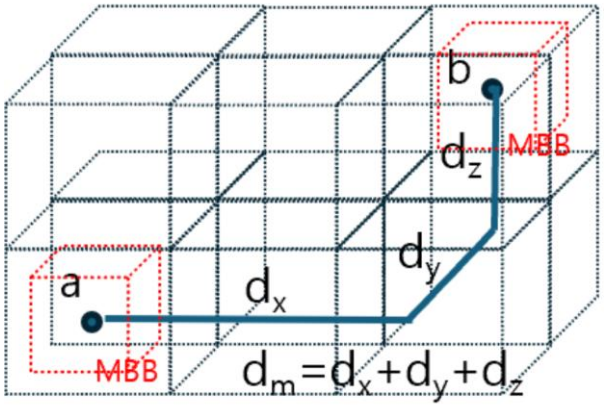
두 객체의 MBB 중심 사이 맨해튼 거리(6 이웃 거리).

distanceManhattan

메서드	파라미터 / 반환	설명
distanceManhattan	(	객체 1(cubes1)과 객체 2(cubes2)의 최소경계상자(Minimum Bounding Box,MBB) 중심(a 점과 b 점)에서 맨하탄거리(6 이웃거리)를 구하는 연산. dm = |x2-x1|+|y2-y1|+|z2-z1| [단위: 6 이웃으로 한번 이동시 1 로 계산] (cubes1, cubes2 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes1	레스터 데이터를 가지는 큐브 리스트. (하나의 객체를 표현하므로 서로 같은 타입 임) 지원타입: bit,byte,short,int,float,double 입력 큐브가 bit 일경우 false=0, true=1 로 수치화 하여 연산.
	Object minNumber1	최소값 1 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber1	최대값 1 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	int option1	DISTANCE_PASS_TRUE : 최소값 1~최대값 1 을 만족하는 복셀을 TRUE 로 인식 DISTANCE_PASS_FALSE : 최소값 1~최대값 1 을 만족하는 복셀을 FALSE 로 인식
	BasicDataCube[] cubes2	레스터 데이터를 가지는 큐브 리스트. (하나의 객체를 표현하므로

		서로 같은 타입 임) 지원타입: bit,byte,short,int,float,double 입력 큐브가 bit 일경우 false=0, true=1 로 수치화 하여 연산.
	Object minNumber2	최소값 2 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber2	최대값 2 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	int option2)	DISTANCE_PASS_TRUE : 최소값 2~최대값 2 을 만족하는 복셀을 TRUE 로 인식 DISTANCE_PASS_FALSE : 최소값 2~최대값 2 을 만족하는 복셀을 FALSE 로 인식
	[int, BasicDataCube[]]	연산후 결과를 맨하탄거리 및 큐브리스트의 배열로 반환: int distance : 맨하탄거리값 (int 값) BasicDataCube[] lineCubes : 맨하탄거리를 시각화하는 큐브리스트 (bit 큐브리스트)
		V2/V3/V4 타입에 대한 지원은 미정.

distanceManhattan



반환값: [int $distance_m$, BasicDataCube[] lineCubes]
맨하탄 거리를 표현
(거리값+1 = 표현복셀수)

$$d_m = |x_2-x_1|+|y_2-y_1|+|z_2-z_1|$$

예제

시험 시나리오 TS-206-DIS(범위 밖 영역과 고점)의 입력 파일이 있는 폴더에서 실행.

```
TS-206-DIS>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-206-DIS>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import java.io.FileInputStream;
 import java.io.FileOutputStream;
 import java.util.ArrayList;
 import java.util.List;
 import java.util.zip.ZipEntry;
 import java.util.zip.ZipInputStream;
 import java.util.zip.ZipOutputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.VoxelOpDistance;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes1 = readZip("cubes1.zip");
         BasicDataCube[] cubes2 = readZip("cubes2.zip");
         Object[] result = VoxelOpDistance.distanceManhattan(cubes1, 10, 85,
VoxelOpDistance.DISTANCE_PASS_FALSE, cubes2, 70, null, VoxelOpDistance.DISTANCE_PASS_TRUE);
         System.out.println("distance: " + result[0]);
         BasicDataCube[] lineCubes = (BasicDataCube[]) result[1];
         System.out.println("lineCubes cube count: " + lineCubes.length);
         writeZip("lineCubes.zip", lineCubes);
         System.out.println("saved: lineCubes.zip");
     }


     // zip 안의 .bdc 를 저장 순서대로 읽음
     static BasicDataCube[] readZip(String path) throws Exception {
         List<BasicDataCube> list = new ArrayList<>();
         try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
             for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                 list.add(new BasicDataCube(zip.readAllBytes()));
             }
         }
         return list.toArray(new BasicDataCube[0]);
     }


     // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
     static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
         try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
             for (int i = 0; i < cubes.length; i++) {
                 BasicDataCube c = cubes[i];
                 zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                         i, c.level(), c.col(), c.row(), c.floor())));
                 zip.write(c.getBdcBinary());
                 zip.closeEntry();
             }
         }
     }


 }
```

Parameters:

`cubes1` - 객체 1 을 담은 큐브 리스트

`minNumber1` - 객체 1 범위조건의 하한(포함).
`null` 이면 제한 없음

`maxNumber1` - 객체 1 범위조건의 상한(제외).
`null` 이면 제한 없음

`option1` - `DISTANCE_PASS_TRUE` 또는 `DISTANCE_PASS_FALSE`

`cubes2` - 객체 2 를 담은 큐브 리스트

`minNumber2` - 객체 2 범위조건의 하한(포함).
`null` 이면 제한 없음

`maxNumber2` - 객체 2 범위조건의 상한(제외).
`null` 이면 제한 없음

`option2` - `DISTANCE_PASS_TRUE` 또는 `DISTANCE_PASS_FALSE`

Returns:

`[0]` 맨해튼 거리(Integer), `[1]` 경로를 그린 bit 큐브 리스트

Throws:

`IllegalArgumentException` - 큐브 리스트가 `null` 이거나 비었을 때, 리스트에 `null` 원소가 있을 때, 지원하지 않는 datatype 일 때, 한 리스트 안에서 level, resolution, datatype 이 서로 다를 때, 한 리스트 안에서 `(col, row, floor)`가 중복될 때, 두 리스트의 level 또는 resolution 이 다를 때, 상한·하한이 Boolean/Byte/Short/Integer/Float/Double 이 아니거나 NaN 일 때, 둘 다 주었는데 `minNumber >= maxNumber` 일 때, `option` 이 두 상수 중 하나가 아닐 때, 범위조건을 만족하는 복셀이 하나도 없어 객체가 비었을 때

`ArithmeticException` - 센티넬이 없는 큐브(`'V'`)의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 거리가 `int` 범위를 넘을 때

15.3. distanceChebyshev API

```
public static Object[] distanceChebyshev(BasicDataCube[] cubes1,
    Object minNumber1,
    Object maxNumber1,
    int option1,
    BasicDataCube[] cubes2,
    Object minNumber2,
    Object maxNumber2,
    int option2)
```

두 객체의 MBB 중심 사이 체비셰프 거리(26 이웃 거리).

