Introduction to Ops:
Usage and Development

Zach Petersen

Leon Yang

Nov 3rd, 2015

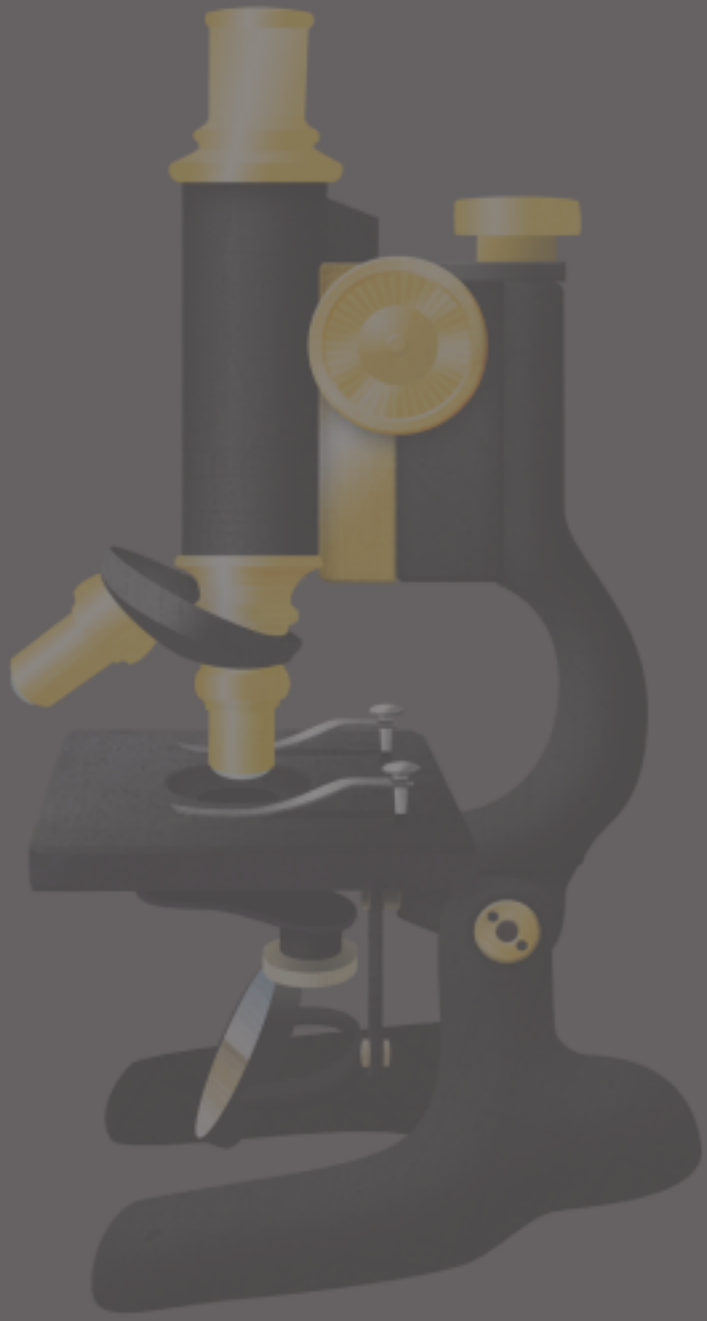
Warm Up: Tutorials

https://github.com/imagej/imagej-tutorials/tree/master/using-ops/src/main/java

- UsingOps
  - Overview of ops
- UsingOpTypes (later)
  - Overview of SpecialOps
- CreateANewOp

How Images are Represented

- IterableInterval
  - Uses cursors
- RandomAccessibleInterval
  - Random access
- Img
  - Implements both
- From imglib2



