Overview of Java 8 Foundations

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

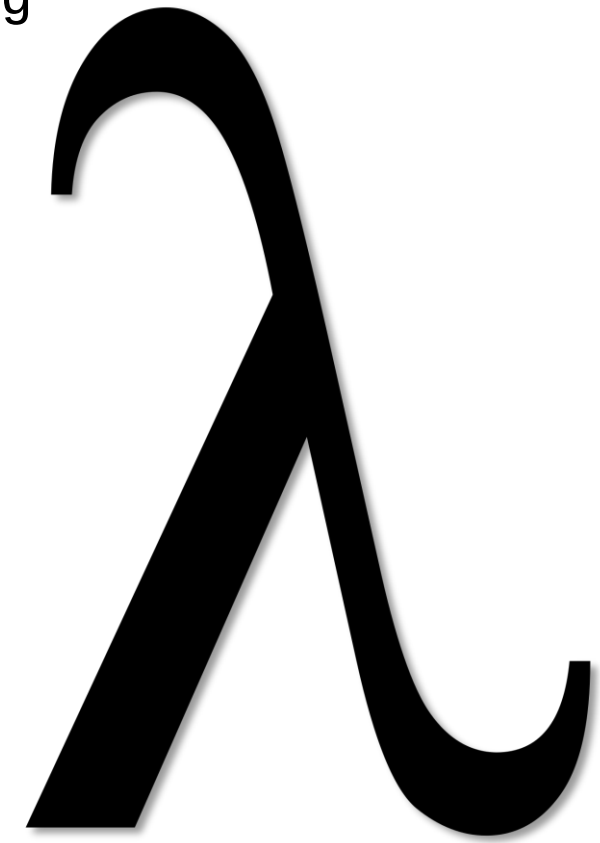
Professor of Computer Science

Institute for Software
Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

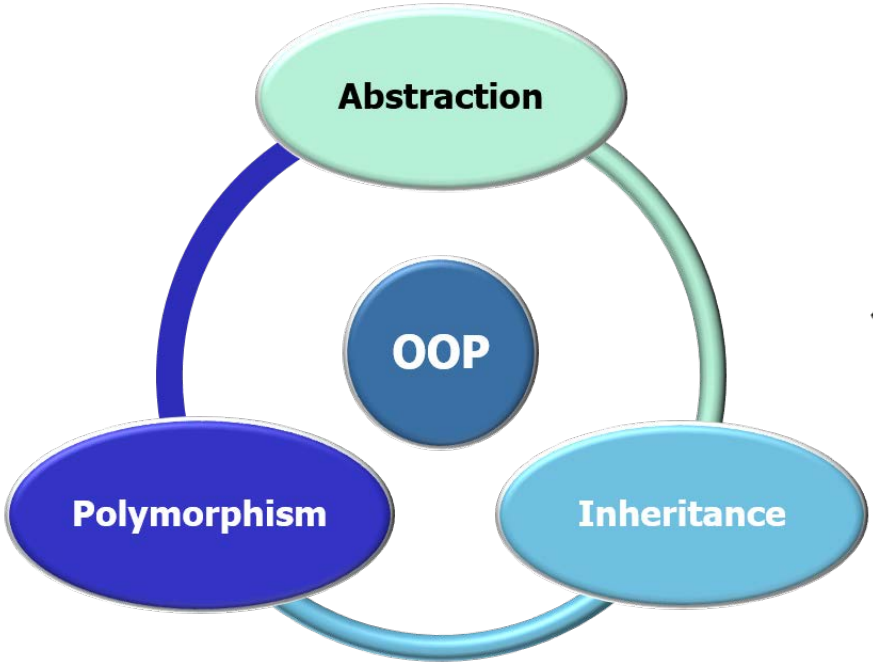
Learning Objectives in this Lesson

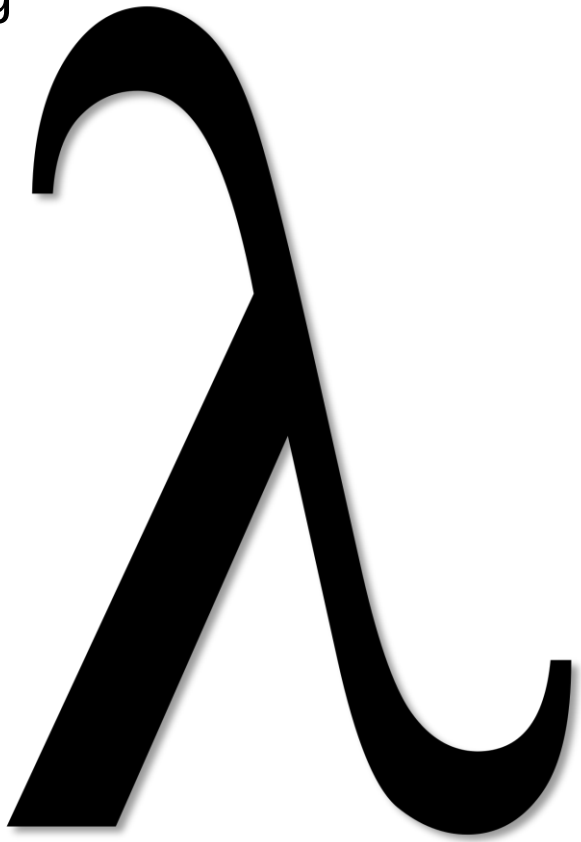
- Understand key aspects of functional programming



Learning Objectives in this Lesson

- Understand key aspects of functional programming
  - Contrasted with object-oriented programming





