Introduction to ImgLib2

Tobias Pietzsch
MPI

Motivation

- Image data sets in the life sciences:
  - n-dimensional
  - multi-modal
  - excessive size
- Algorithm implementations are often not re-usable:
  - implemented for fixed dimensionality (often 2d),
  - specific data type,
  - limited image size.
- We want to write code that is independent of image dimensionality, data type, and storage strategy.
- Integration with ImageJ/Fiji (Java, data-structure wrappers)

ImgLib2 Design Goals

Library for *n*-dimensional data representation and manipulation.

Design goals:

- Re-usability, avoid code duplication.
- Decouple algorithm development and data management.
- High-level programming interface.
- High performance.
- Extensibility
  (adding algorithms, pixel types, storage strategies).
- Adaptability
  (to existing data structures).

ImgLib2 Design Goals

Design goals:

- Make accessing a double[] array as complicated as humanly possible.

What is an Image in ImgLib2?

What is an Image in ImgLib2?

$$f : \Omega \rightarrow \mathbb{T} \quad \text{with} \quad \Omega \subset \mathbb{R}^n$$

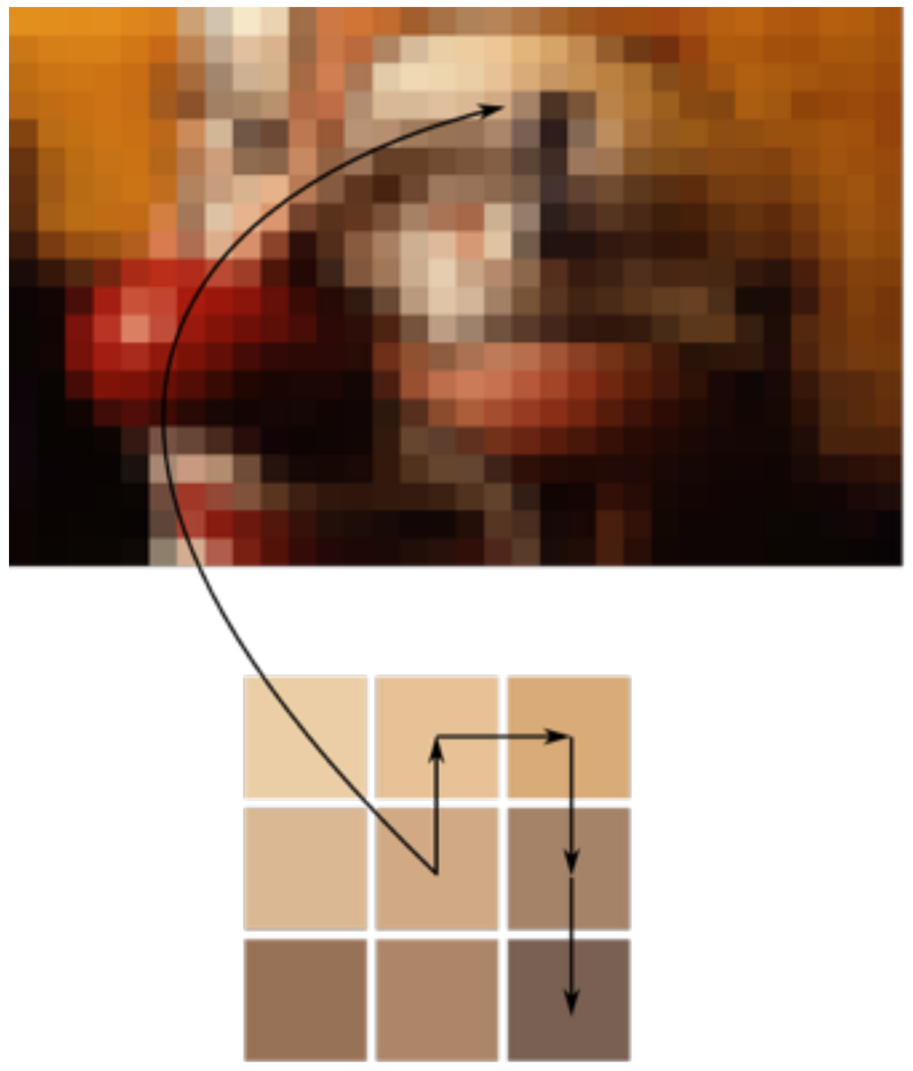
- Arbitrary co-domain T.
- Bounded or un-bounded domain.
- Integer or real coordinates.
- Discrete (grid or sparsely sampled) or continuous domain.

Examples:

- 1D, 2D, …, *n*D pixel image.
- interpolated pixel image.
- (interpolated) sparse *n*D sample set.
- virtual view into another image (transformed, sliced, …).
- procedurally generated image.

Abstractions

- **Accessible** ("Image")
  - Provides Accessors.
  - May provide bounds.
- **Accessor**
  - Is moved across the image.
  - Provides access to Types.
- **Type** ("Pixel value")
  - Represents sample value ∈ T.
  - Operations on T.



