# Overview of Java

## Combining Object-Oriented & Functional Programming

Douglas C. Schmidt

# Learning Objectives in This Lesson

- Recognize the benefits of combining object-oriented and functional programming in Java.

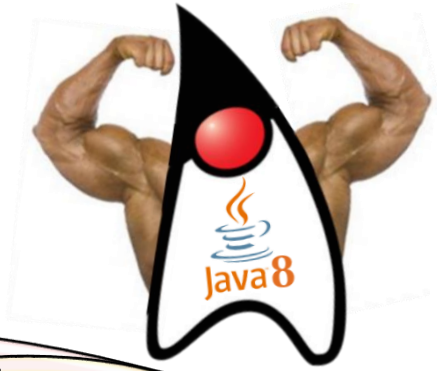

Again, we'll show Java code fragments that'll be covered in more detail later.

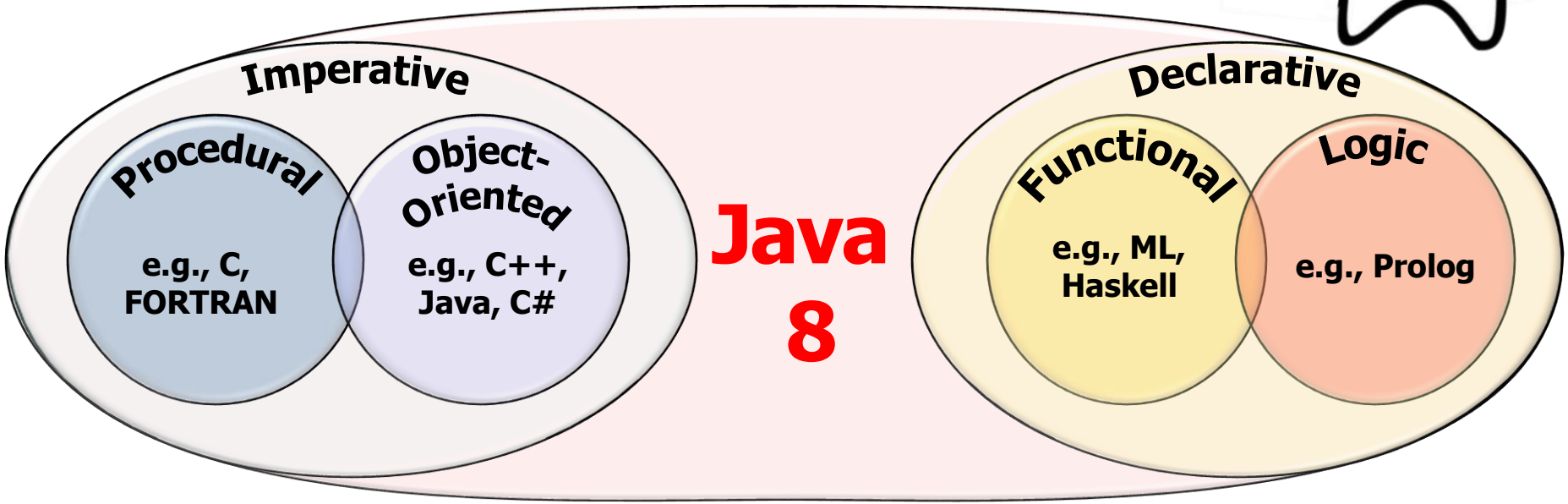
Douglas C. Schmidt

# Combining Object-Oriented & Functional Programming in Java

# Combining Object-Oriented & Functional Programming in Java

- Java's combination of functional & object-oriented paradigms is powerful!