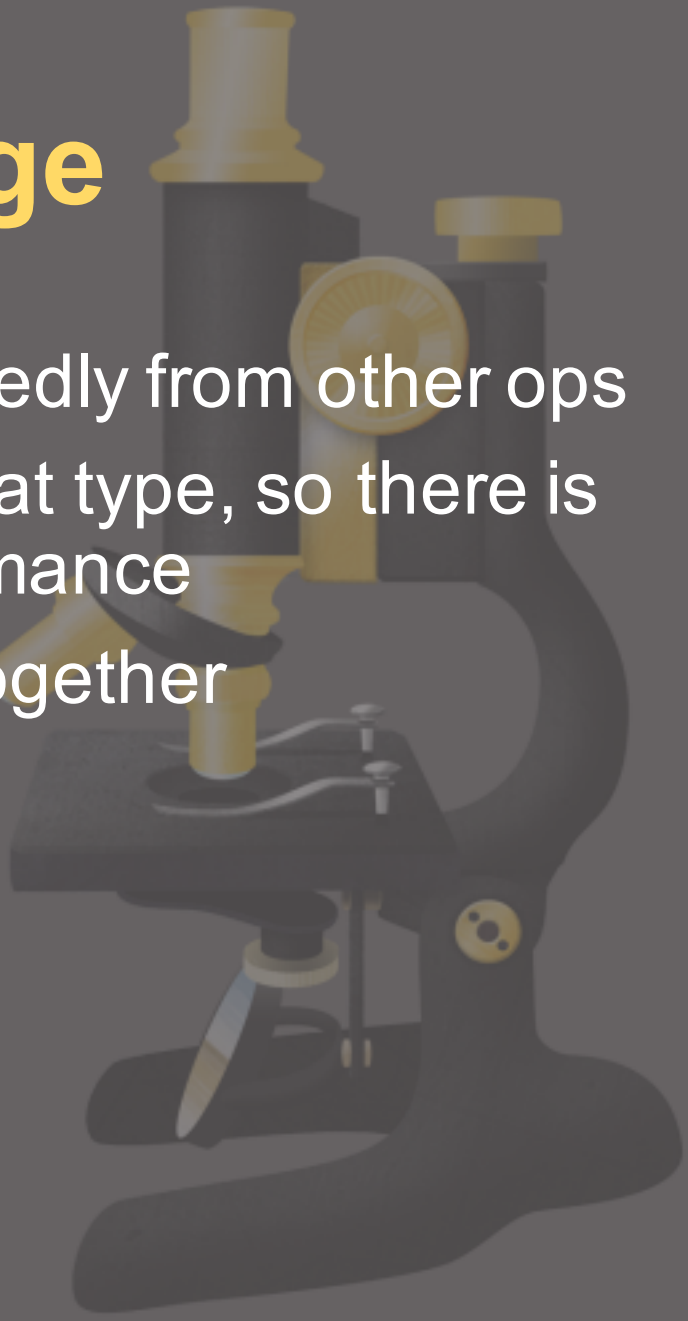
SpecialOps: Four types

- ComputerOp: Stores result into the specified output reference.
- FunctionOp: Returns output as a new object
- HybridOp: Can be used either as a function or computer op
- InplaceOp: Mutates the given input reference

```
DoubleType out = new DoubleType();
ComputerOp<DoubleType, DoubleType> add5 =
        ij.op().computer(Ops.Math.Add.class, DoubleType.class, DoubleType.class, 5.0);
add5.compute(new DoubleType(5.0), out);
```

SpecialOps: Usage

- Intended to be used repeatedly from other ops
- They only look for ops of that type, so there is an improved search performance
- Intuitive, can easily chain together



SpecialOps: Helper classes

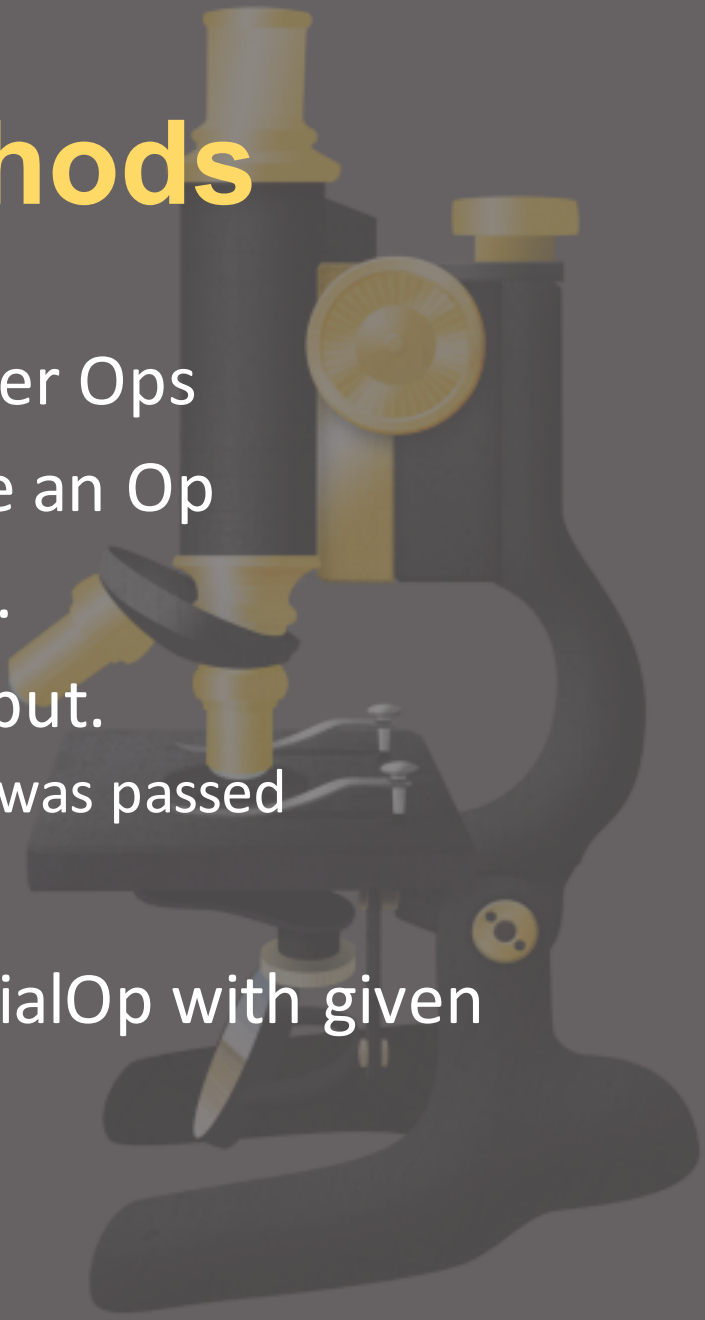
- Raw type if generics used as I/O and not parameterized
  - Unsafe, can lead to unseen errors

```
ComputerOp<IterableInterval, RandomAccessibleInterval> badAdd =
    ij.op().computer(Ops.Math.Add.class, RandomAccessibleInterval.class, IterableInterval.class, in);
```