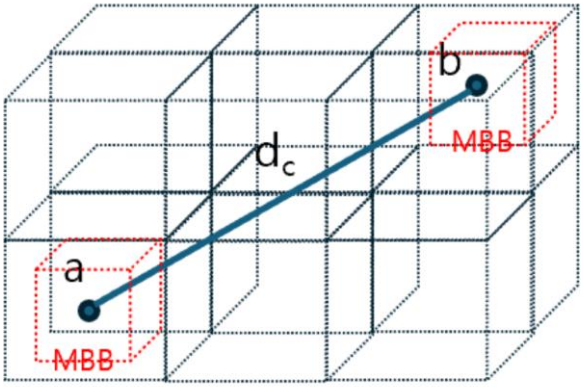
경로는 밟을 자리를 직선 위의 점만으로 정하므로 **두 객체를 바꿔 넣어도 같은 복셀이 켜짐**(`distanceManhattan` 과 다른 점임).

distanceChebyshev

메서드	파라미터 / 반환	설명
distanceChebyshev	(	객체 1(cubes1)과 객체 2(cubes2)의 최소경계상자(Minimum Bounding Box,MBB) 중심(a 점과 b 점)에서 체비셰프거리(26 이웃거리)를 구하는 연산. dc = max(|x2-x1|,|y2-y1|,|z2-z1|) [단위: 26 이웃으로 한번 이동시 1 로 계산] (cubes1, cubes2 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes1	레스터 데이터를 가지는 큐브 리스트. (하나의 객체를 표현하므로 서로 같은 타입 임) 지원타입: bit,byte,short,int,float,double 입력 큐브가 bit 일경우 false=0, true=1 로 수치화 하여 연산.
	Object minNumber1	최소값 1 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber1	최대값 1 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	int option1	DISTANCE_PASS_TRUE : 최소값 1~최대값 1 을 만족하는 복셀을 TRUE 로 인식 DISTANCE_PASS_FALSE : 최소값 1~최대값 1 을 만족하는 복셀을 FALSE 로 인식
	BasicDataCube[] cubes2	레스터 데이터를 가지는 큐브 리스트. (하나의 객체를 표현하므로 서로 같은 타입 임) 지원타입: bit,byte,short,int,float,double 입력 큐브가 bit 일경우 false=0, true=1 로 수치화 하여 연산.
	Object minNumber2	최소값 2 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber2	최대값 2 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 bit: Boolean 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	int option2)	DISTANCE_PASS_TRUE : 최소값 2~최대값 2 을 만족하는 복셀을

		TRUE 로 인식 DISTANCE_PASS_FALSE : 최소값 2~최대값 2 을 만족하는 복셀을 FALSE 로 인식
	[int, BasicDataCube[]]	연산후 결과를 체비셰프거리 및 큐브리스트의 배열로 반환: int distance : 체비셰프거리값 (int 값) BasicDataCube[] lineCubes : 체비셰프거리를 시각화하는 큐브리스트 (bit 큐브리스트)
		V2/V3/V4 타입에 대한 지원은 미정.

distanceChebyshev



반환값: [int $distance_c$,
BasicDataCube[] lineCubes]
체비셰프 거리를 표현
(거리값+1 = 표현복셀수)

$d_c = \max(|x_2-x_1|, |y_2-y_1|, |z_2-z_1|)$

예제

시험 시나리오 TS-210-DIS(나선관과 위층 구)의 입력 파일이 있는 폴더에서 실행.

```
TS-210-DIS>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-210-DIS>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpDistance;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes1 = readZip("cubes1.zip");
        BasicDataCube[] cubes2 = readZip("cubes2.zip");
```

```
        Object[] result = VoxelOpDistance.distanceChebyshev(cubes1, true, null,
VoxelOpDistance.DISTANCE_PASS_TRUE, cubes2, true, null, VoxelOpDistance.DISTANCE_PASS_TRUE);
        System.out.println("distance: " + result[0]);
        BasicDataCube[] lineCubes = (BasicDataCube[]) result[1];
        System.out.println("lineCubes cube count: " + lineCubes.length);
        writeZip("lineCubes.zip", lineCubes);
        System.out.println("saved: lineCubes.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes1` - 객체 1 을 담은 큐브 리스트

`minNumber1` - 객체 1 범위조건의 하한(포함).

`null` 이면 제한 없음

`maxNumber1` - 객체 1 범위조건의 상한(제외).

`null` 이면 제한 없음

`option1` - `DISTANCE_PASS_TRUE` 또는 `DISTANCE_PASS_FALSE`

`cubes2` - 객체 2 를 담은 큐브 리스트

`minNumber2` - 객체 2 범위조건의 하한(포함).

`null` 이면 제한 없음

`maxNumber2` - 객체 2 범위조건의 상한(제외).

`null` 이면 제한 없음

`option2` - `DISTANCE_PASS_TRUE` 또는 `DISTANCE_PASS_FALSE`

Returns:

`[0]` 체비셰프 거리(Integer), `[1]` 경로를 그린 bit 큐브 리스트

Throws:

`IllegalArgumentException` - 큐브 리스트가 `null` 이거나 비었을 때, 리스트에 `null` 원소가 있을 때, 지원하지 않는 datatype 일 때, 한 리스트 안에서 level, resolution, datatype 이 서로 다를 때, 한 리스트 안에서 `(col, row, floor)`가 중복될 때, 두 리스트의 level 또는 resolution 이 다를 때, 상한·하한이 Boolean/Byte/Short/Integer/Float/Double 이 아니거나 NaN 일 때, 둘 다 주었는데 `minNumber >= maxNumber` 일 때, `option` 이 두 상수 중 하나가 아닐 때, 범위조건을 만족하는 복셀이 하나도 없어 객체가 비었을 때

`ArithmeticException` - 센티넬이 없는 큐브(`'V'`)의 복셀값이 NaN 일 때, 복셀값이 무한대일 때, 거리가 `int` 범위를 넘을 때

16. VoxelOpConnectedComponent [레스터]

클래스 설명

```
public final class VoxelOpConnectedComponent
    extends Object
```

[레스터] 연결성분추출 연산자.

이웃으로 이어진 복셀들을 한 덩어리로 묶고 덩어리마다 번호(Label ID)를 매김.

진입점은 정적 메서드이며 인스턴스를 만들 수 없음.

상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한).

대상 고르기

큐브 리스트 하나가 공간 하나임.

리스트 안의 큐브들은 level, resolution, datatype 이 모두 같아야 하고 `(col, row, floor)`가 서로 달라야 함.

어느 복셀을 덩어리로 묶을지는 오버로드가 정함.

```
bit 큐브      true 인 복셀이 대상

레스터 큐브    범위조건을 만족하는 복셀이 대상
              minNumber <= 복셀값 < maxNumber      (null 을 준 쪽은 제한 없음)
              CC_RANGE_TRUE     범위에 드는 복셀이 대상
              CC_RANGE_FALSE    범위에 들지 않는 복셀이 대상
```

센티넬 자리는 언제나 배경임.

`CC_RANGE_FALSE` 로 뒤집어도 대상이 되지 않음.

경계값의 무한대는 값으로 쓰지만, **센티넬이 아닌 float/double 복셀값이 NaN 이면** **`ArithmeticException`** 임 (센티넬이 있는 `'S'` 큐브의 NaN 은 센티넬로 보므로 해당 없음).

이어짐

대상 복셀 둘이 서로 `neighbor` 가 정한 이웃이면 같은 덩어리임.

레스터 오버로드는 여기에 조건을 하나 더 걸 수 있음.