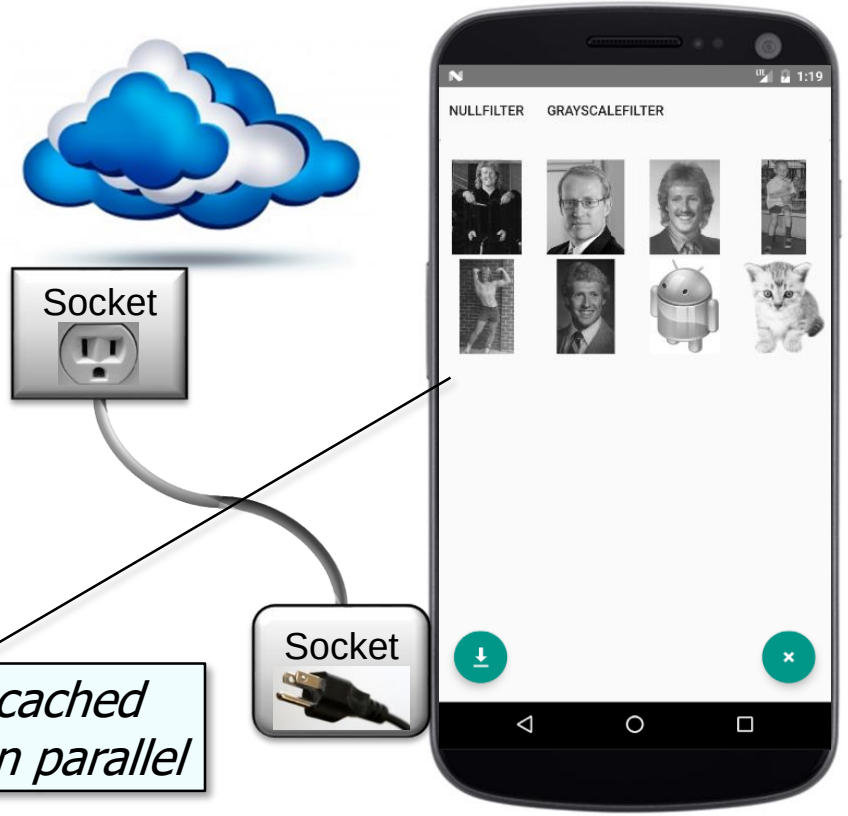
# Combining Object-Oriented & Functional Programming in Java

- Java's functional features help close the gap between a program's "domain intent" & its computations


See www.toptal.com/software/declarative-programming

# Combining Object-Oriented & Functional Programming in Java

- Java's functional features help close the gap between a program's "domain intent" & its computations, e.g.,
  - Domain intent defines "what."



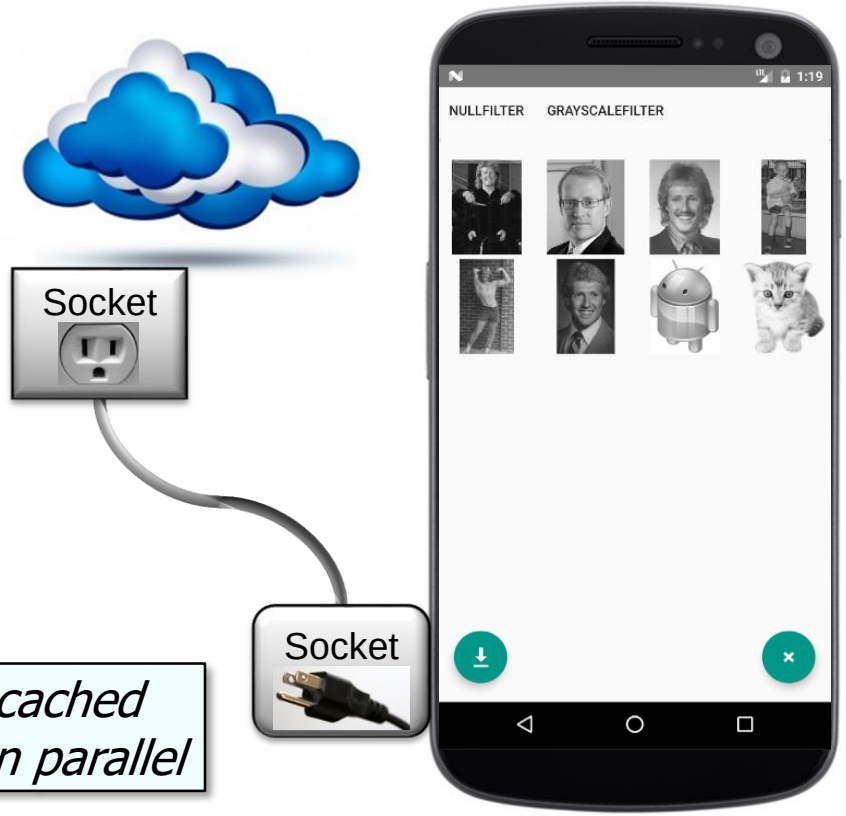

*Process a list of URLs to images that aren't cached and download/transform/store these images in parallel*