We'll show some Java 8 code fragments that will be covered in more detail later

Learning Objectives in this Lesson

- Understand key aspects of functional programming
- Recognize the benefits of applying functional programming in Java 8

Learning Objectives in this Lesson

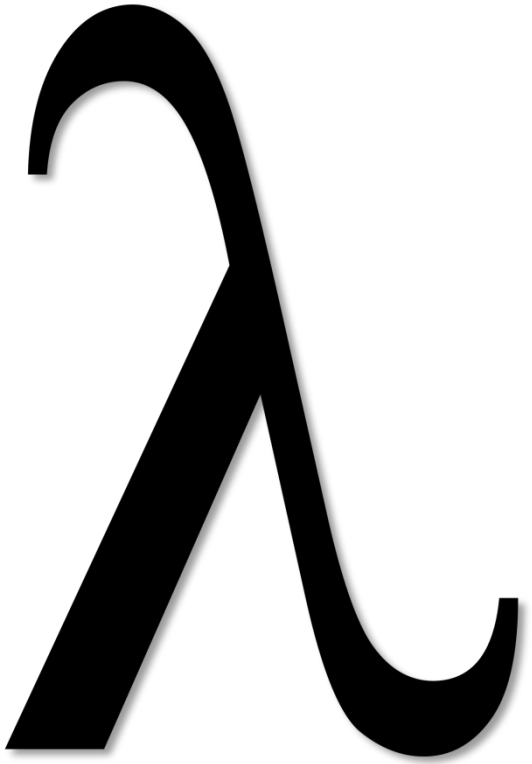
- Understand key aspects of functional programming
- Recognize the benefits of applying functional programming in Java 8
  - Especially when used in conjunction with object-oriented programming