```
CC_ALL_NEIGHBOR          대상 복셀이기만 하면 이어진 것으로 봄
CC_SAME_VALUE_NEIGHBOR   값까지 같아야 이어진 것으로 봄
```

이어짐은 **큐브 경계를 넘어서도 이어짐** — 한 객체가 여러 큐브에 쪼개져 담겨 있어도 리스트 안에서 맞닿아 있으면 한 덩어리로 묶임.

다만 리스트에 없는 자리는 공간 바깥이라 그쪽으로는 이어지지 않음.

번호 매기기

번호는 1 부터 하나씩 올라감.

입력 리스트 순서로 큐브를 훑고, 큐브 안에서는 복셀 인덱스 순서(x → y → z)로 훑다가 아직 번호가 없는 대상 복셀을 처음 만나면 그 덩어리에 다음 번호를 줌.

그래서 같은 입력에는 언제나 같은 번호가 나옴.

반환

입력 자리마다 결과 큐브 하나를, 입력과 같은 순서로 돌려줌 (덩어리가 하나도 없는 자리의 큐브도 0 으로 채워 그대로 담음).

결과 큐브는 int 타입이고 데이터 블록 코드는 `'V'`(센티넬 없음)이며, level·resolution·`(col, row, floor)`는 입력 큐브를 그대로 물려받음.

배경과 센티넬 자리는 모두 0 임.

입력 큐브는 변경하지 않음.

지원 타입

bit 오버로드는 bit 만, 레스터 오버로드는 byte, short, int, float, double 을 받음.

해상도는 `BasicDataCube.Resolution.R64`, `BasicDataCube.Resolution.R128` 둘 다 지원함.

사용 예:

```
// bit – 26 이웃으로 이어진 덩어리마다 번호를 매김
BasicDataCube[] labels = VoxelOpConnectedComponent.connectedComponent(cubes, Neighbor.N26);


int id = labels[0].getInt(x, y, z);   // 0 이면 배경, 1 이상이면 덩어리 번호


// 레스터 – 값이 5 이상 9 미만인 복셀만, 값이 같은 것끼리 묶음
BasicDataCube[] sameValue = VoxelOpConnectedComponent.connectedComponent(
        cubes, Byte.valueOf((byte) 5), Byte.valueOf((byte) 9),
        VoxelOpConnectedComponent.CC_RANGE_TRUE,
        VoxelOpConnectedComponent.CC_SAME_VALUE_NEIGHBOR, Neighbor.N26);
```

Field Summary

Fields

Modifier and Type	Field	Description
`static final int`	`CC_ALL_NEIGHBOR`	이웃이 대상이기만 하면 이어진 것으로 봄.
`static final int`	`CC_RANGE_FALSE`	범위조건을 만족하지 **않는** 복셀을 대상으로 봄.
`static final int`	`CC_RANGE_TRUE`	범위조건(`minNumber <= 복셀값 < maxNumber`)을 만족하는 복셀을 대상으로 봄.
`static final int`	`CC_SAME_VALUE_NEIGHBOR`	이웃이 대상이고 **값까지 같을 때만** 이어진 것으로 봄.

Method Summary

Modifier and Type	Method	Description
`static BasicDataCube[]`	`connectedComponent(BasicDataCube[] cubes, Neighbor neighbor)`	bit 큐브에서 `true` 인 복셀들을 이웃으로 이어 덩어리마다 번호를 매김.
`static BasicDataCube[]`	`connectedComponent(BasicDataCube[] cubes, Object minNumber, Object maxNumber, int rangeOption, int condition, Neighbor neighbor)`	레스터 큐브에서 범위조건을 만족하는 복셀들을 이웃으로 이어 덩어리마다 번호를 매김.

Field Details

CC_RANGE_TRUE

```
public static final int CC_RANGE_TRUE
```

범위조건(`minNumber <= 복셀값 < maxNumber`)을 만족하는 복셀을 대상으로 봄.

See Also:

- Constant Field Values

CC_RANGE_FALSE

```
public static final int CC_RANGE_FALSE
```

범위조건을 만족하지 **않는** 복셀을 대상으로 봄. 센티넬 자리는 여기에도 들지 않음.

See Also:

- Constant Field Values

CC_SAME_VALUE_NEIGHBOR

```
public static final int CC_SAME_VALUE_NEIGHBOR
```

이웃이 대상이고 **값까지 같을 때만** 이어진 것으로 봄.

See Also:

- Constant Field Values

CC_ALL_NEIGHBOR

```
public static final int CC_ALL_NEIGHBOR
```

이웃이 대상이기만 하면 이어진 것으로 봄.

See Also:

- Constant Field Values

16.1. connectedComponent API

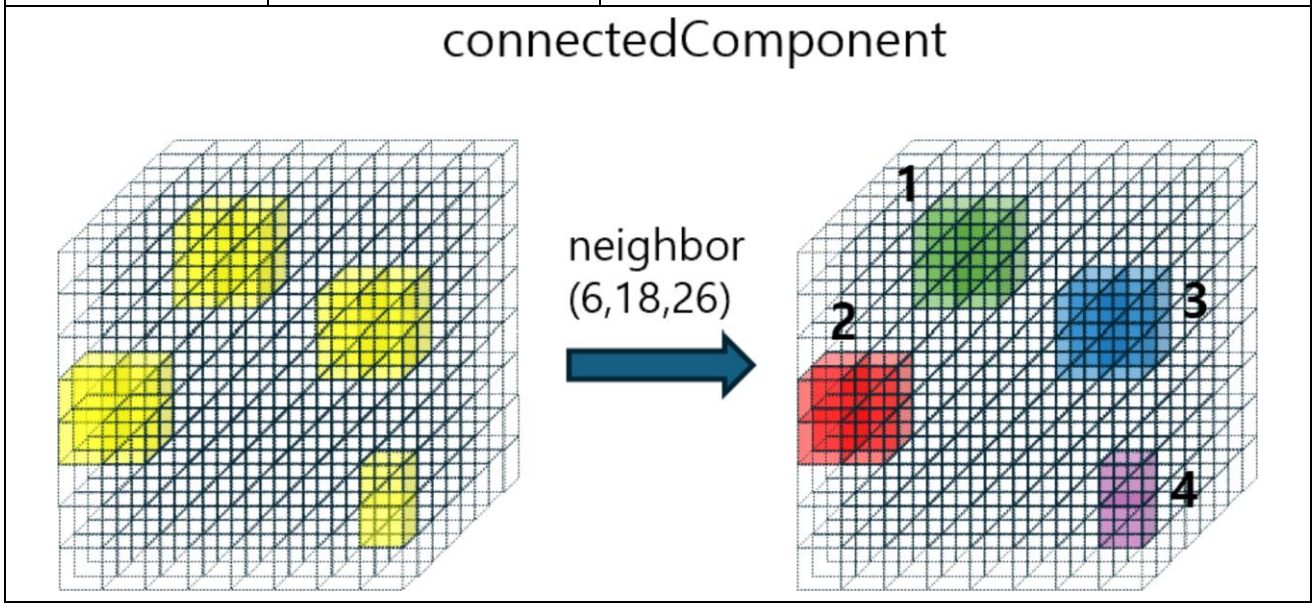
connectedComponent(BasicDataCube[] cubes, Neighbor neighbor)

```
public static BasicDataCube[] connectedComponent(BasicDataCube[] cubes,
    Neighbor neighbor)
```

bit 큐브에서 `true` 인 복셀들을 이웃으로 이어 덩어리마다 번호를 매김.

connectedComponent

메서드	파라미터 / 반환	설명
connectedComponent	(	입력으로 들어온 비트타입 객체(cubes)에 대해, 이웃복셀 선택 기반으로 연결된 복셀들을 하나의 그룹으로 묶어 Label ID (1,2,3,...) 를 부여하는 연산. (cubes 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes	비트 데이터를 가지는 큐브 리스트. (서로 다른 공간위치(col,row,floor)를 가짐) 지원타입: bit
	Neighbor neighbor)	이웃선택 방법: N6, N18, N26 중 하나.
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 반환타입: int (데이터 블록 코드는 'V'이며, 센티널 및 배경(false 값)은 모두 0 값으로 채움)
		V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-211-CCL(구 무리)의 입력 파일이 있는 폴더에서 실행.

```
TS-211-CCL>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-211-CCL>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.Neighbor;
```

```
import com.voxel.operator.VoxelOpConnectedComponent;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        BasicDataCube[] result = VoxelOpConnectedComponent.connectedComponent(cubes, Neighbor.N6);
        System.out.println("result cube count: " + result.length);
        writeZip("result.zip", result);
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - bit 큐브 리스트

`neighbor` - 이웃선택 방법.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

Returns:

입력과 같은 개수·같은 순서의 int 큐브 리스트.