See github.com/douglascraigschmidt/LiveLessons/tree/master/ImageStreamGang

# Combining Object-Oriented & Functional Programming in Java

- Java's functional features help close the gap between a program's "domain intent" & its computations, e.g.,
  - Domain intent defines "what."
  - Computations define "how."

```
List<Image> images = urls
   .parallelStream()
   .filter(not(this::urlCached))
   .map(this::downloadImage)
   .flatMap(this::applyFilters)
   .collect(toList());
```



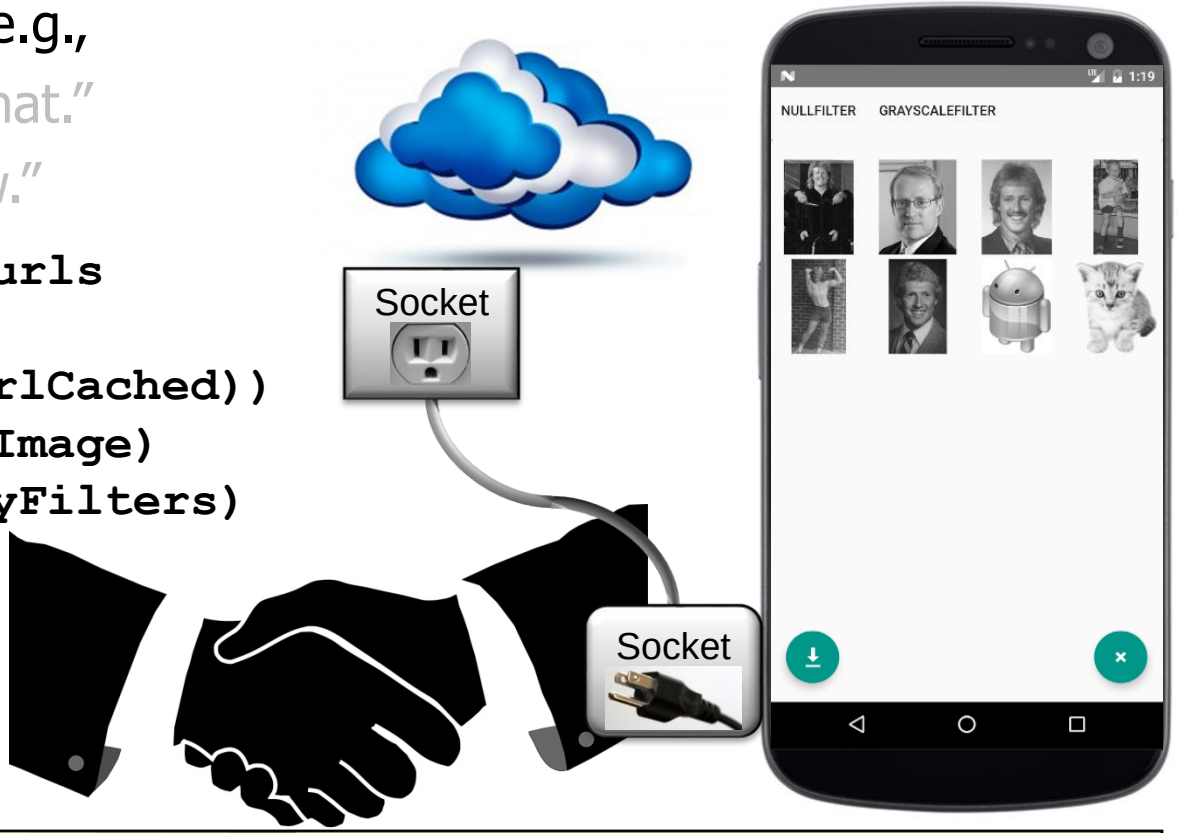

*Process a list of URLs to images that aren't cached and download/transform/store these images in parallel*

See github.com/douglascraigschmidt/LiveLessons/tree/master/ImageStreamGang

# Combining Object-Oriented & Functional Programming in Java

- Java's functional features help close the gap between a program's "domain intent" & its computations, e.g.,
  - Domain intent defines "what."
  - Computations define "how."

```
List<Image> images = urls
   .parallelStream()
   .filter(not(this::urlCached))
   .map(this::downloadImage)
   .flatMap(this::applyFilters)
   .collect(toList());
```

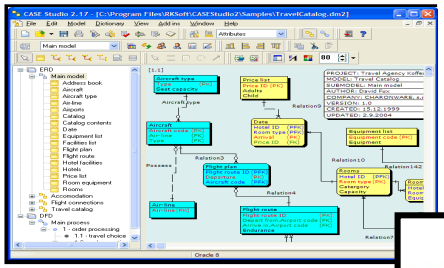


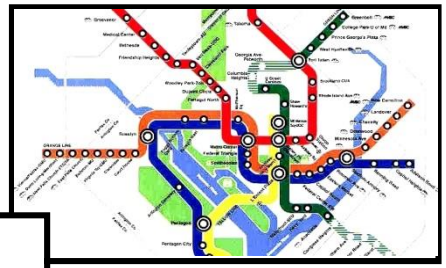

Java functional programming features connect domain intent & computations.