Types (Pixel Values)

- Concrete Pixel Types:
  - `UnsignedByteType`
  - `ByteType`
  - `ComplexFloatType`
  - ...
- Hierarchy of generic interfaces implemented by concrete types:
  - Every `NumericType` has add and multiply operations.
  - Every `Comparable` is equipped with a partial order.
  - …
- Algorithms are implemented to most abstract type.

Accessors

- **RandomAccess**:
  - Access pixels at specific coordinates.
- **Cursor** (iteration):
  - Visit every pixel once.
  - Arbitrary (but fixed) iteration order.

Virtualized Sample Access

- Calling .`get()` on accessor yields type T (pixel value)
- Pixels are *always* accessed through accessor interfaces.
- Allows for:
  - Arbitrary image data structures.
  - Procedural images.
  - Virtual coordinate and pixel value transformation (on-the-fly, no copying, transparent).

Virtualized Sample Access

$$f : \Omega \to \mathbb{T} \quad \text{with} \quad \Omega \subset \mathbb{R}^n$$

- **Views**:
  virtual coordinate transformation

$$g : \Omega' \to \Omega$$
$$f \circ g = f' : \Omega' \to \mathbb{T}$$

- **Converters**:
  virtual value transformation

$$h : \mathbb{T} \to \mathbb{T}'$$
$$h \circ f = f' : \Omega \to \mathbb{T}'$$

Setting up Code Examples

1. Clone the imglib2-introductory-workshop github repository into your workspace:

```
~/workspace$
git clone https://github.com/imglib/imglib2-introductory-workshop.git
```

2. Import the project into your IDE:

   - In Eclipse:
     File > Import…
     then select
     Maven > Existing Maven Projects
     then select the directory you just cloned.

   - In IntelliJ:
     TODO

Code Organization

Project imglib2-introductory-workshop.
One package per topic:

```
t01sandbox
t02accessors
t03types
...
```

t01: Loading Images and Displaying Results

- *Not* part of ImgLib2.

- Using ImageJ:

```
// open an ImageJ1 ImagePlus
ImagePlus imp = IJ.openImage( "http://imagej.net/images/clown.png" );
// wrap it as an ImgLib2 Img
Img<?> img = ImageJFunctions.wrap( imp );

...
// show ImgLib2 Img as an ImageJ1 (virtual) stack
Img< IntType > img2;
ImageJFunctions.show( img2 );
```

- Using ImageJ2:

```
// open an ImageJ2 Dataset (implements Img)
final Img< ? > img = ij.scifio().datasetIO().open("http://imagej.net/images/clown.png" );

...
// show ImgLib2 Img using ImageJ2
Img< IntType > img2;
ij.ui().show( img2 );
```

t02: Accessors

Images are manipulated using *Accessors*.
“Movable reference to a pixel.”

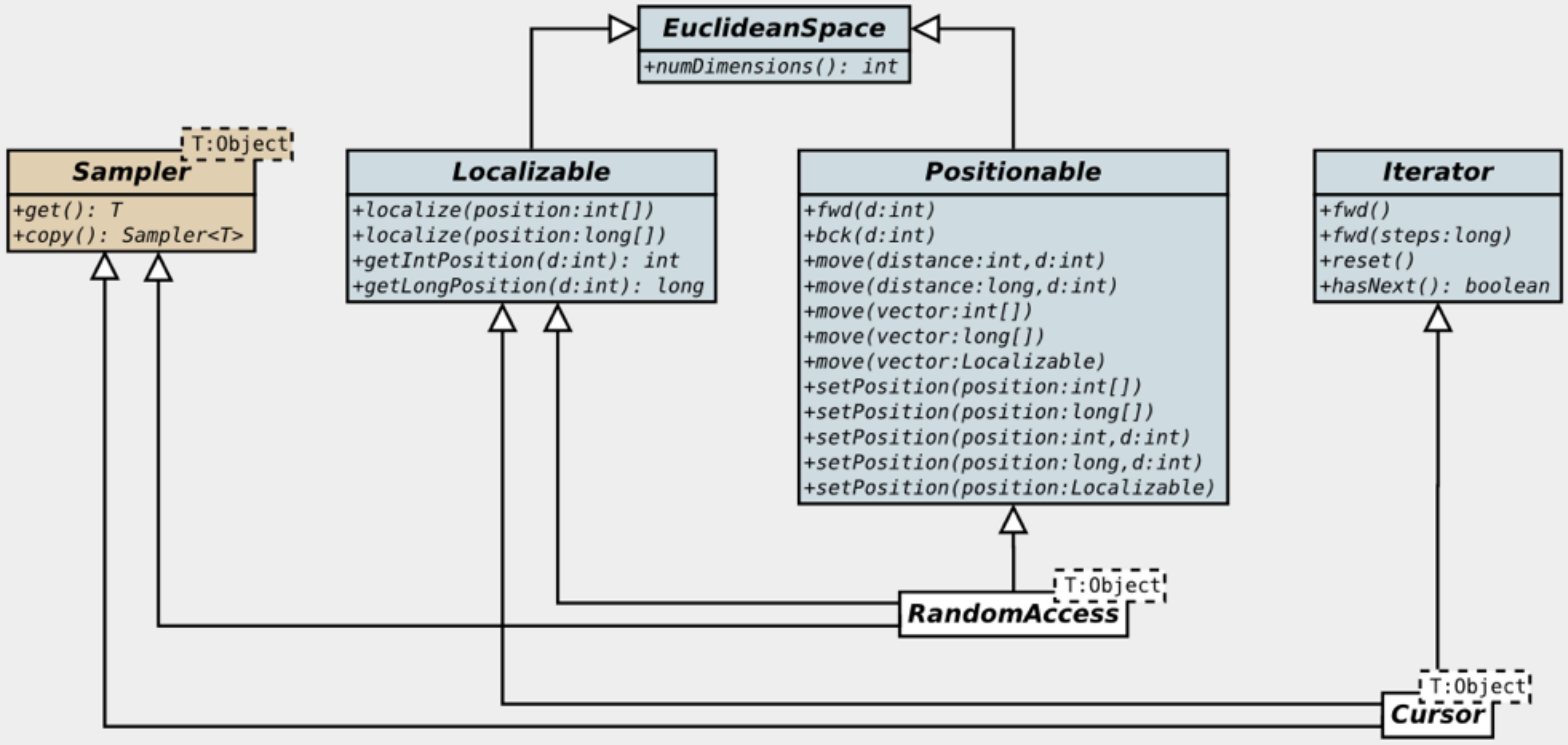
You can:

- Move it around the image
  (to specific coordinates, relative to current position, to “next” position, …)

- Ask it for its current position.

- De-reference it to get the pixel value.

Accessors
(simplified)


EuclideanSpace
+numDimensions(): int
Sampler
T:Object
+get(): T
+copy(): Sampler<T>
Localizable
+localize(position:int[])
+localize(position:long[])
+getIntPosition(d:int): int
+getLongPosition(d:int): long
Positionable
+fwd(d:int)
+bck(d:int)
+move(distance:int,d:int)
+move(distance:long,d:int)
+move(vector:int[])
+move(vector:long[])
+move(vector:Localizable)
+setPosition(position:int[])
+setPosition(position:long[])
+setPosition(position:int,d:int)
+setPosition(position:long,d:int)
+setPosition(position:Localizable)
Iterator
+fwd()
+fwd(steps:long)
+reset()
+hasNext(): boolean
RandomAccess
T:Object
Cursor
T:Object

t02: Accessors

ImgLib2

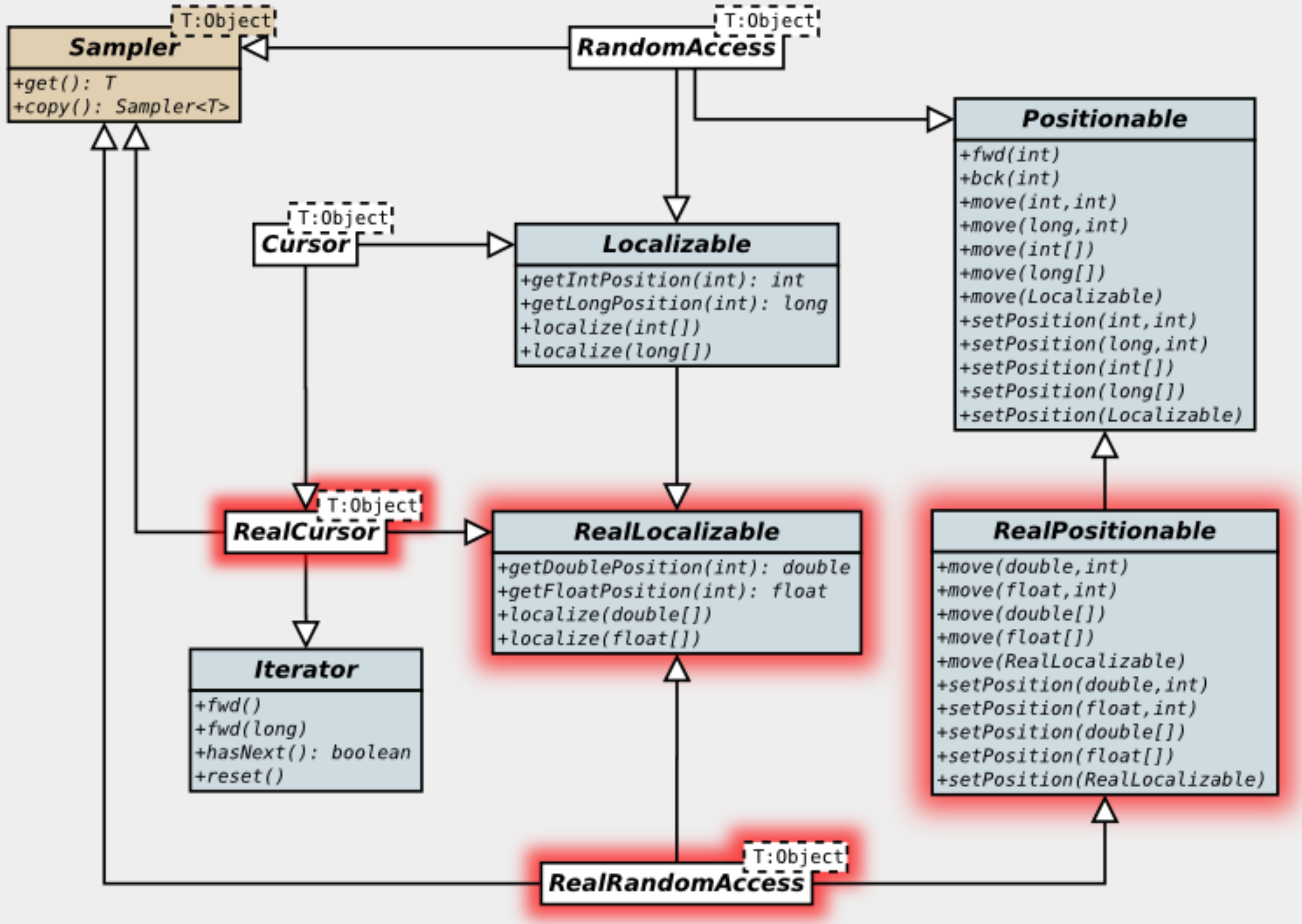
- **RandomAccess**:
  - Access pixels at specific coordinates.

```
T02E01RandomAccess
```