덩어리에 든 복셀은 그 덩어리의 번호(`1` 부터), 배경과 센티넬 자리는 `0`

Throws:

`IllegalArgumentException` - 큐브 리스트가 `null` 이거나 비었을 때, 리스트에 `null` 원소가 있을 때, datatype 이 bit 가 아닐 때, 한 리스트 안에서 level, resolution, datatype 이 서로 다를 때, 한 리스트 안에서 `(col, row, floor)`가 중복될 때, `neighbor` 가 `null` 일 때

connectedComponent(BasicDataCube[] cubes, Object minNumber, Object maxNumber, int rangeOption, int condition, Neighbor neighbor)

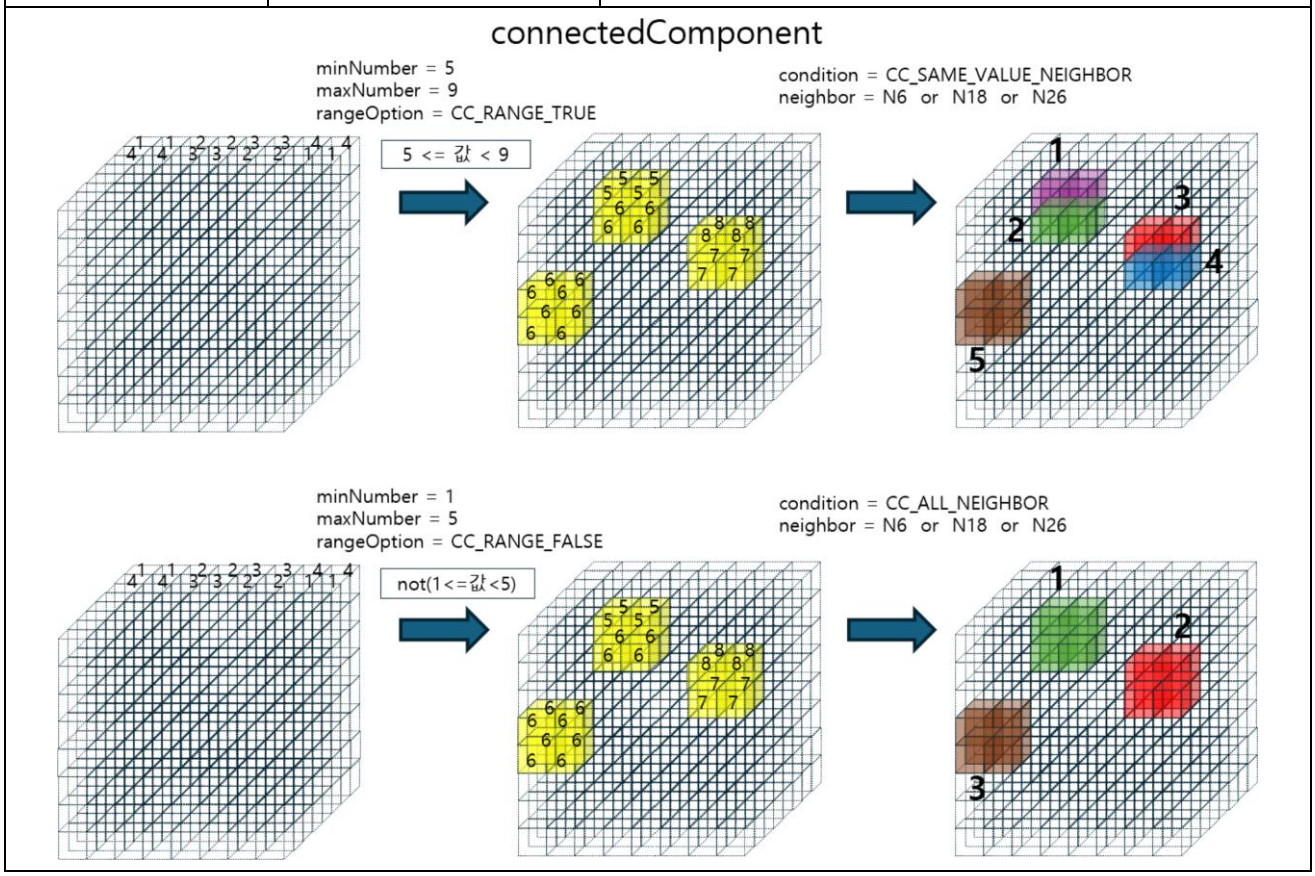
```
public static BasicDataCube[] connectedComponent(BasicDataCube[] cubes,
    Object minNumber,
    Object maxNumber,
    int rangeOption,
    int condition,
    Neighbor neighbor)
```

레스터 큐브에서 범위조건을 만족하는 복셀들을 이웃으로 이어 덩어리마다 번호를 매김.

connectedComponent

메서드	파라미터 / 반환	설명
connectedComponent	(	입력으로 들어온 레스터 객체(cubes)에 대해, 범위조건을 만족하는 복셀들만을 대상으로, 이웃복셀을 선택하여, 동일값-이웃 또는 모든-이웃 기반으로 연결된 복셀들을 하나의 그룹으로 묶어 Label ID (1,2,3,...) 를 부여하는 연산. (cubes 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes	레스터 데이터를 가지는 큐브 리스트. (서로 다른 공간위치(col,row,floor)를 가짐) 지원타입: byte,short,int,float,double
	Object minNumber	최소값 (이값을 포함(minNumber<=voxelValue)하여 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 이 값이 null 이면 최소값 제한이 없음을 의미
	Object maxNumber	최대값 (이값은 포함하지않고(voxelValue<maxNumber) 범위 비교) 상수값: byte: Byte 객체 short: Short 객체 int: Integer 객체 float: Float 객체 double: Double 객체 이 값이 null 이면 최대값 제한이 없음을 의미
	int rangeOption	CC_RANGE_TRUE : 최소값~최대값을 만족하는 복셀을 TRUE 로 인식 CC_RANGE_FALSE : 최소값~최대값을 만족하는 복셀을 FALSE 로 인식 (true 로 인식된 복셀만을 대상으로 연산 함)
	int condition	CC_SAME_VALUE_NEIGHBOR : 찾은 이웃복셀들중 해당복셀의 값과 같을 때만 연결된 복셀로 인정 CC_ALL_NEIGHBOR : 찾은 이웃복셀들 모두를 연결된 복셀로 인정
	Neighbor neighbor)	이웃선택 방법: N6, N18, N26 중 하나.
	BasicDataCube[]	연산후 결과를 큐브 리스트로 반환. 반환타입: int