# Combining Object-Oriented & Functional Programming in Java

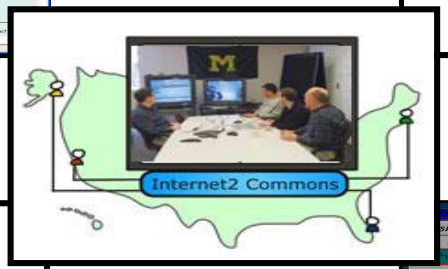
- Likewise, Java's object-oriented features help to structure a program's software architecture.

**Logical View**



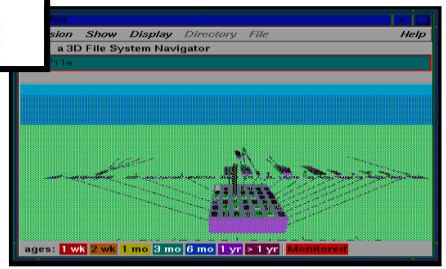


**Process View**




**Physical View**

**Use Case View**



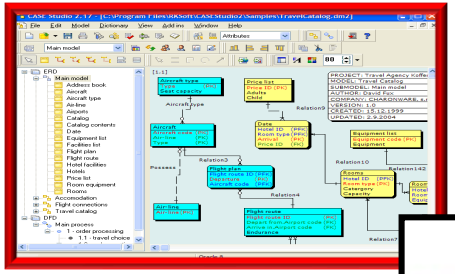

**Development View**

See en.wikipedia.org/wiki/Software_architecture

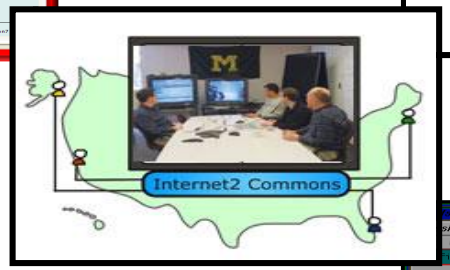
# Combining Object-Oriented & Functional Programming in Java

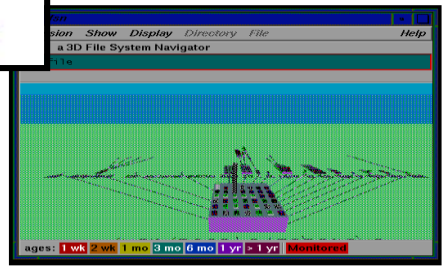
- Likewise, Java's object-oriented features help to structure a program's software architecture.