- **Cursor** (iteration):
  - Visit every pixel once.
  - Arbitrary (but fixed) iteration order.

```
T02E02Cursor
T02E03LocalizingCursor
T02E04IterationOrder
```

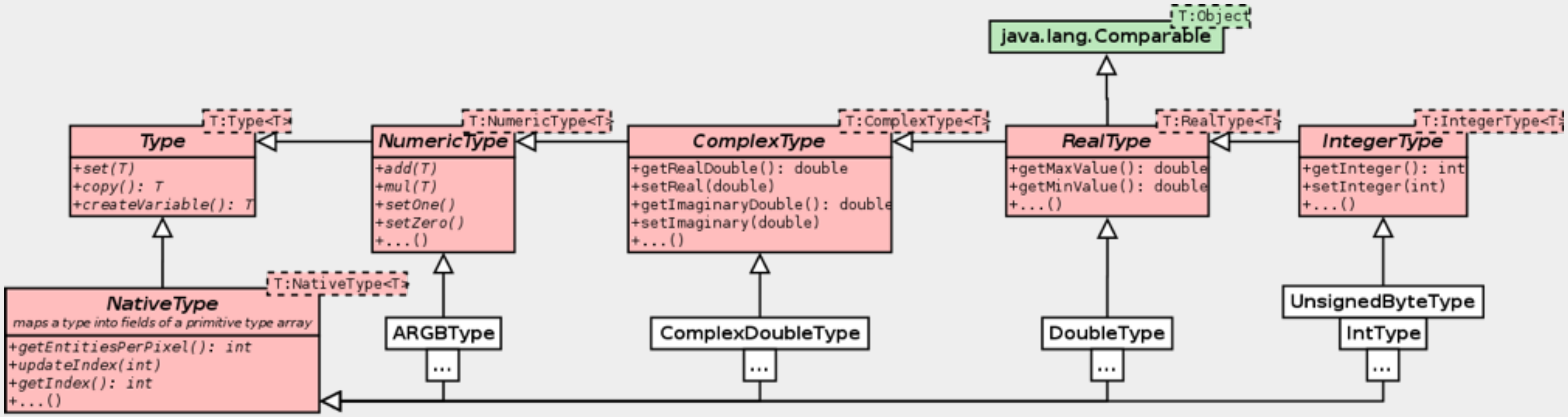
Accessors
(real coordinates)



t03: Types

- Types represent pixel values.
- Hierarchy of generic interfaces implemented by concrete types:
  - Every `NumericType` has add and multiply operations.
  - Every `Comparable` is equipped with a partial order.
  - …
- Algorithms are implemented to most abstract type.

Types



java.lang.Comparable
T:Object
Type
T:Type<T>
+set(T)
+copy(): T
+createVariable(): T
NumericType
T:NumericType<T>
+add(T)
+mul(T)
+setOne()
+setZero()
+...()
ComplexType
T:ComplexType<T>
+getRealDouble(): double
+setReal(double)
+getImaginaryDouble(): double
+setImaginary(double)
+...()
RealType
T:RealType<T>
+getMaxValue(): double
+getMinValue(): double
+...()
IntegerType
T:IntegerType<T>
+getInteger(): int
+setInteger(int)
+...()
NativeType
T:NativeType<T>
maps a type into fields of a primitive type array
+getEntitiesPerPixel(): int
+updateIndex(int)
+getIndex(): int
+...()
ARGBType
ComplexDoubleType
DoubleType
UnsignedByteType
IntType
...