		(데이터 블록 코드는 'V'이며, 센티널 및 배경(false 로 인식된 복셀)은 모두 0 값으로 채움)
		V2/V3/V4 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-214-CCL(분류 블록 같은 값끼리)의 입력 파일이 있는 폴더에서 실행.

```
TS-214-CCL>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-214-CCL>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
 import java.io.FileInputStream;
 import java.io.FileOutputStream;
 import java.util.ArrayList;
 import java.util.List;
 import java.util.zip.ZipEntry;
 import java.util.zip.ZipInputStream;
 import java.util.zip.ZipOutputStream;


 import com.voxel.operator.BasicDataCube;
 import com.voxel.operator.Neighbor;
 import com.voxel.operator.VoxelOpConnectedComponent;


 public class Example {


     public static void main(String[] args) throws Exception {
         BasicDataCube[] cubes = readZip("cubes.zip");
         BasicDataCube[] result = VoxelOpConnectedComponent.connectedComponent(cubes, null, null,
VoxelOpConnectedComponent.CC_RANGE_TRUE, VoxelOpConnectedComponent.CC_SAME_VALUE_NEIGHBOR,
Neighbor.N6);
         System.out.println("result cube count: " + result.length);
         writeZip("result.zip", result);
```

```
        System.out.println("saved: result.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - byte, short, int, float, double 중 하나인 큐브 리스트

`minNumber` - 범위조건의 하한(포함).

`null` 이면 제한 없음

`maxNumber` - 범위조건의 상한(제외).

`null` 이면 제한 없음

`rangeOption` - `CC_RANGE_TRUE` 또는 `CC_RANGE_FALSE`

`condition` - `CC_SAME_VALUE_NEIGHBOR` 또는 `CC_ALL_NEIGHBOR`

`neighbor` - 이웃선택 방법.

`Neighbor.N6`, `Neighbor.N18`, `Neighbor.N26` 중 하나

Returns:

입력과 같은 개수·같은 순서의 int 큐브 리스트.

덩어리에 든 복셀은 그 덩어리의 번호(`1` 부터), 배경과 센티넬 자리는 `0`

Throws:

`IllegalArgumentException` - 큐브 리스트가 `null` 이거나 비었을 때, 리스트에 `null` 원소가 있을 때, datatype 이 byte, short, int, float, double 이 아닐 때, 한 리스트 안에서 level, resolution, datatype 이 서로 다를 때, 한 리스트 안에서 `(col, row, floor)`가 중복될 때, 상한·하한이

Byte/Short/Integer/Float/Double 이 아니거나 NaN 일 때, 둘 다 주었는데 `minNumber >= maxNumber` 일 때, `rangeOption` 이나 `condition` 이 두 상수 중 하나가 아닐 때, `neighbor` 가 `null` 일 때

`ArithmeticException` - 센티넬이 아닌 float/double 복셀값이 NaN 일 때

17. VoxelOpEarthwork [벡터]

클래스 설명

```
public final class VoxelOpEarthwork
    extends Object
```

[벡터] 토공량분석 연산자. DEM 지형에 부지 폴리곤과 사면 각도를 적용해 절토/성토량을 계산함.

진입점은 정적 메서드 `earthworkGround(BasicDataCube[], int[][], int, int)`(부지)와 `earthworkRoad(BasicDataCube[], int[][], int[][], int, int, int)`(도로) 두 개이며 인스턴스를 만들 수 없음. 상태를 갖지 않으므로 여러 스레드에서 동시에 호출해도 됨 (호출하는 동안 입력 큐브를 다른 곳에서 수정하지 않는 한). 입력 큐브는 변경하지 않음.

계산 방식 (부지)

14. 폴리곤(부지) 내부 기둥들의 기존 지표고를 산술평균해 계획고로 삼음.
15. 폴리곤 밖 기둥은 폴리곤 경계까지의 2D 유클리드 최단거리에 `tan(degree)`를 곱해 계획고로부터 이어지는 사면 높이를 계산하되, 사면이 기존 지표고와 만나면(daylight) 그 지점부터는 지표고를 그대로 씀.
16. 사면 자체의 거리(수평거리가 아닌, 피타고라스 정리로 구한 빗변, 복셀 단위 반올림)가 `maxSlopeLength` 를 넘어서면 더 이상 사면을 잇지 않고 그 경계에서 수직벽으로 끊음.

계산 방식 (도로)

17. BP → IP 들 → EP 꺾은선의 각 IP 모서리를 반지름 `radius` 의 원호(단곡선)로 둥글게 이어 도로 중심선을 만듦. `radius` 가 0 이거나 꺾인 각도가 170° 이상이면 꺾은선 그대로.
18. 중심선을 따라 잰 누적거리(측점)로 BP·IP·EP 의 높이를 선형 보간해 계획고를 정함. IP 의 측점은 곡선 시점 측점 + 접선장.
19. 기둥마다 중심선까지의 수평 최단거리가 폭/2 이내이면 평탄한 도로면, 그 밖은 부지와 같은 사면·수직벽 규칙을 적용. 도로 끝(BP·EP)은 반원 모양이 됨.

표면 (반환 배열의 마지막 원소)

계획이 적용된 지형의 **평탄한 부지(도로면) 표면 + 경사면 표면 + 수직벽 표면** 복셀만 담은 큐브 리스트를 함께 돌려줌. 기둥(x, y)마다 부지(도로면)·사면·daylight·최대길이 초과 중 하나로 나누고 결과 지형의 높이로 다음 복셀을 고름.

- 평탄한 부지(도로면) 표면: 부지 안(도로 폭 안) 기둥마다 계획고 높이의 복셀 하나. 계획고가 기존 지표고와 같아 바뀌지 않은 기둥도 포함
- 경사면 표면: 사면 기둥(daylight 전, 최대길이 이내)마다 계획 높이의 복셀 하나
- 세로 이음: 부지·사면 기둥이 상하좌우 이웃 기둥보다 2 칸 이상 높으면 그 사이의 드러난 옆면 복셀 (경사가 가팔라도 표면에 구멍이 없도록, 성토 쪽 수직벽도 여기에 해당)

- 수직벽 표면 (절토 쪽): 최대길이를 넘어 손대지 않은 기둥이 이웃한 부지·사면 기둥보다 높으면, 그 기둥에서 이웃 높이 위로 드러난 옆면 복셀

daylight(사면이 지표와 만남) 쪽에는 벽이 없음. 표면 복셀이 하나라도 있는 큐브만 입력 순서대로 담으며 개수는 입력과 다를 수 있음. 새 좌표는 만들지 않으므로 큐브 범위를 넘는 표면은 잘림.

범위 밖의 취급

`(col, row, floor)` 조합이 입력 `cubes` 에 없으면 복셀이 모두 0 인 바깥으로 봄 (다른 연산자와 동일한 컨벤션). 이 연산자는 **새 큐브를 만들지 않음** — 성토로 쌓아야 할 높이가 그 (col,row)에 있는 큐브들의 수직 범위를 넘으면 그 이상은 잘림. 성토를 위한 여유 공간이 필요하면 호출 전에 같은 (col,row)에 빈(전부 0) 큐브를 위쪽 floor 로 미리 포함해서 넘겨야 함.

지원 타입

입력과 반환 모두 bit(`BasicDataCube.DataType.BIT`). 다른 타입이 섞이면 `IllegalArgumentException` 을 던짐.

Method Summary

Modifier and Type	Method	Description
`static Object[]`	`earthworkGround(BasicDataCube[] cubes, int[][] polygon, int degree, int maxSlopeLength)`	DEM 지형에 부지 폴리곤과 주변 경사각을 적용해 절토/성토량을 계산함.
`static Object[]`	`earthworkRoad(BasicDataCube[] cubes, int[][] bp_ep, int[][] ips, int width, int degree, int maxSlopeLength)`	DEM 지형에 도로 중심선(BP·IP·EP)과 폭, 주변 경사각을 적용해 절토/성토량을 계산함.

17.1. earthworkGround API

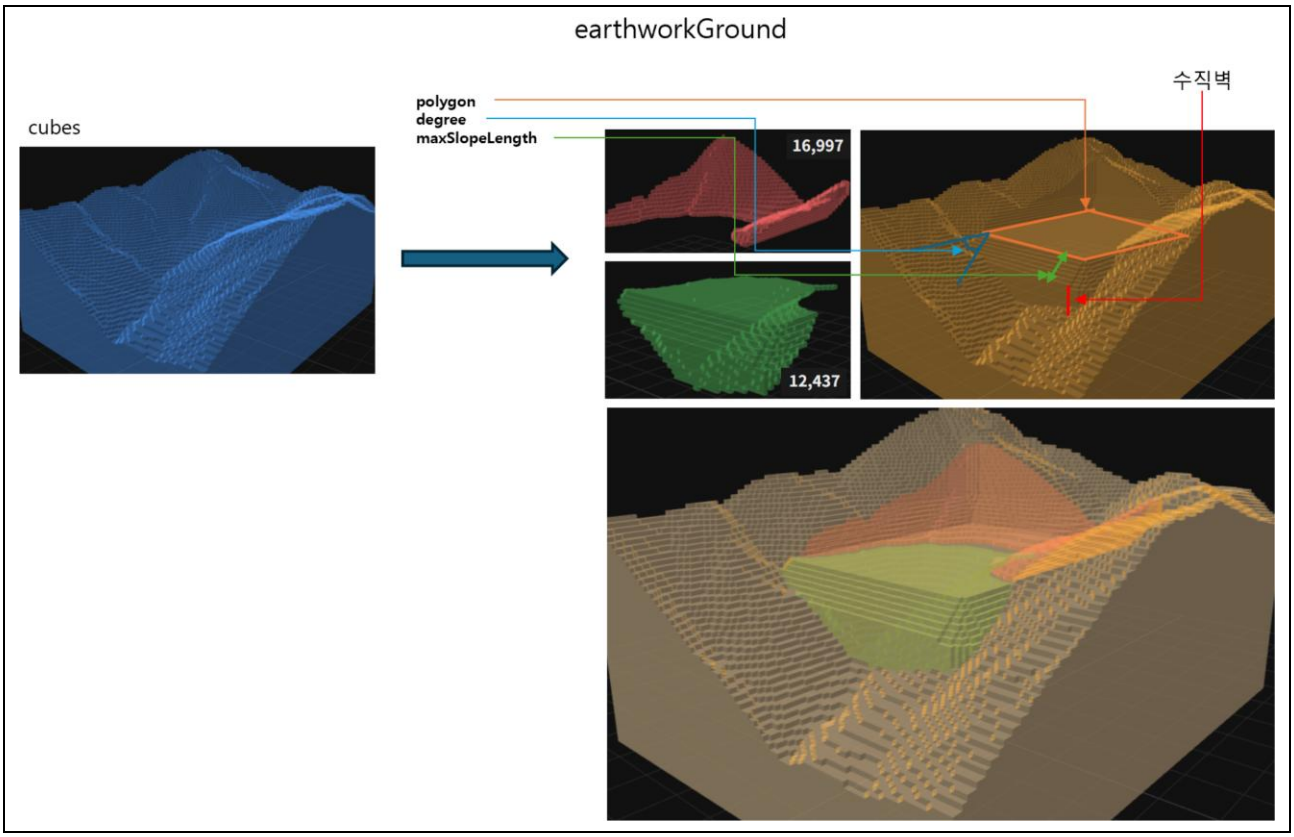
```
public static Object[] earthworkGround(BasicDataCube[] cubes,
    int[][] polygon,
    int degree,
    int maxSlopeLength)
```

DEM 지형에 부지 폴리곤과 주변 경사각을 적용해 절토/성토량을 계산함.

earthworkGround

메서드	파라미터 / 반환	설명
earthworkGround	(	fillDEM 을 사용해서 속이 채워진 DEM 객체(cubes)를 지형형상으로 입력 받고, 높이(floor,z)와 무관한 xy 평면상의 폴리곤(polygon) 좌표를 토공사 부지(ground)로 입력 받고, 토공사 부지 주변의 경사면은 지면으로 부터의 각도(degree)로 입력 받아, 토공량(절토량,성토량)을 계산하는 연산. 참고 : 토공사 부지의 높이(계획고)는 점고법에 의해 평균고로 정해짐(VWorld 참고)

		(cubes 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes	속이 채워진 DEM 데이터를 가지는 큐브 리스트. (서로 다른 공간위치(col,row,floor)를 가짐) 지원타입: bit
	int[][] polygon	높이와 무관한 xy 평면상의 폴리곤 좌표(토공사 부지)를 위한 입력 자료구조. 하나의 좌표를 [int cubesIndex, int x, int y]와 같이 배열로 표현하고, 이러한 좌표를 N 개로 구성하여 N 각형을 구성하는 이중 배열. cubesIndex : cubes 중에서 어떤 큐브인지 가리키는 0 부터 시작하는 인덱스 x : cubesIndex 에 해당하는 큐브 내 에서의 X 복셀좌표 y : cubesIndex 에 해당하는 큐브 내 에서의 Y 복셀좌표 (참고: 입력된 좌표 순서로 N 각형을 구성하므로 꼬이지 않게 입력해야 함) (참고: 하나의 좌표가 cubesIndex,x,y 3 개로 구성되었는지만 검증)
	int degree	토공사 부지 주변의 경사면이 지면으로 부터 이루는 각도 범위 : 0 < degree < 90 (참고 : VWorld 토공량 분석에서 각도를 정수로 입력받음)
	int maxSlopeLength)	토공사 부지 주변의 경사면의 최대 길이. 성토시: 경사면을 이루며 내려가다가 maxSlopeLength 에 도달할 때까지 지면과 만나지 않으면 그 곳에서 부터는 수직벽으로 지면까지 성토. 절토시: 경사면을 이루면 올라가다가 maxSlopeLength 에 도달할 때까지 지면과 만나지 않으면 그 곳에서 부터는 수직벽으로 지면까지 절토.
	[int, double, BasicDataCube[], int, double, BasicDataCube[], BasicDataCube[], BasicDataCube[]]	절토 복셀개수, 절토량[m^3], 절토 큐브리스트(bit), 성토 복셀개수, 성토량[m^3], 성토 큐브리스트(bit), 결과형상 큐브리스트(bit), 부지표면+경사면표면+수직벽표면 형상 큐브리스트(bit) 순서로 반환
		bit 타입만 지원. 나머지 타입에 대한 지원은 미정.



예제

시험 시나리오 TS-217-EWK(평평한 지형, 4 큐브에 걸친 부지)의 입력 파일이 있는 폴더에서 실행.