Again, we'll show Java 8 code fragments that'll be covered in more detail later

Overview of Functional Programming in Java 8

Overview of Functional Programming in Java 8

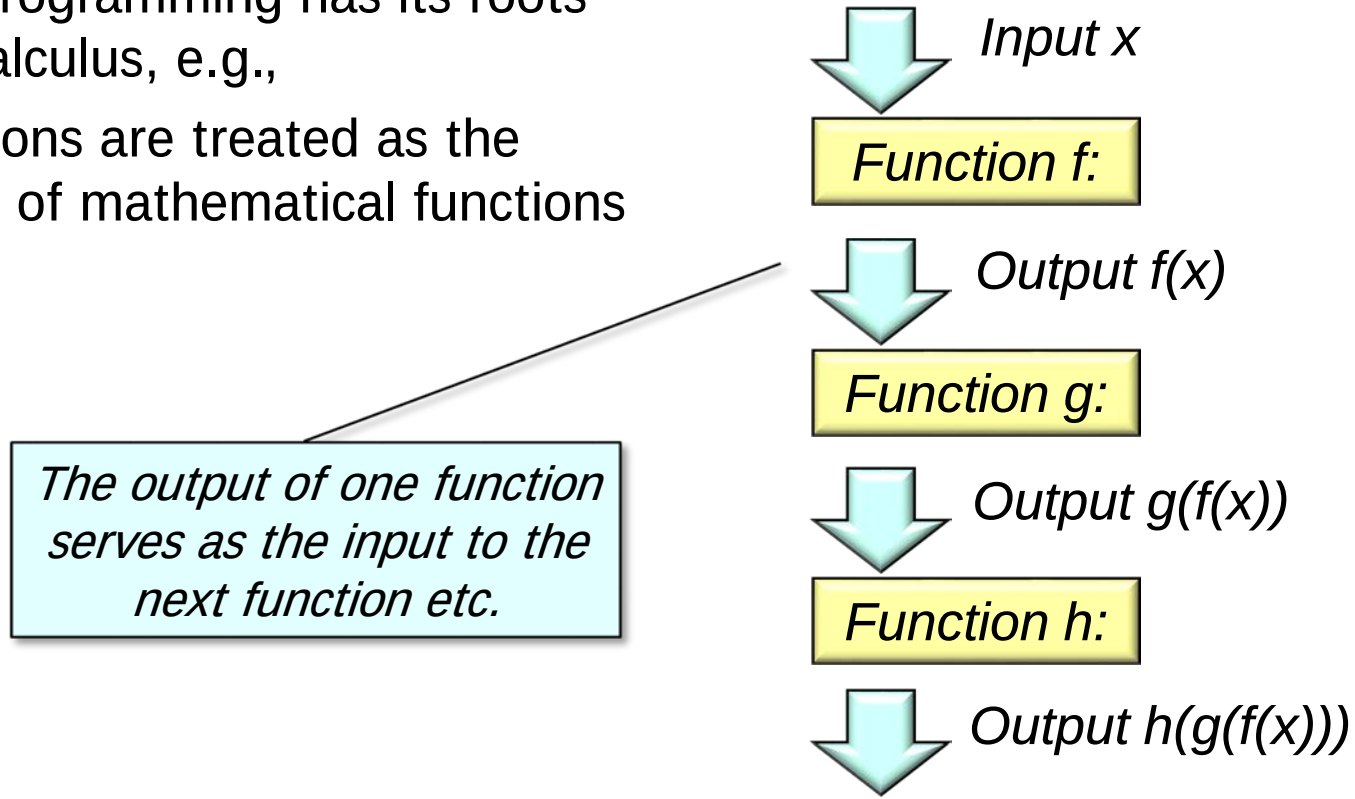
- Functional programming has its roots in lambda calculus



See en.wikipedia.org/wiki/Functional_programming

Overview of Functional Programming in Java 8

- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions

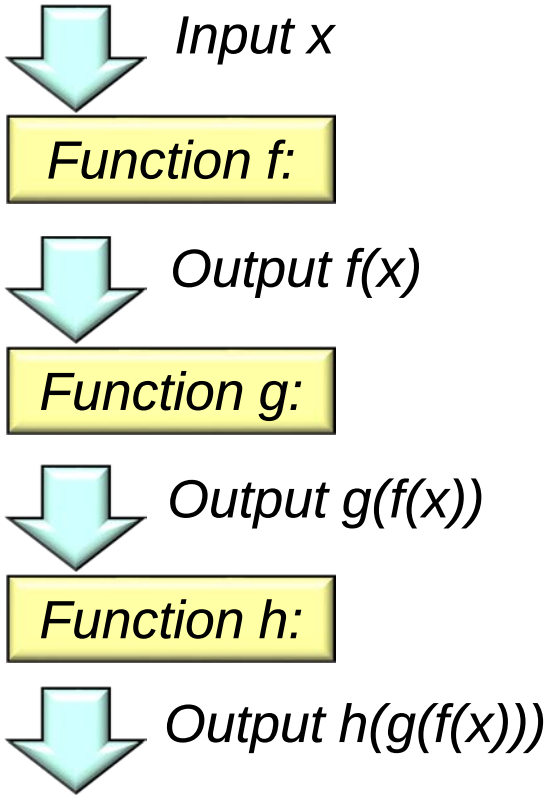


See en.wikipedia.org/wiki/Functional_programming#Pure_functions

Overview of Functional Programming in Java 8

- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions

```
long parallelFactorial(long n) {
  return LongStream
    .rangeClosed(1, n)
    .parallel()
    .reduce(1, (a, b) -> a * b);
}
```



See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex16

Overview of Functional Programming in Java 8

- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions
  - Changing state & mutable data are discouraged/avoided

See en.wikipedia.org/wiki/Side_effect_(computer_science)

Overview of Functional Programming in Java 8

- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions
  - Changing state & mutable data are discouraged/avoided

```
long parallelFactorial(long n) {
  Total t = new Total();
  LongStream.rangeClosed(1, n)
            .parallel()
            .forEach(t::mult);
  return t.mTotal;
}
```

```
class Total {
  public long mTotal = 1;

  public void mult(long n)
  { mTotal *= n; }
}
```

Beware of race conditions!!!

See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex16

Overview of Functional Programming in Java 8

- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions
  - Changing state & mutable data are discouraged/avoided

```
long parallelFactorial(long n) {
  Total t = new Total();
  LongStream.rangeClosed(1, n)
            .parallel()
            .forEach(t::mult);
  return t.mTotal;
}
```

```
class Total {
  public long mTotal = 1;

  public void mult(long n)
  { mTotal *= n; }
}
```

Only you can prevent race conditions!