t03: Types

- Types are used to get/set pixel values

    `T03E01Types`

- Algorithms use generic type interfaces

    `T03E02GenericCopy`

- NativeTypes are proxies into primitive arrays

    `T03E03Proxy`

t04: Accessibles

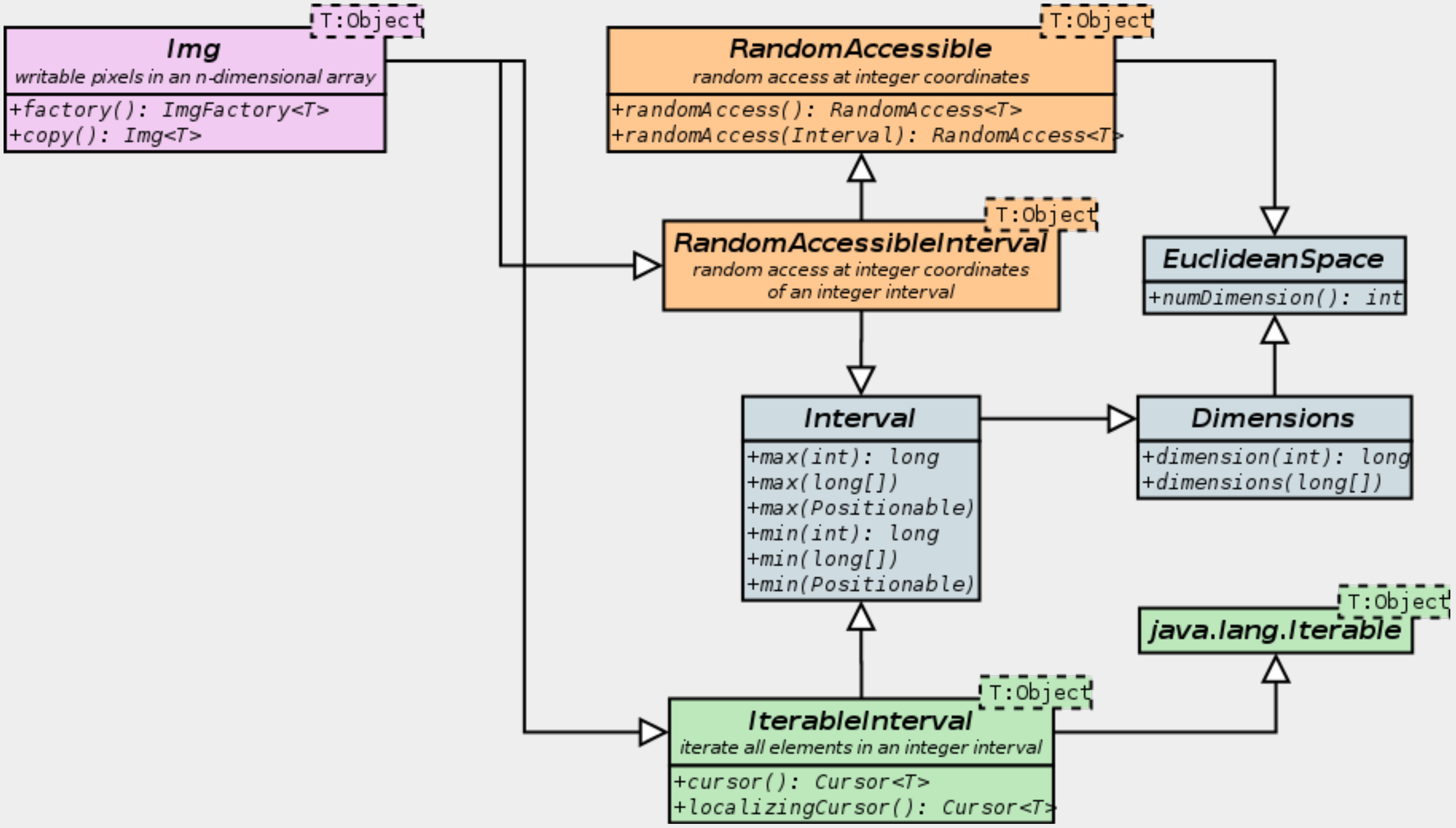
- **Accessibles** are “Images”:
  - Provide Accessors, *e.g.*,

    `RandomAccessible.randomAccess()` gives `RandomAccess`

    `IterableInterval.cursor()` gives `Cursor`
  - May provide bounds, e.g.,

    `RandomAccessibleInterval` extends `Interval`

Accessibles
(integral coordinates)