- Avoid by using helper classes
  - Lets you parameterize input and output
  - Currently have RAIs, IIs, RTs
  - Could also pass in instances instead

```
ComputerOp<IterableInterval<DoubleType>, RandomAccessibleInterval<DoubleType>> goodAdd =
    RAIs.computer(ij.op(), Ops.Math.Add.class, ii, value);

ComputerOp<IterableInterval<DoubleType>, RandomAccessibleInterval<DoubleType>> goodAdd2 =
        ij.op().computer(Ops.Math.Add.class, rai, ii, value);
```

SpecialOps: Methods



- ops() – gateway to access other Ops
- run() – default way to execute an Op
- in() – gets the template input.
- out() – gets the template output.
  - Both return null if class object was passed
- initialize() – initialize the Op
- compute() - execute the SpecialOp with given parameters

SpecialOps: Methods

- Why use initialize()?

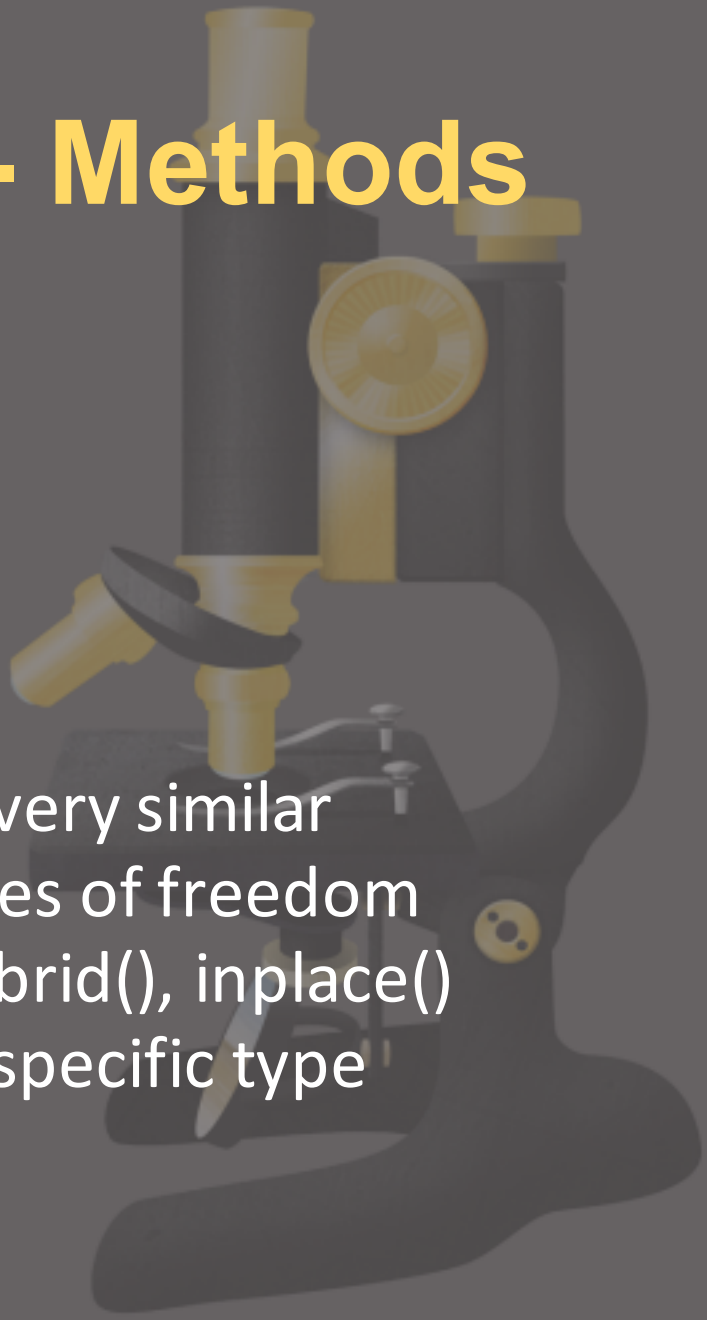
```
@Override¤¶
public·void·initialize()·{¤¶
»  final·double[]·sigmas1·=·new·double[in().numDimensions()];¤¶
»  final·double[]·sigmas2·=·new·double[in().numDimensions()];¤¶
@Override¤¶
public·void·compute(final·RandomAccessibleInterval<T>·input,¤¶
»  final·RandomAccessibleInterval<T>·output)¤¶
{¤¶
»  final·double[]·sigmas1·=·new·double[input.numDimensions()];¤¶          2,¤¶
»  final·double[]·sigmas2·=·new·double[input.numDimensions()];¤¶

»  Arrays.fill(sigmas1,·sigma1);¤¶
»  Arrays.fill(sigmas2,·sigma2);¤¶

»  ops().run(Ops.Filter.DoG.class,·output,·input,·sigmas1,·sigmas2,·fac);
}¤¶
»  final·RandomAccessibleInterval<T>·output)¤¶
{¤¶
»  op.compute(input,·output);¤¶
}¤¶
```