```
TS-217-EWK>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
```

```
TS-217-EWK>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpEarthwork;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        int[][] polygon = {{0, 40, 40}, {1, 20, 40}, {3, 20, 20}, {2, 40, 20}};
        Object[] result = VoxelOpEarthwork.earthworkGround(cubes, polygon, 45, 20);
        System.out.println("cut voxel count: " + result[0] + ", cut volume[m3]: " + result[1]);
        System.out.println("fill voxel count: " + result[3] + ", fill volume[m3]: " + result[4]);
        writeZip("cut.zip", (BasicDataCube[]) result[2]);
        writeZip("fill.zip", (BasicDataCube[]) result[5]);
        writeZip("result.zip", (BasicDataCube[]) result[6]);
        writeZip("surface.zip", (BasicDataCube[]) result[7]);
        System.out.println("saved: cut.zip, fill.zip, result.zip, surface.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
    static BasicDataCube[] readZip(String path) throws Exception {
```

```
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - `VoxelOpFill.fillDEM(com.voxel.operator.BasicDataCube[])`이 이미 적용된 지형 큐브 리스트. bit 타입이어야 하고, 원소가 하나 이상이어야 하며, 모든 큐브의 level 과 resolution 이 같아야 하고, `(col, row, floor)` 좌표가 서로 달라야 함. 같은 (col,row)에 여러 floor 를 담아 넘길 수 있음

`polygon` - 토공사 부지를 나타내는 xy 평면 폴리곤. `[cubesIndex, x, y]` 형태의 좌표를 3 개 이상 담은 이중배열. `cubesIndex` 는 `cubes` 안에서의 0 부터 시작하는 인덱스, `x/y` 는 그 큐브 안의 로컬 복셀좌표. 좌표당 원소 개수와 `cubesIndex` 범위만 검증하며 자기교차 여부는 검증하지 않음

`degree` - 부지 주변 사면이 지면으로부터 이루는 각도. `0 < degree < 90`

`maxSlopeLength` - 사면 자체의 최대 길이(수평거리가 아닌, 피타고라스 정리로 구한 빗변 거리를 복셀 단위로 반올림한 값). 이 길이 안에서 기존 지표고와 만나지 못하면 그 지점에서 수직벽으로 처리하고 더 이상 사면을 잇지 않음. 양수여야 함