t04: Accessibles

- ImgFactories and Img implementations

```
T04E01ImgFactories
```

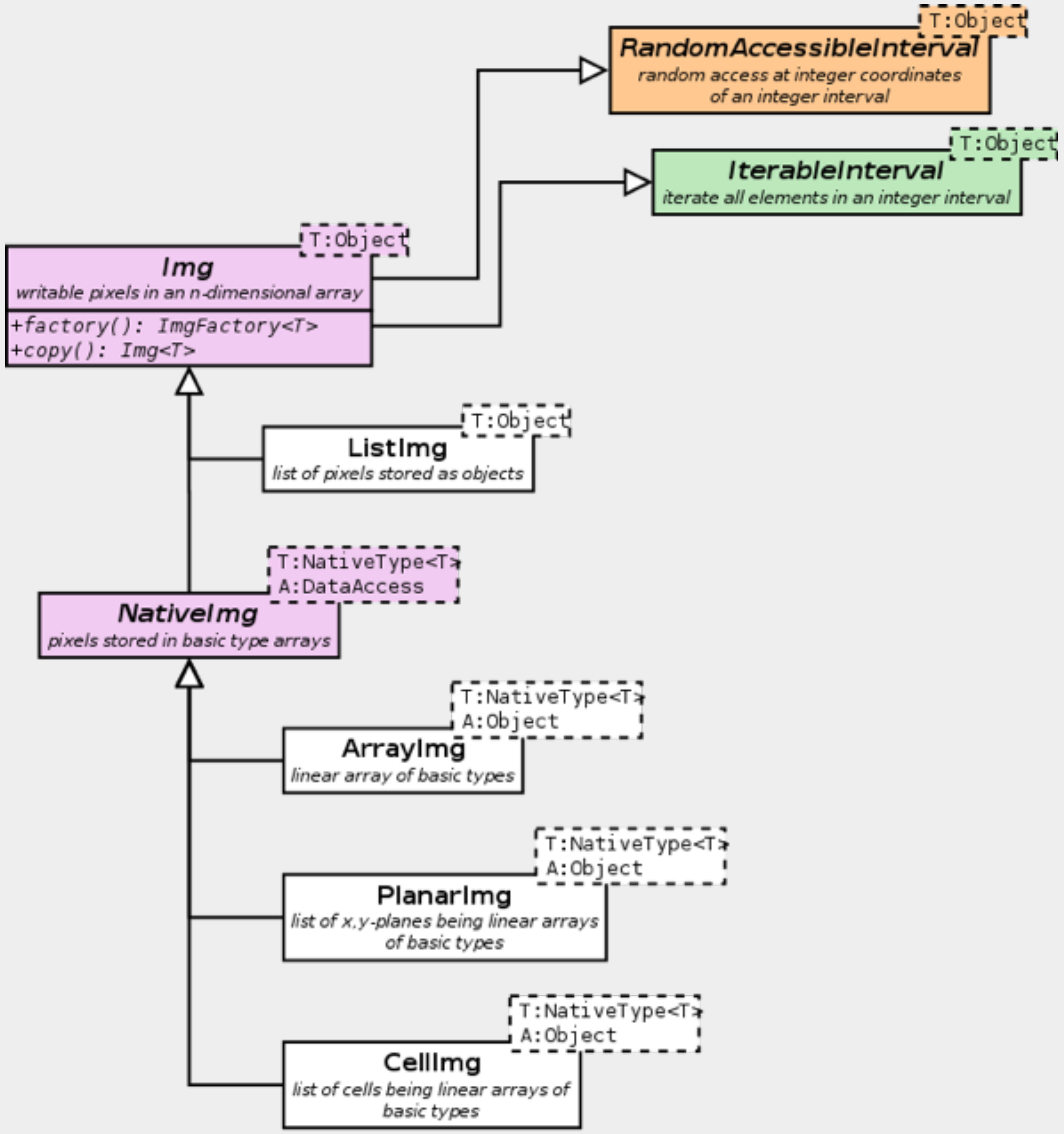
- Views and Converters transform Accessibles

```
T04E02Views
T04E03MoreViews
T04E04RealViews
T04E05Converters
T04E06EvenMoreViews
```