SpecialOps: Tutorial

- UsingOpTypes
- https://github.com/imagej/imagej-tutorials/tree/ops-tutorials/using-ops/src/main/java

OpEnvironment - Methods

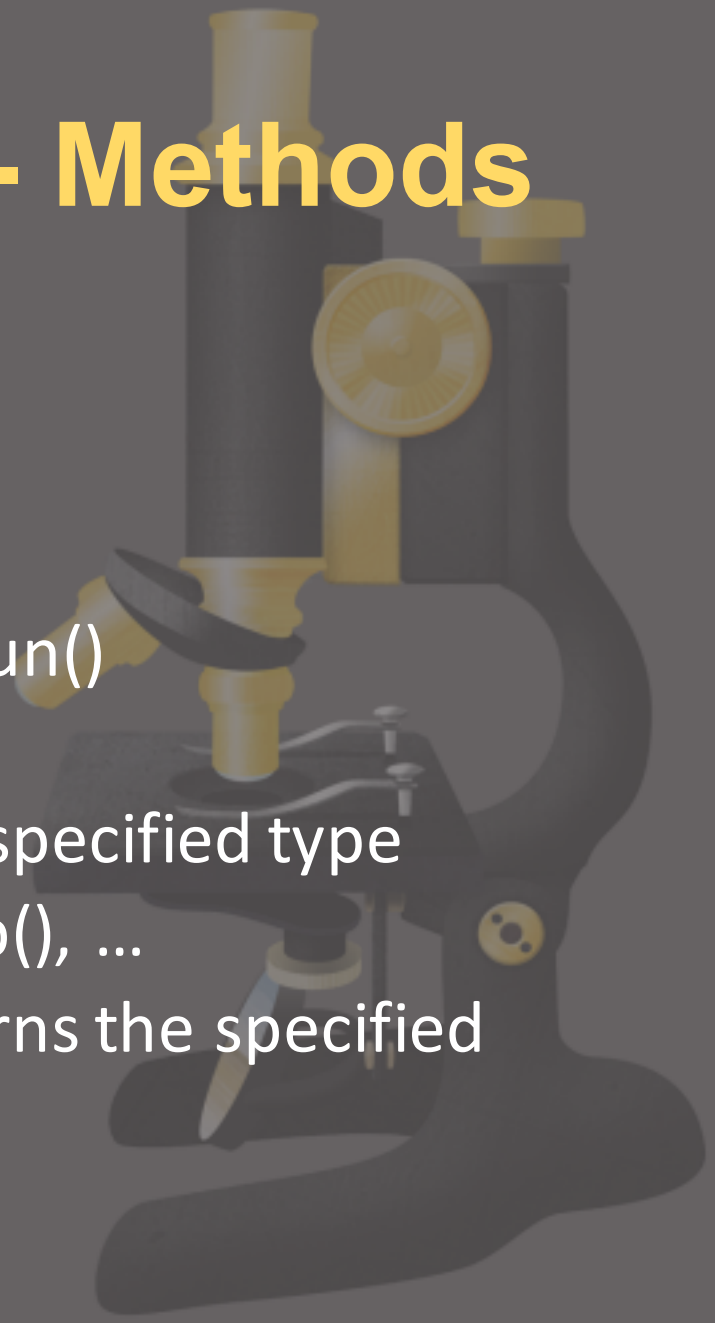
OpEnvironment - Methods



- Get an Op
  - op()
    - matches all Ops
  - module()
    - not frequently used
    - not exactly an Op, but very similar
    - low-level, higher degrees of freedom
  - computer(), function(), hybrid(), inplace()
    - only match Ops of the specific type
    - higher performance