**Logical View**



**Process View**



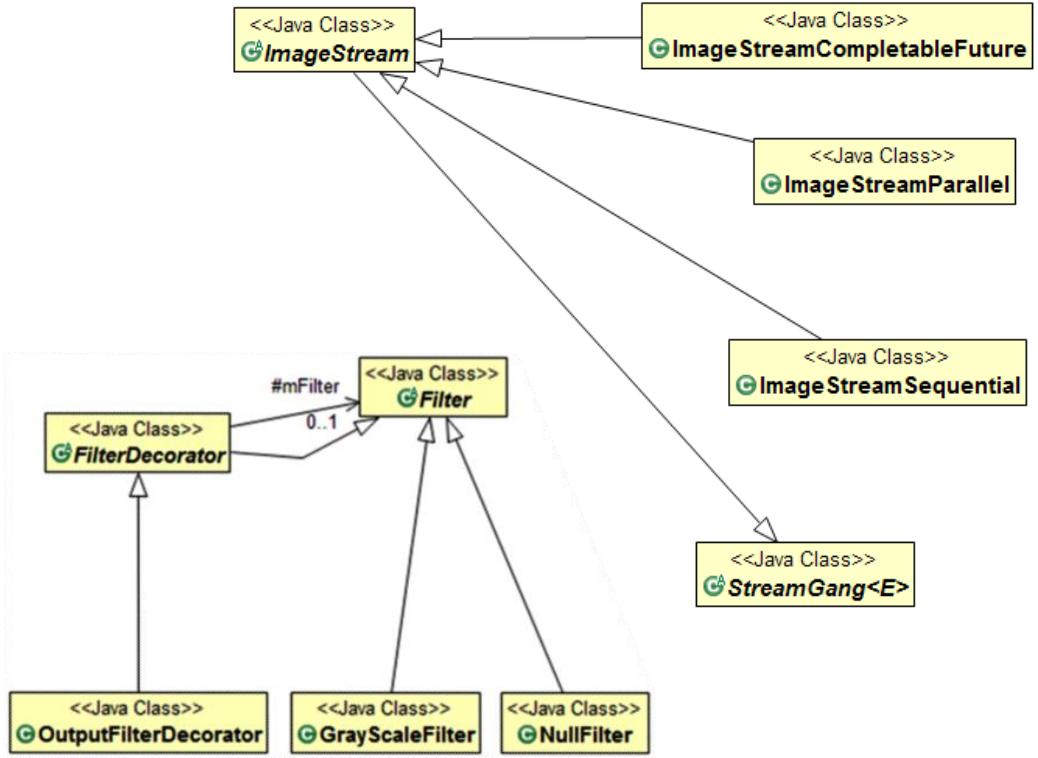

**Development View**

*This view depicts subsystems, packages, & classes that exhibit architecturally important structure & behavior.*

See sce.uhcl.edu/helm/rationalunifiedprocess/process/workflow/ana_desi/co_lview.htm

# Combining Object-Oriented & Functional Programming in Java

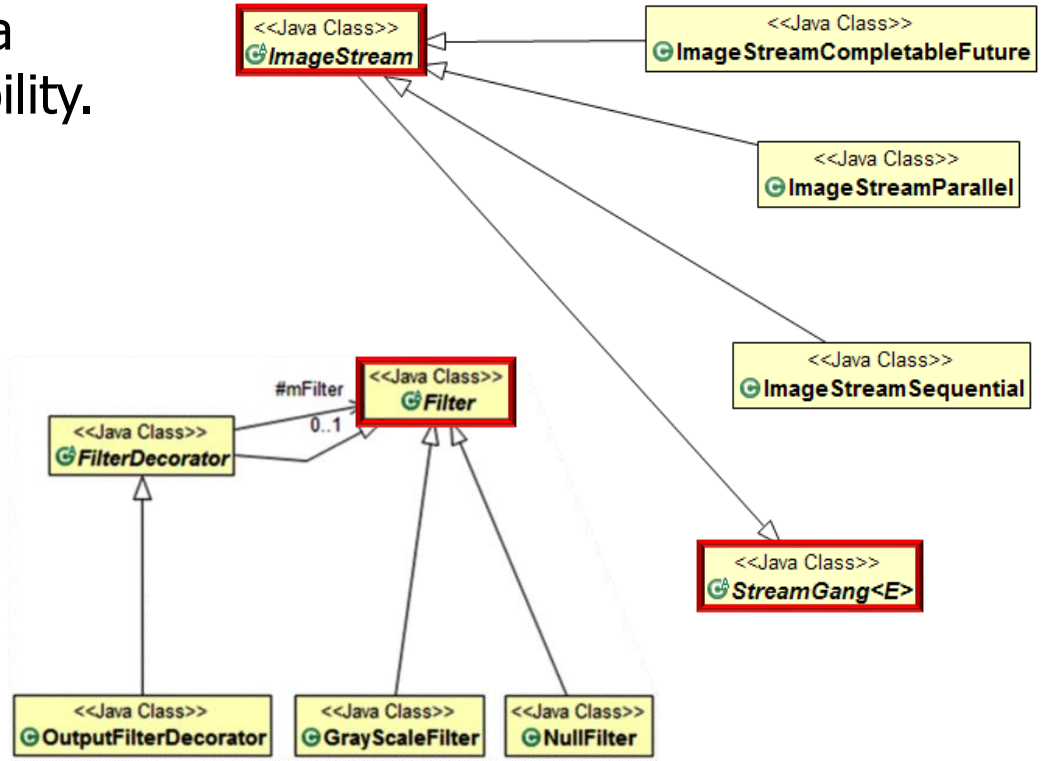
- e.g., consider the ImageStreamGang program.