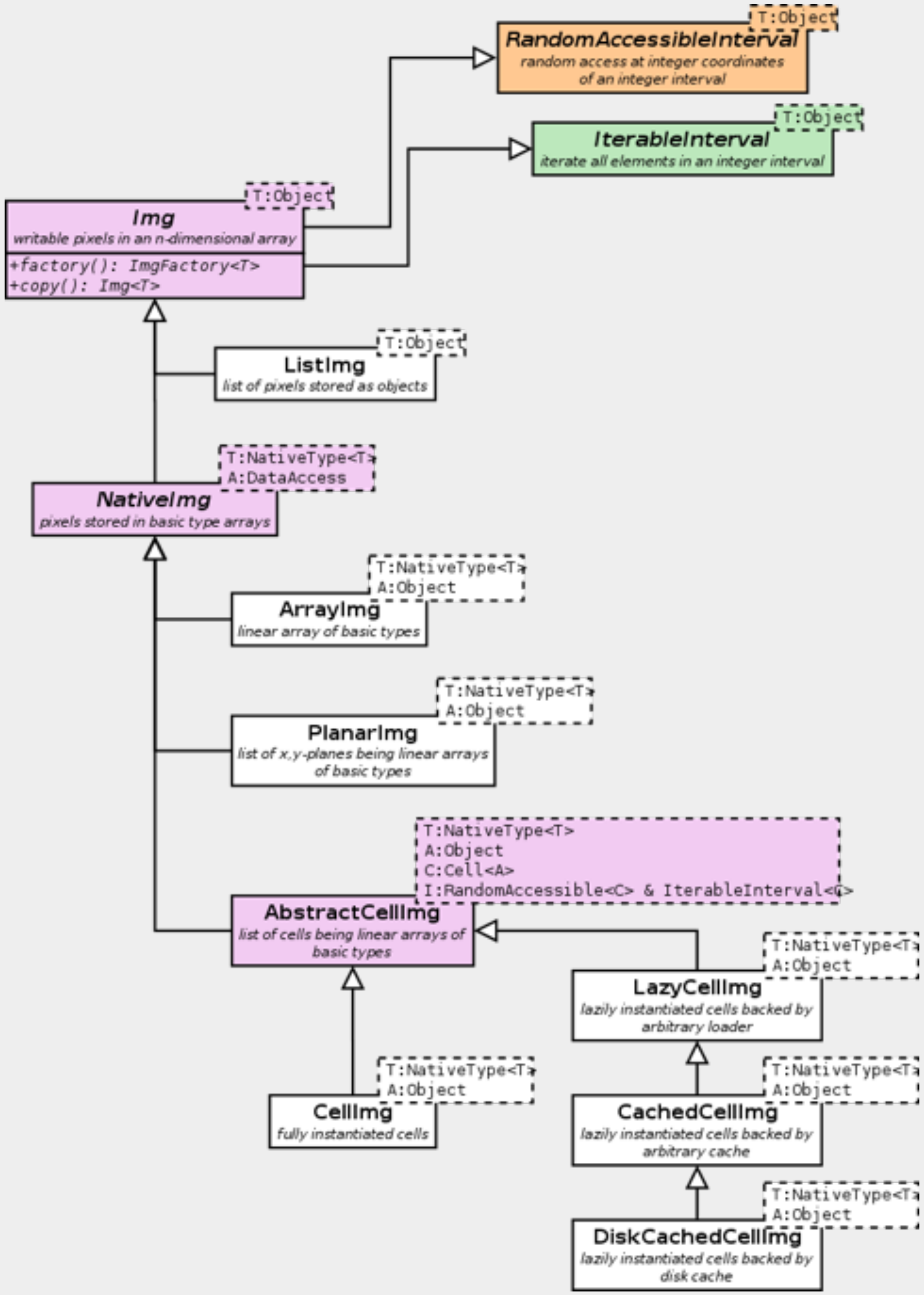
- Applications

```
T04E07Smoothing
T04E08SmoothingAndStacking
```