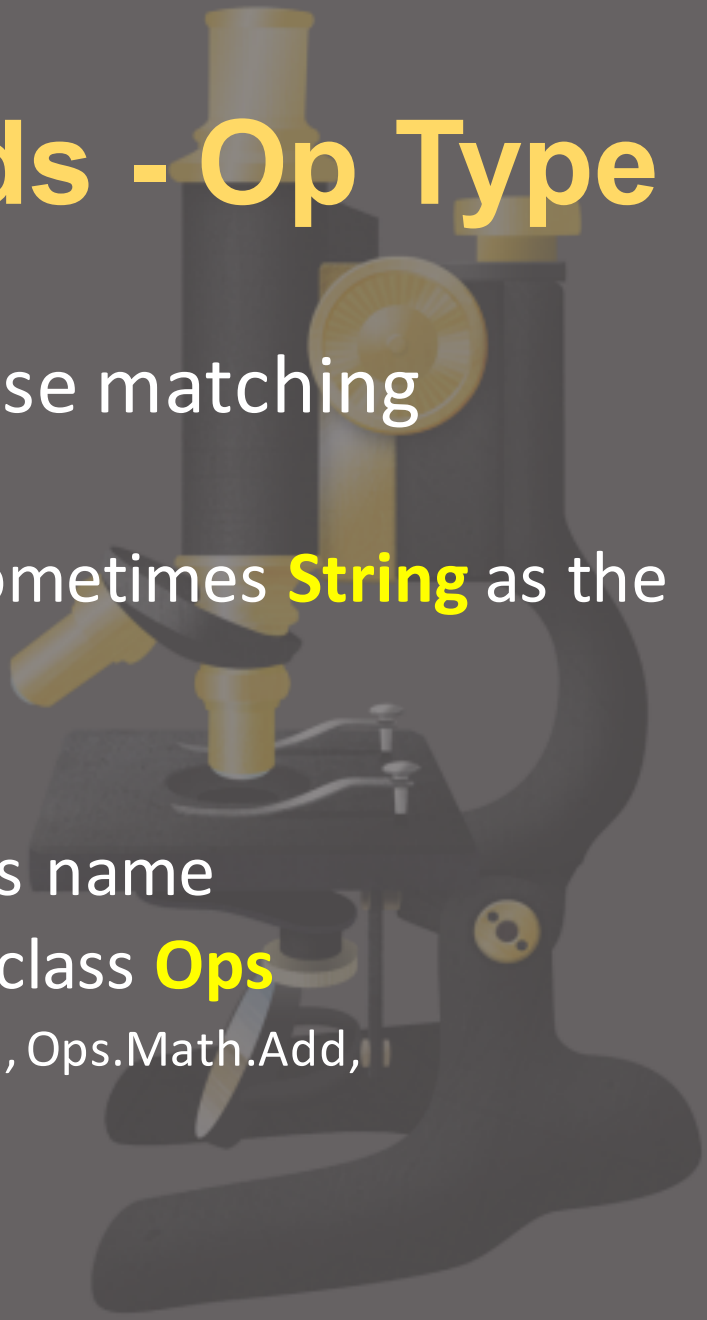
OpEnvironment - Methods

- Execute an Op
  - run()
    - basically the same as op(SameParameters).run()
  - map(), join(), help(), ...
    - execute the Op of the specified type
  - math().add(), image().crop(), …
    - math() or image() returns the specified "Namespace"



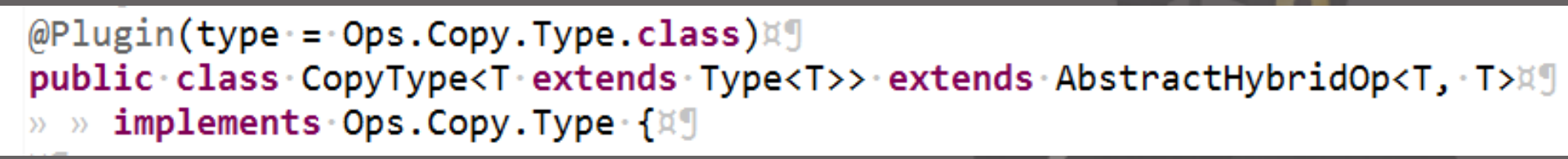
Matching Methods - Op Type

- The first parameter in those matching methods
  - Accepts **class object**, or sometimes **String** as the name of the Op type
- Marker interface
  - Contains nothing except its name
  - All of them live inside the class **Ops**
    - For example: Ops.Map, Ops.Join, Ops.Math.Add, Ops.Image.Crop



Matching Methods - Op Type

- Actual (implemented) class
  - Not usually used
  - Can do more specific matching
- Question:
  - what are the marker interface and the implemented class, respectively?



```
@Plugin(type = Ops.Copy.Type.class)¤¶
public class CopyType<T extends Type<T>> extends AbstractHybridOp<T, T>¤¶
» » implements Ops.Copy.Type {¤¶
```