Returns:

길이 8 의 배열 `{ cutVoxelCount:int, cutVolume:double, cutCubes:BasicDataCube[], fillVoxelCount:int, fillVolume:double, fillCubes:BasicDataCube[], resultCubes:BasicDataCube[], surfaceCubes:BasicDataCube[] }`. `cutVolume`/`fillVolume` 은 절토/성토 부피[m³]로, 복셀 개수 × 복셀 부피의 합임. 복셀 부피는 입체격자체계 기준 남북 × 동서 × 높이이며, 동서 길이는 큐브 중심 위도 하나로 계산함(타일 크기 s = 36/2^level 도, 위도 = (row + 0.5)·s − 90, 남북 = s/tick × 111,319.4667m, 동서 = 남북 × cos(위도), 높이 = 4,007,500.8/2^level/tick m). `cutCubes`/`fillCubes`/`resultCubes` 모두 `cubes` 와 길이·순서·좌표가 같은 bit 큐브 리스트임(이 연산자는 새 좌표를 만들지 않음). `resultCubes` 는 절토/성토가 반영된 최종 지형이고, 변화가 없는 좌표는 입력 큐브 객체를 그대로 돌려줌. `surfaceCubes` 는 평탄한 부지 표면 + 경사면 표면 + 수직벽

표면 복셀만 켠 bit 큐브 리스트로, 표면 복셀이 하나라도 있는 큐브만 입력 순서대로 담음(개수는 입력과 다를 수 있음, 표면이 없으면 빈 배열). 표면의 정의는 클래스 문서의 <표면> 참고

Throws:

`IllegalArgumentException` - `cubes` 가 `null` 이거나 비었을 때, `cubes` 에 `null` 원소가 있을 때, bit 가 아닌 큐브가 섞였을 때, 한 리스트 안에서 level 또는 resolution 이 서로 다를 때, `(col, row, floor)`가 중복될 때, `polygon` 이 `null` 이거나 3 개 미만일 때, `polygon` 의 원소가 3 개 값으로 구성되지 않았거나 `cubesIndex`/`x`/`y` 가 범위를 벗어날 때, `degree` 가 0 보다 작거나 같거나 90 이상일 때, `maxSlopeLength` 가 0 보다 작거나 같을 때, 폴리곤 내부에 기존 지형이 하나도 없어 계획고를 정할 수 없을 때

17.2. earthworkRoad API

```
public static Object[] earthworkRoad(BasicDataCube[] cubes,
    int[][] bp_ep,
    int[][] ips,
    int width,
    int degree,
    int maxSlopeLength)
```

DEM 지형에 도로 중심선(BP·IP·EP)과 폭, 주변 경사각을 적용해 절토/성토량을 계산함.

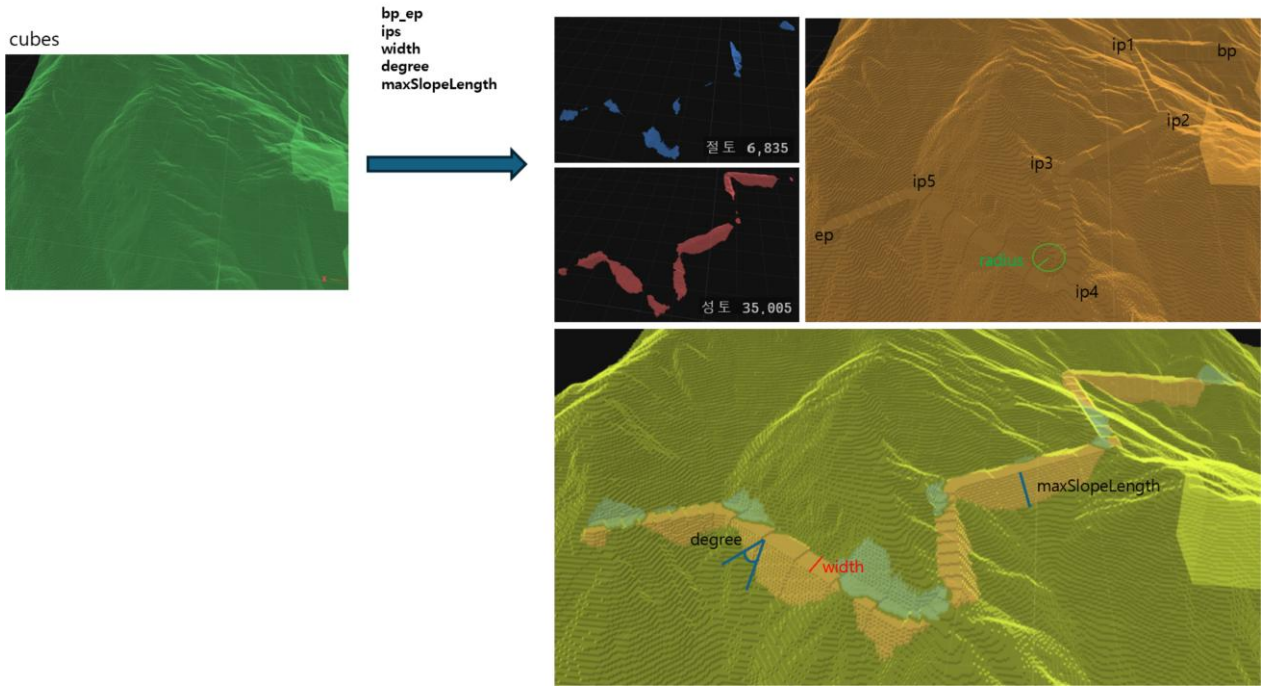
earthworkRoad

메서드	파라미터 / 반환	설명
earthworkRoad	(	fillDEM 을 사용해서 속이 채워진 DEM 객체(cubes)를 지형형상으로 입력 받고, xyz 공간상에서 도로의 시작점 및 끝점을 입력 받고, xyz 공간상에서 도로의 중간중간에 존재하는 커브점들의 정보를 입력 받고, 도로의 폭 및 도로 주변의 경사면 각도(지면기준,degree)를 입력 받아, 토공량(절토량,성토량)을 계산하는 연산. (cubes 는 같은 공간크기(level), 같은 해상도(resolution)를 가짐)
	BasicDataCube[] cubes	속이 채워진 DEM 데이터를 가지는 큐브 리스트. (서로 다른 공간위치(col,row,floor)를 가짐) 지원타입: bit
	int[][] bp_ep	xyz 공간상의 시작점(bp) 및 끝점(ep) 좌표를 위한 입력 자료구조. 하나의 좌표를 [int cubesIndex, int x, int y, int z]와 같이 배열로 표현하고, 이러한 좌표를 2 개로 구성하여 시작점 및 끝점을 구성하는 이중 배열. cubesIndex : cubes 중에서 어떤 큐브인지 가리키는 0 부터 시작하는 인덱스 x : cubesIndex 에 해당하는 큐브 내 에서의 X 복셀좌표 y : cubesIndex 에 해당하는 큐브 내 에서의 Y 복셀좌표