Imgs

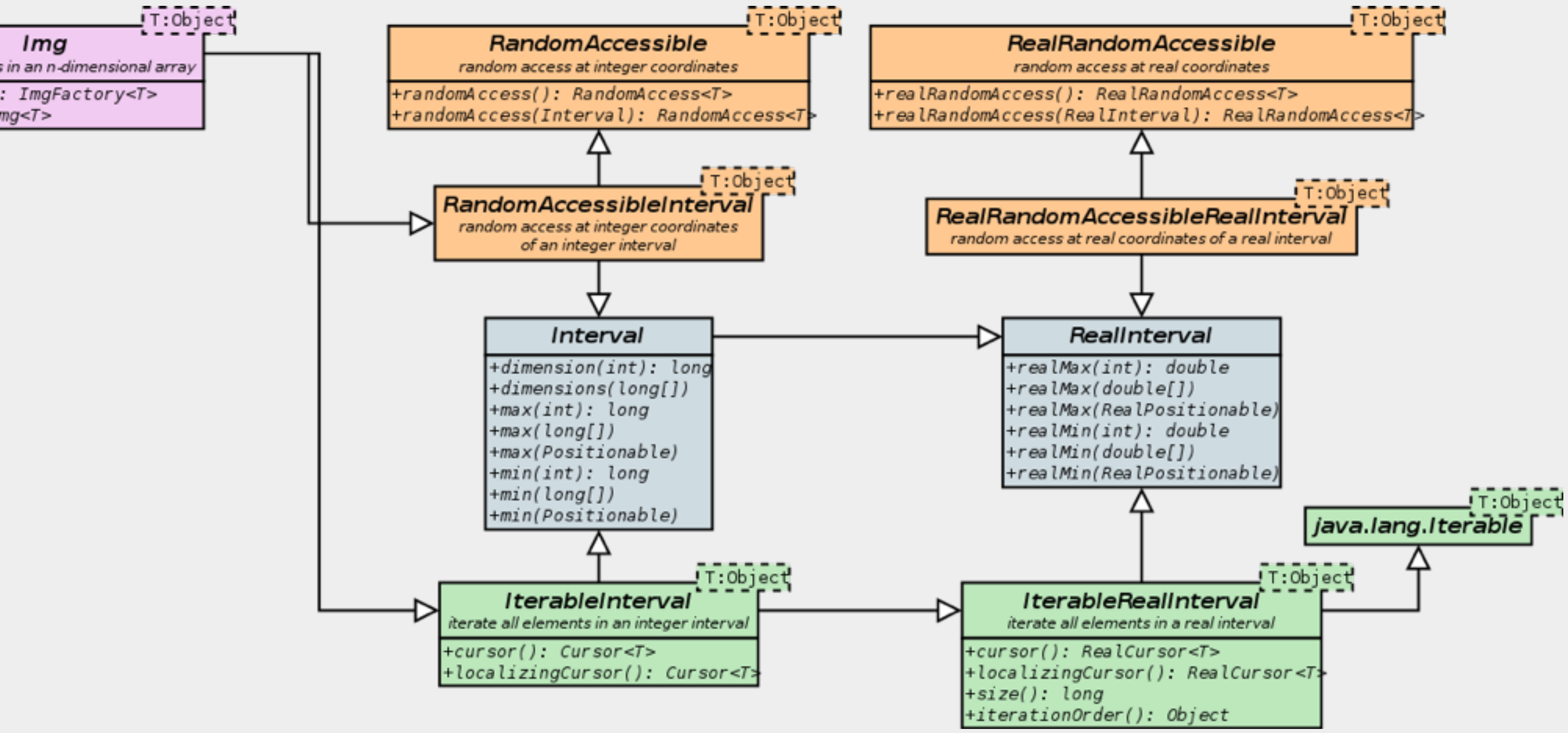

T:Object
RandomAccessibleInterval
random access at integer coordinates of an integer interval
T:Object
IterableInterval
iterate all elements in an integer interval
T:Object
Img
writable pixels in an n-dimensional array
+factory(): ImgFactory<T>
+copy(): Img<T>
T:Object
ListImg
list of pixels stored as objects
T:NativeType<T>
A:DataAccess
NativeImg
pixels stored in basic type arrays
T:NativeType<T>
A:Object
ArrayImg
linear array of basic types
T:NativeType<T>
A:Object
PlanarImg
list of x,y-planes being linear arrays of basic types
T:NativeType<T>
A:Object
CellImg
list of cells being linear arrays of basic types

Imgs



T:Object
RandomAccessibleInterval
random access at integer coordinates of an integer interval
T:Object
IterableInterval
iterate all elements in an integer interval
T:Object
Img
writable pixels in an n-dimensional array
+factory(): ImgFactory<T>
+copy(): Img<T>
T:Object
ListImg
list of pixels stored as objects
T:NativeType<T>
A:DataAccess
NativeImg
pixels stored in basic type arrays
T:NativeType<T>
A:Object
ArrayImg
linear array of basic types
T:NativeType<T>
A:Object
PlanarImg
list of x,y-planes being linear arrays of basic types
T:NativeType<T>
A:Object
C:Cell<A>
I:RandomAccessible<C> & IterableInterval<C>
AbstractCellImg
list of cells being linear arrays of basic types
T:NativeType<T>
A:Object
LazyCellImg
lazily instantiated cells backed by arbitrary loader
T:NativeType<T>
A:Object
CellImg
fully instantiated cells
T:NativeType<T>
A:Object
CachedCellImg
lazily instantiated cells backed by arbitrary cache
T:NativeType<T>
A:Object
DiskCachedCellImg
lazily instantiated cells backed by disk cache

Accessibles
(real coordinates)



Img
T:Object
s in an n-dimensional array
: ImgFactory<T>
mg<T>
RandomAccessible
T:Object
random access at integer coordinates
+randomAccess(): RandomAccess<T>
+randomAccess(Interval): RandomAccess<T>
RealRandomAccessible
T:Object
random access at real coordinates
+realRandomAccess(): RealRandomAccess<T>
+realRandomAccess(RealInterval): RealRandomAccess<T>
RandomAccessibleInterval
T:Object
random access at integer coordinates of an integer interval
RealRandomAccessibleRealInterval
T:Object
random access at real coordinates of a real interval
Interval
+dimension(int): long
+dimensions(long[])
+max(int): long
+max(long[])
+max(Positionable)
+min(int): long
+min(long[])
+min(Positionable)
RealInterval
+realMax(int): double
+realMax(double[])
+realMax(RealPositionable)
+realMin(int): double
+realMin(double[])
+realMin(RealPositionable)
java.lang.Iterable
T:Object
IterableInterval
T:Object
iterate all elements in an integer interval
+cursor(): Cursor<T>
+localizingCursor(): Cursor<T>
IterableRealInterval
T:Object
iterate all elements in a real interval
+cursor(): RealCursor<T>
+localizingCursor(): RealCursor<T>
+size(): long
+iterationOrder(): Object

Resources

- https://imagej.net/ImgLib2
- http://forum.imagej.net/tags/c/development/imglib
- https://github.com/imglib/imglib2-tutorials
- https://github.com/imglib/imglib2-introductory-workshop
- https://github.com/imglib/imglib2-advanced-workshop
- https://github.com/imglib/imglib2-cache-examples

t05: Labeling

- No slides, just examples …

```
T05E07Smoothing
T05E01Labeling.java
T05E02LabelRepresentation.java
T05E03ObjectSegmentation1.java
T05E04ObjectSegmentation2.java
T05E05LabelRegions1.java
T05E06LabelRegions2.java
T05E07Unfinished.java
```