Matching Methods - I/O Types

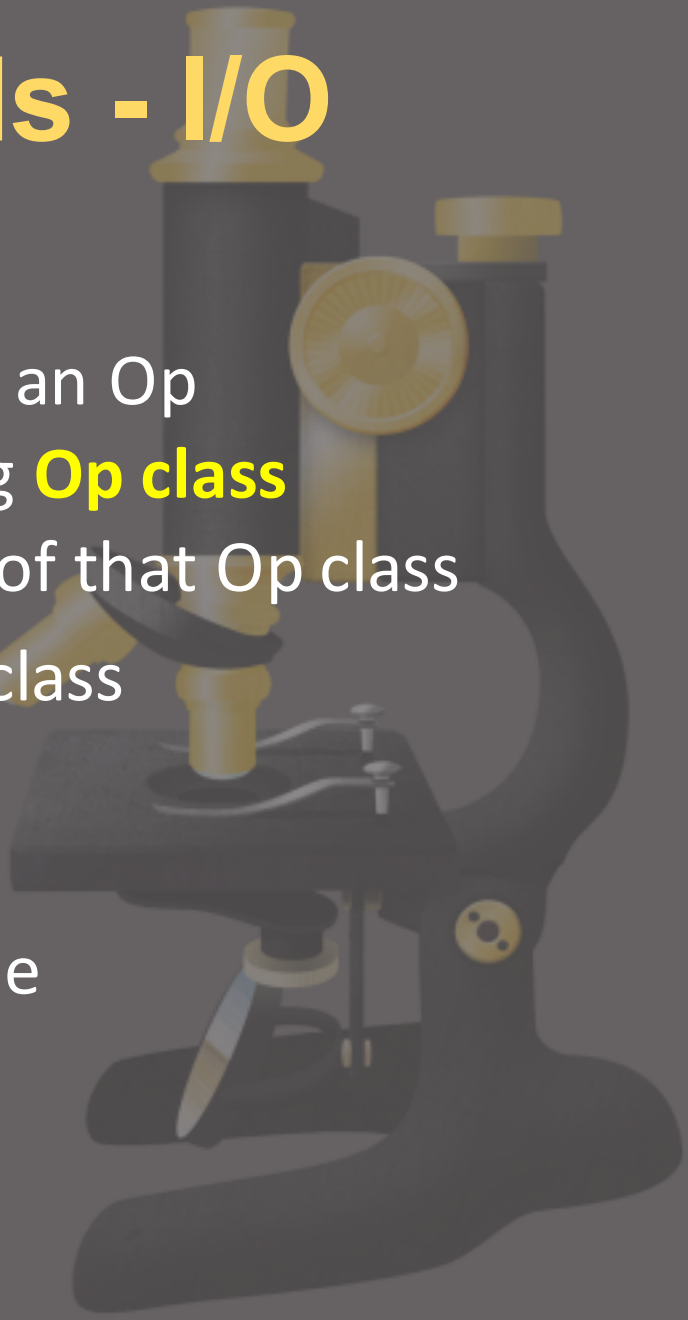
- Usually the parameters right after the Op type
- In most cases, we can pass either a **class object** or an **instance**, of the type
  - for example: "Img.class" vs. "new Img()"
- Three overloaded versions of the computer() methods

```
computer(Class<OP>, Class<O>, Class<I>, Object...) <I, O, OP extends Op> : ComputerOp<I, O>
computer(Class<OP>, Class<O>, I, Object...) <I, O, OP extends Op> : ComputerOp<I, O>
computer(Class<OP>, O, I, Object...) <I, O, OP extends Op> : ComputerOp<I, O>
```

- If we can pass a class object to get an Op, why do we bother passing a default/empty instance?

Matching Methods - I/O Types

- Either to get an Op or execute an Op
  - First find the best matching **Op class**
  - Then initialize **an instance** of that Op class
- To find the best matching Op class
  - Only cares about types
- To initialize an Op
  - Use the instances if possible
  - Default values otherwise



Matching Methods - I/O Types

- Not only "**type indicators**"
- But also "**template parameters**“
- Passing class objects as I/O type parameters could sometimes causes unexpected behaviors

Matching Methods - I/O Types

- For example, to get the CopyIterableInterval Op, both statements seem to work
  - ops().hybrid(Ops.Copy.IterableInterval.class, null, img);
  - ops().hybrid(Ops.Copy.IterableInterval.class, null, img.getClass());
- However, the second statement (the one uses class object) throws an exception
- This is because when initializing the actual Op, the input parameter's method is called to get a helper Op