See github.com/douglascraigschmidt/LiveLessons/tree/master/ImageStreamGang

# Combining Object-Oriented & Functional Programming in Java

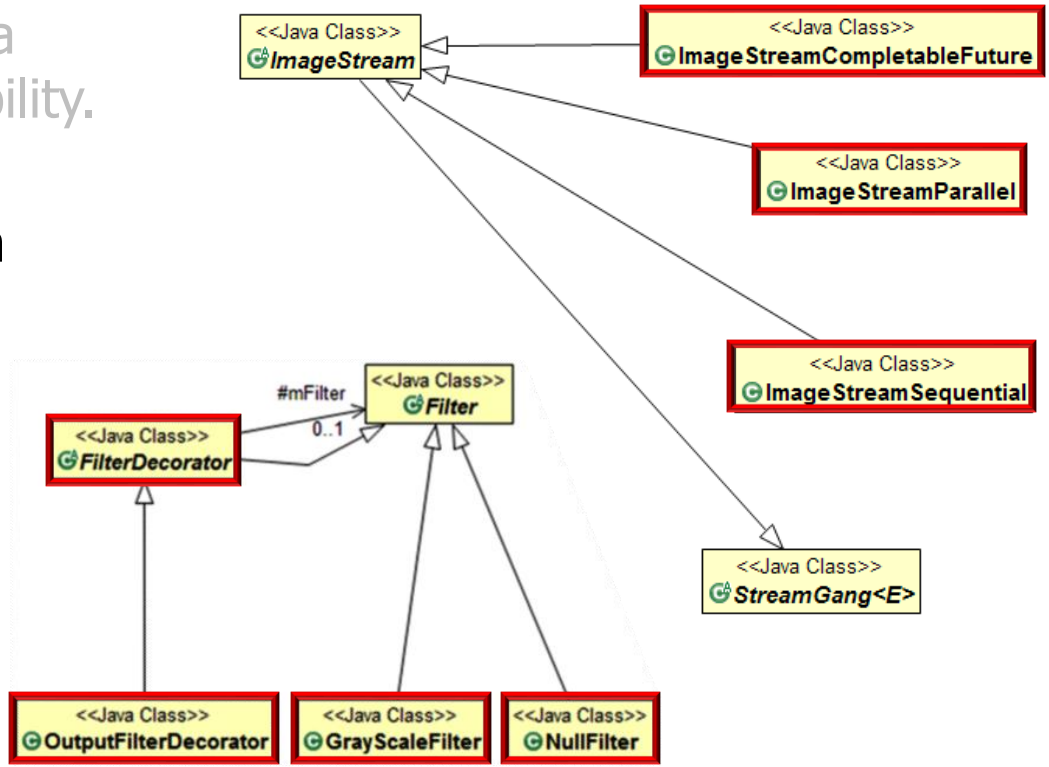
- e.g., consider the ImageStreamGang program.
  - Common super classes provide a reusable foundation for extensibility.




See www.dre.vanderbilt.edu/~schmidt/PDF/Commonality_Variability.pdf

# Combining Object-Oriented & Functional Programming in Java

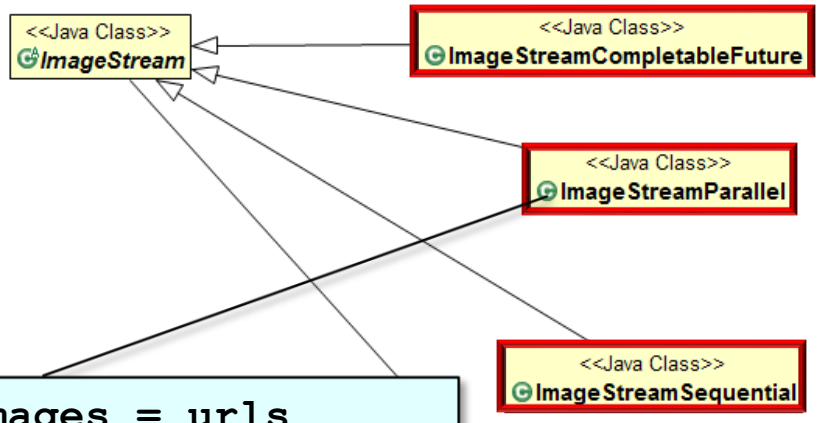
- e.g., consider the ImageStreamGang program.
  - Common super classes provide a reusable foundation for extensibility.
  - Subclasses extend the common classes to create various custom implementation strategies.