In Java *you* must avoid race conditions, i.e., the compiler & JVM won't save you..

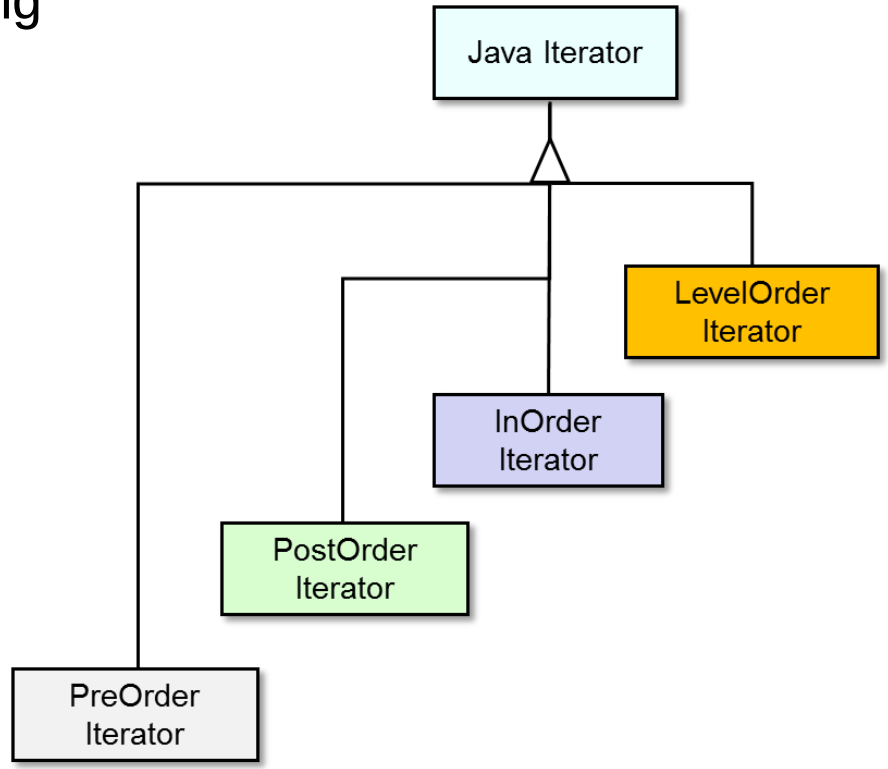
Overview of Functional Programming in Java 8

- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions
  - Changing state & mutable data are discouraged/avoided
  - Instead, the focus is on “immutable” objects
    - i.e., objects whose state cannot change after they are constructed

See docs.oracle.com/javase/tutorial/essential/concurrency/immutable.html

Overview of Functional Programming in Java 8

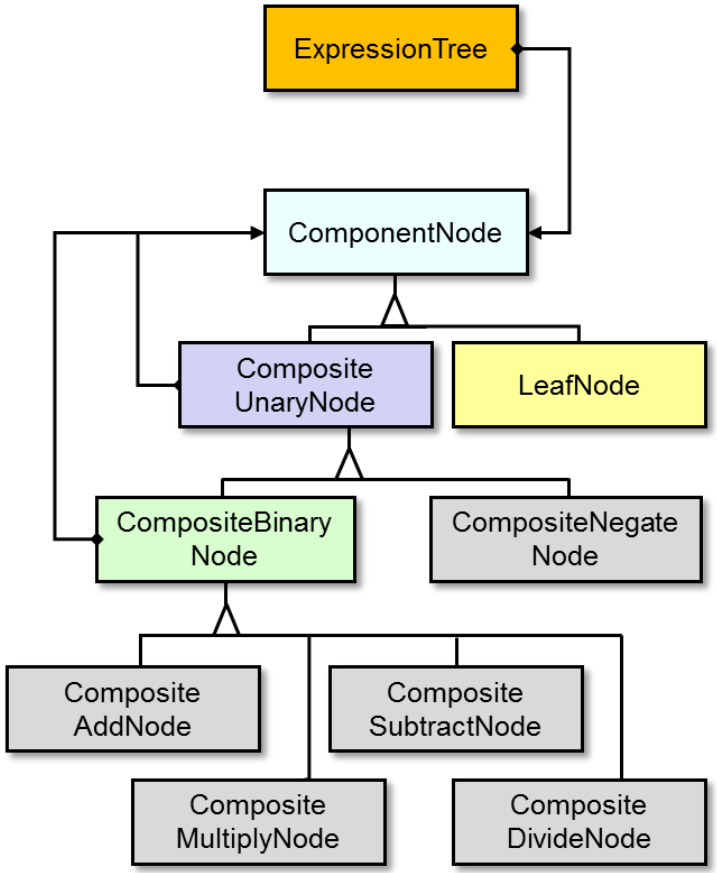
- In contrast, object-oriented programming employs “hierarchical data abstraction”