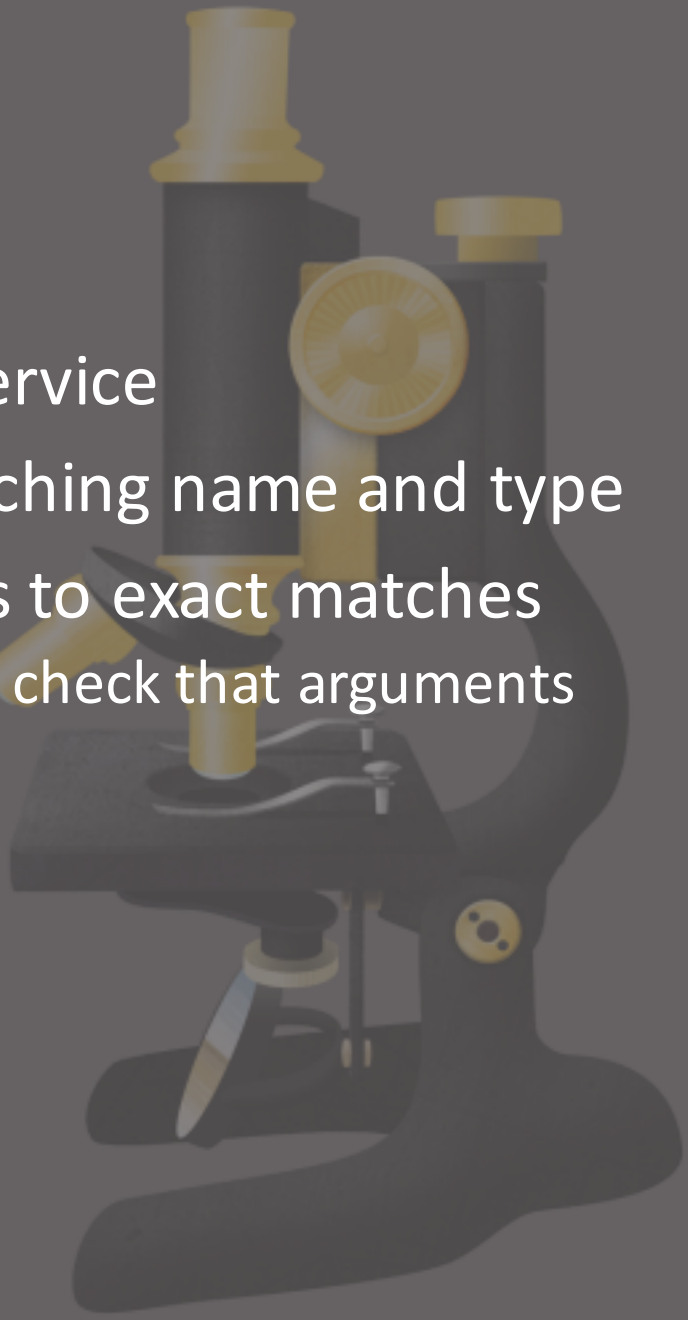
Matching Methods - I/O Types

Question: What Exception will be thrown?

```
@Plugin(type = Ops.Copy.IterableInterval.class, priority = 1.0)
public class CopyIterableInterval<T> extends
    AbstractHybridOp<IterableInterval<T>, IterableInterval<T>> implements
    Ops.Copy.IterableInterval, Contingent {

    @Parameter
    protected OpService ops;

    // used internally
    private ComputerOp<IterableInterval<T>, IterableInterval<T>> map;

    @Override
    public void initialize() {
        map = ops.computer(Ops.Map.class, in(), in(),
            ops.computer(Ops.Copy.Type.class,
                in().firstElement().getClass(),
                in().firstElement().getClass()));
    }
```