See www.dre.vanderbilt.edu/~schmidt/PDF/Commonality_Variability.pdf

# Combining Object-Oriented & Functional Programming in Java

- e.g., consider the ImageStreamGang program.
  - Common super classes provide a reusable foundation for extensibility.
  - Subclasses extend the common classes to create various custom implementation strategies.
- Java's FP features are most effective when used to simplify computations within the context of an OO software architecture.

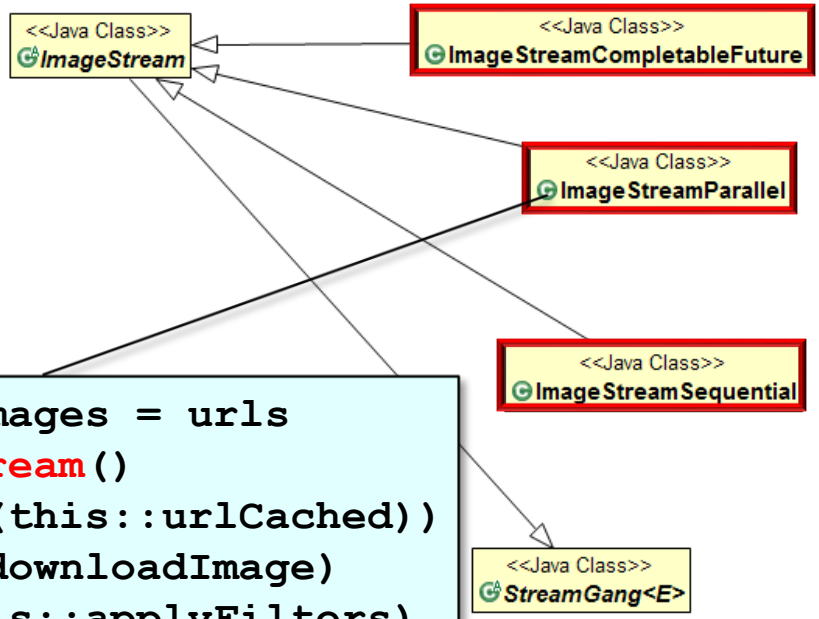



```
List<Image> images = urls
   .parallelStream()
   .filter(not(this::urlCached))
   .map(this::downloadImage)
   .flatMap(this::applyFilters)
   .collect(toList());
```

See www.drdobbs.com/jvm/lambda-expressions-in-java-8/240166764

# Combining Object-Oriented & Functional Programming in Java

- e.g., consider the ImageStreamGang program.
  - Common super classes provide a reusable foundation for extensibility.
  - Subclasses extend the common classes to create various custom implementation strategies.
  - Java's FP features are most effective when used to simplify computations within the context of an OO software architecture.
    - Especially concurrent & parallel computations.

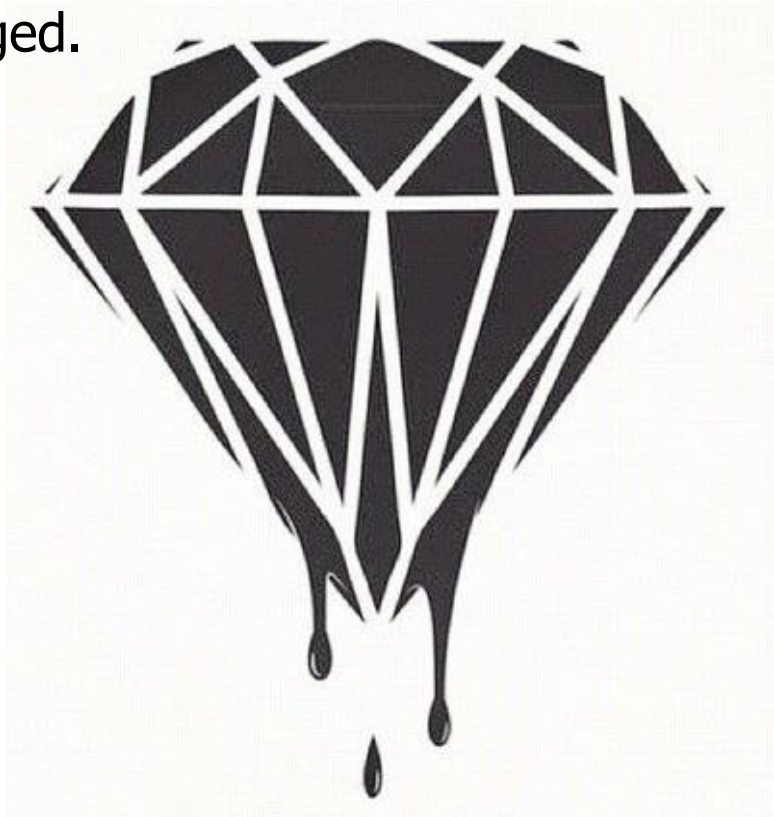



```
List<Image> images = urls
   .parallelStream()
   .filter(not(this::urlCached))
   .map(this::downloadImage)
   .flatMap(this::applyFilters)
   .collect(toList());
```