		z : cubesIndex 에 해당하는 큐브를 기준으로 하는 Z 복셀좌표 (음수 또는 해상도값이상 가능) (참고: 입력 순서는 시작점 먼저 입력 함) (참고: 하나의 좌표가 cubesIndex,x,y,z 4 개로 구성되었는지만 검증)
	int[][] ips	xyz 공간상의 커브(ip)점들의 정보를 위한 입력 자료구조. 하나의 정보를 [int cubesIndex, int x, int y, int z, int radius]와 같이 배열로 표현하고, 이러한 정보를 N 개로 구성하여 커브점들을 구성하는 이중 배열. cubesIndex : cubes 중에서 어떤 큐브인지 가리키는 0 부터 시작하는 인덱스 x : cubesIndex 에 해당하는 큐브 내 에서의 X 복셀좌표 y : cubesIndex 에 해당하는 큐브 내 에서의 Y 복셀좌표 z : cubesIndex 에 해당하는 큐브를 기준으로 하는 Z 복셀좌표 (음수 또는 해상도값이상 가능) radius : 실제 커브(곡선)를 계산하기 위한 반지름 정보 (int, 복셀단위) 주의: 임의의 ip 점에서 도로의 꺽인 각도가 대략 170 도~180 도 사이가 될경우 radius=0 으로 입력. radius 값이 0 이면 이 ip 점에서는 커브(곡선)를 계산하지 않고 꺽은선으로 계산함. (참고: 입력 순서는 시작점에서 도로를 따라 가까운 커브점 정보 먼저 입력함) (참고: 하나의 정보가 cubesIndex,x,y,z,radius 5 개로 구성되었는지만 검증)
	int width	도로의 전체 폭 (복셀단위)(양수)
	int degree	도로 주변의 경사면이 지면으로 부터 이루는 각도 범위 : 0 < degree < 90 (참고 : VWorld 토공량 분석에서 각도를 정수로 입력받음)
	int maxSlopeLength)	토공사 부지 주변의 경사면의 최대 길이. 성토시: 경사면을 이루며 내려가다가 maxSlopeLength 에 도달할 때까지 지면과 만나지 않으면 그 곳에서 부터는 수직벽으로 지면까지 성토. 절토시: 경사면을 이루면 올라가다가 maxSlopeLength 에 도달할 때까지 지면과 만나지 않으면 그 곳에서 부터는 수직벽으로 지면까지 절토.
	[int, double, BasicDataCube[], int, double, BasicDataCube[], BasicDataCube[], BasicDataCube[]]	절토 복셀개수, 절토량[m^3], 절토 큐브리스트(bit), 성토 복셀개수, 성토량[m^3], 성토 큐브리스트(bit), 결과형상 큐브리스트(bit), 도로표면+경사면표면+수직벽표면 형상 큐브리스트(bit) 순서로 반환

		bit 타입만 지원. 나머지 타입에 대한 지원은 미정.

earthworkRoad



예제

시험 시나리오 TS-220-EWK(굴곡 지형 대각 직선 도로)의 입력 파일이 있는 폴더에서 실행.

```
TS-220-EWK>javac -encoding UTF-8 -cp ..\..\libs\voxel-operator-<버전>.jar Example.java
TS-220-EWK>java -cp ..\..\libs\voxel-operator-<버전>.jar;. Example
```

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.util.ArrayList;
import java.util.List;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipOutputStream;


import com.voxel.operator.BasicDataCube;
import com.voxel.operator.VoxelOpEarthwork;


public class Example {


    public static void main(String[] args) throws Exception {
        BasicDataCube[] cubes = readZip("cubes.zip");
        int[][] bp_ep = {{0, 10, 10, 44}, {3, 54, 54, 48}};
        int[][] ips = {};
        Object[] result = VoxelOpEarthwork.earthworkRoad(cubes, bp_ep, ips, 10, 45, 30);
        System.out.println("cut voxel count: " + result[0] + ", cut volume[m3]: " + result[1]);
        System.out.println("fill voxel count: " + result[3] + ", fill volume[m3]: " + result[4]);
        writeZip("cut.zip", (BasicDataCube[]) result[2]);
        writeZip("fill.zip", (BasicDataCube[]) result[5]);
        writeZip("result.zip", (BasicDataCube[]) result[6]);
        writeZip("surface.zip", (BasicDataCube[]) result[7]);
        System.out.println("saved: cut.zip, fill.zip, result.zip, surface.zip");
    }


    // zip 안의 .bdc 를 저장 순서대로 읽음
```

```
    static BasicDataCube[] readZip(String path) throws Exception {
        List<BasicDataCube> list = new ArrayList<>();
        try (ZipInputStream zip = new ZipInputStream(new FileInputStream(path))) {
            for (ZipEntry e; (e = zip.getNextEntry()) != null; ) {
                list.add(new BasicDataCube(zip.readAllBytes()));
            }
        }
        return list.toArray(new BasicDataCube[0]);
    }


    // 큐브 리스트를 zip 하나로 저장 (항목 이름: 순번_L 레벨_C 열_R 행_F 층.bdc, 시나리오 zip 과 같음)
    static void writeZip(String path, BasicDataCube[] cubes) throws Exception {
        try (ZipOutputStream zip = new ZipOutputStream(new FileOutputStream(path))) {
            for (int i = 0; i < cubes.length; i++) {
                BasicDataCube c = cubes[i];
                zip.putNextEntry(new ZipEntry(String.format("%03d_L%d_C%d_R%d_F%d.bdc",
                        i, c.level(), c.col(), c.row(), c.floor())));
                zip.write(c.getBdcBinary());
                zip.closeEntry();
            }
        }
    }


}
```

Parameters:

`cubes` - `VoxelOpFill.fillDEM(com.voxel.operator.BasicDataCube[])`이 이미 적용된 지형 큐브 리스트. bit 타입이어야 하고, 원소가 하나 이상이어야 하며, 모든 큐브의 level 과 resolution 이 같아야 하고, `(col, row, floor)` 좌표가 서로 달라야 함

`bp_ep` - 시작점(BP)과 끝점(EP). `[cubesIndex, x, y, z]` 좌표 2 개(BP 먼저)를 담은 이중배열. `x`/`y` 는 그 큐브 안의 로컬 복셀좌표(`0 ~ tick-1`), `z` 는 그 큐브를 기준으로 한 Z 복셀좌표(음수 또는 해상도 이상 가능)

`ips` - 커브점(IP). `[cubesIndex, x, y, z, radius]`를 BP 에서 가까운 순으로 담은 이중배열. 빈 배열이면 BP-EP 직선 도로. `radius` 는 복셀 단위 곡선 반지름으로 0 이면 꺾은선. 꺾인 각도가 170° 이상인 IP 도 꺾은선으로 처리함

`width` - 도로 전체 폭(복셀). 양수여야 함

`degree` - 도로 주변 사면이 지면으로부터 이루는 각도. `0 < degree < 90`

`maxSlopeLength` - 사면 자체의 최대 길이(피타고라스 빗변, 복셀 단위 반올림). 이 길이 안에서 기존 지표고와 만나지 못하면 그 지점에서 수직벽으로 처리함. 양수여야 함

Returns:

길이 8 의 배열 `{ cutVoxelCount:int, cutVolume:double, cutCubes:BasicDataCube[], fillVoxelCount:int, fillVolume:double, fillCubes:BasicDataCube[], resultCubes:BasicDataCube[], surfaceCubes:BasicDataCube[] }`. 규칙은 `earthworkGround(com.voxel.operator.BasicDataCube[], int[][], int, int)`와 같음. `surfaceCubes` 의 평탄한 표면은 도로면(중심선에서 폭/2 이내)임

Throws:

`IllegalArgumentException` - `cubes` 가 올바르지 않을 때(`earthworkGround(com.voxel.operator.BasicDataCube[], int[][], int, int)`와 같은 조건), `bp_ep` 가 `null` 이거나 좌표 2 개가 아닐 때, `ips` 가 `null` 일 때, 좌표 원소 개수가 다르거나(`bp_ep` 4 개, `ips` 5 개) `cubesIndex`/`x`/`y` 가 범위를 벗어날 때, `radius` 가 음수일 때, 연속한 두 제어점의 xy 가 같을 때, 곡선이 들어갈 구간 길이가 모자랄 때, `width·maxSlopeLength` 가 0 이하일 때, `degree` 가 0 이하이거나 90 이상일 때