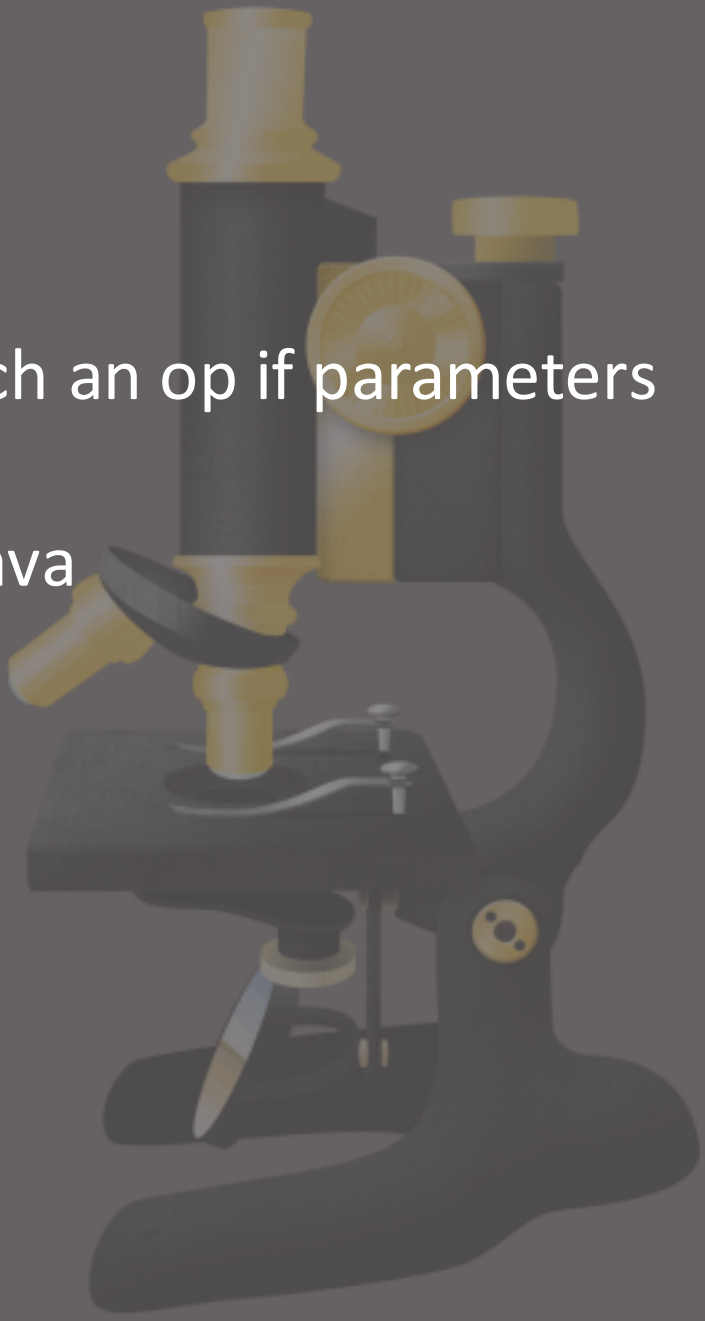
Matching Ops

- Done in DefaultOpMatchingService
- First find candidates with matching name and type
- Then narrow down candidates to exact matches
  - Find highest priority match and check that arguments match
- Initialize the op



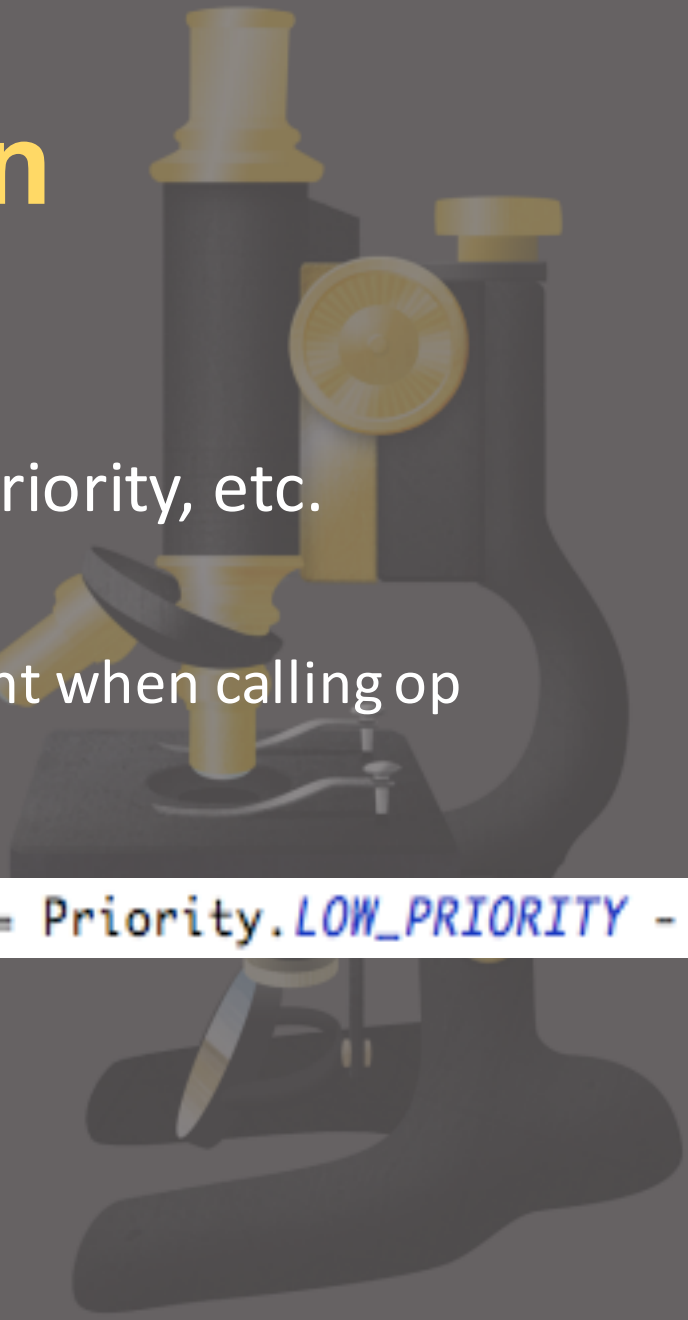
Converters

- Can sometimes convert to match an op if parameters are not a perfect match
- Done by ConvertService in SciJava



Plugin Annotation

- @Plugin
- Defines the ops type, name, priority, etc.
- Used for matching ops
  - Type compared to first argument when calling op



```
@Plugin(type = Ops.Map.class, priority = Priority.LOW_PRIORITY - 1)
```

Parameter Annotation

- @Parameter
  - Indicates field is input or output
- Which type of SpecialOp are the parameter declarations for?

```
@Parameter(type = ItemIO.OUTPUT)
private O out;

@Parameter
private I in;
```

FunctionOp

```
@Parameter(type = ItemIO.BOTH, required = false)
private O out;

@Parameter
private I in;
```

HybridOp

- Thank you for listening to our presentation!
- Questions?