See docs.oracle.com/javase/tutorial/collections/streams/parallelism.html

# Combining Object-Oriented & Functional Programming in Java

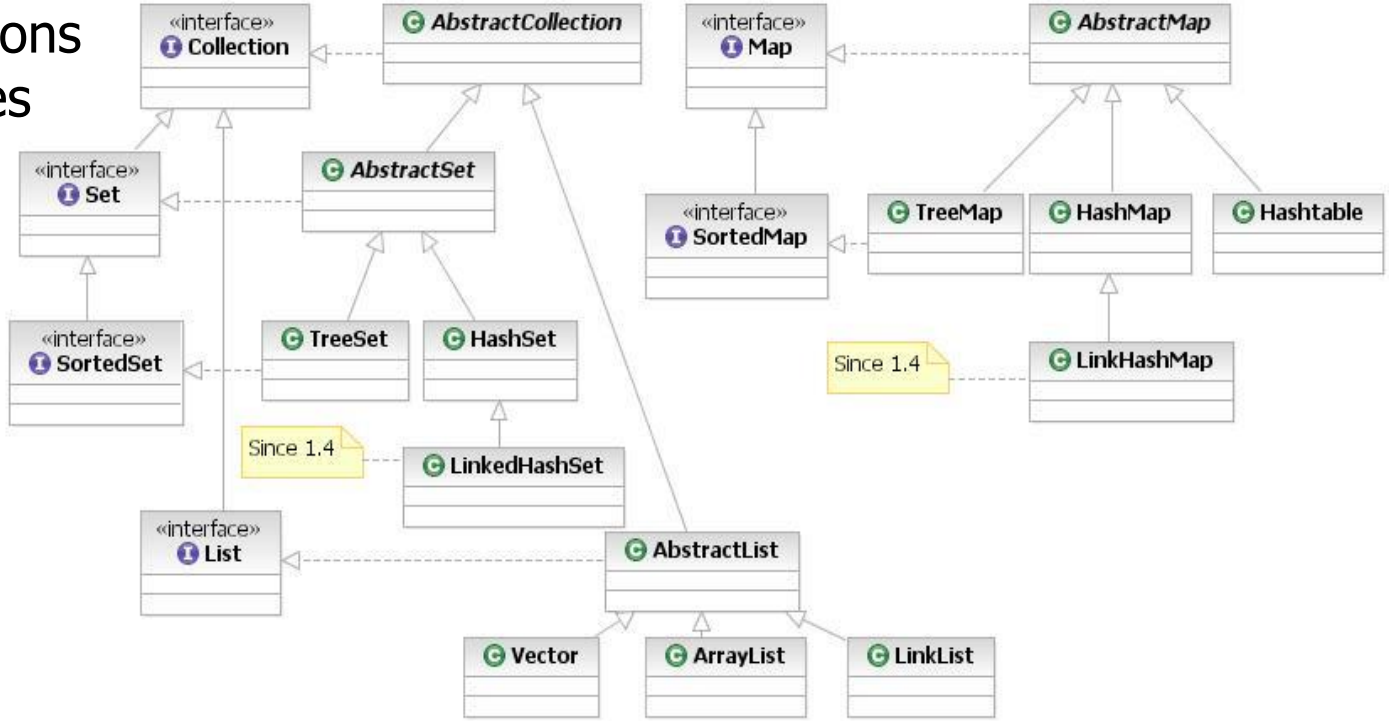
- Since Java is a hybrid language, there are situations in which mutable changes to shared state are allowed/encouraged.



See www.infoq.com/articles/How-Functional-is-Java-8

# Combining Object-Oriented & Functional Programming in Java

- Since Java is a hybrid language, there are situations in which mutable changes to shared state are allowed/encouraged.
  - e.g., Java collections framework classes




See docs.oracle.com/javase/8/docs/technotes/guides/collections

# Combining Object-Oriented & Functional Programming in Java

- However, you're usually better off by minimizing/avoiding the use of shared mutable state in *your* programs!