See en.wikipedia.org/wiki/Object-oriented_design

Overview of Functional Programming in Java 8

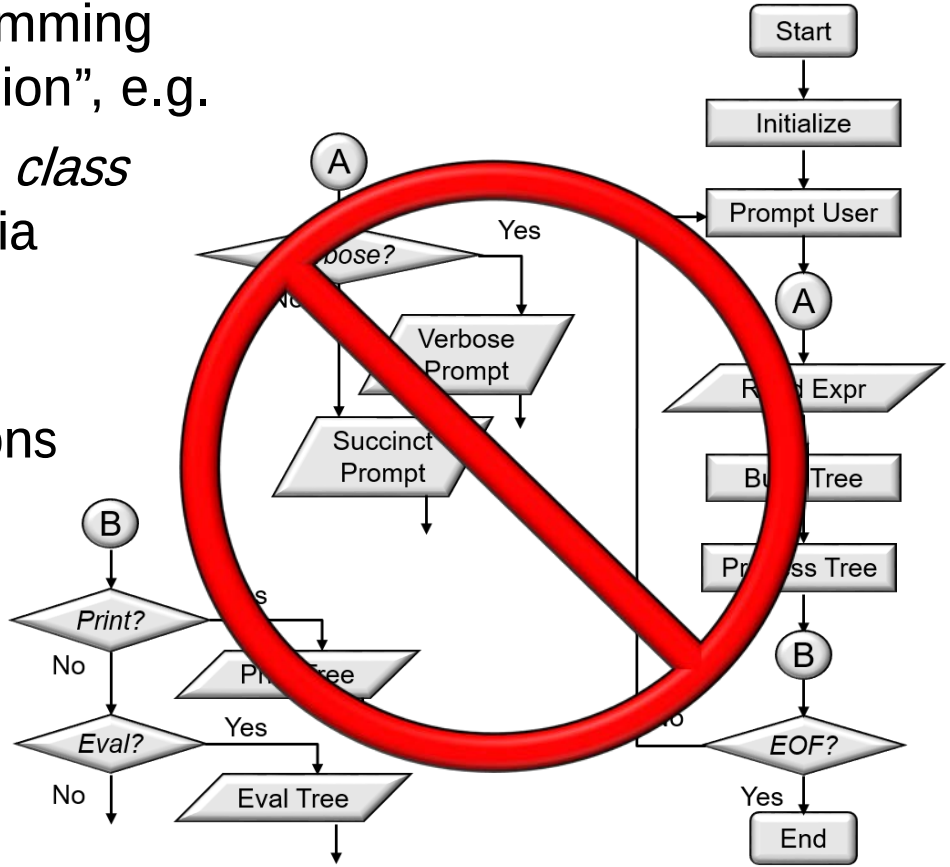
- In contrast, object-oriented programming employs “hierarchical data abstraction”, e.g.
  - Components are based on stable *class* roles & relationships extensible via inheritance & dynamic binding



See en.wikipedia.org/wiki/Object-oriented_programming

Overview of Functional Programming in Java 8

- In contrast, object-oriented programming employs “hierarchical data abstraction”, e.g.
  - Components are based on stable *class* roles & relationships extensible via inheritance & dynamic binding
    - Rather than by functions that correspond to algorithmic actions



See www.drdobbs.com/windows/software-complexity-bringing-order-to-ch/199901062

Overview of Functional Programming in Java 8

- In contrast, object-oriented programming employs “hierarchical data abstraction”, e.g.
  - Components are based on stable *class* roles & relationships extensible via inheritance & dynamic binding
  - State is encapsulated by methods that perform imperative statements

```
Tree tree = ...;
Visitor printVisitor =
   makeVisitor(...);

for(Iterator<Tree> iter =
       tree.iterator();
    iter.hasNext();)
  iter.next()
    .accept(printVisitor);
```

See en.wikipedia.org/wiki/Imperative_programming

Overview of Functional Programming in Java 8

- In contrast, object-oriented programming employs “hierarchical data abstraction”, e.g.
  - Components are based on stable *class* roles & relationships extensible via inheritance & dynamic binding
  - State is encapsulated by methods that perform imperative statements
    - This state is often mutable

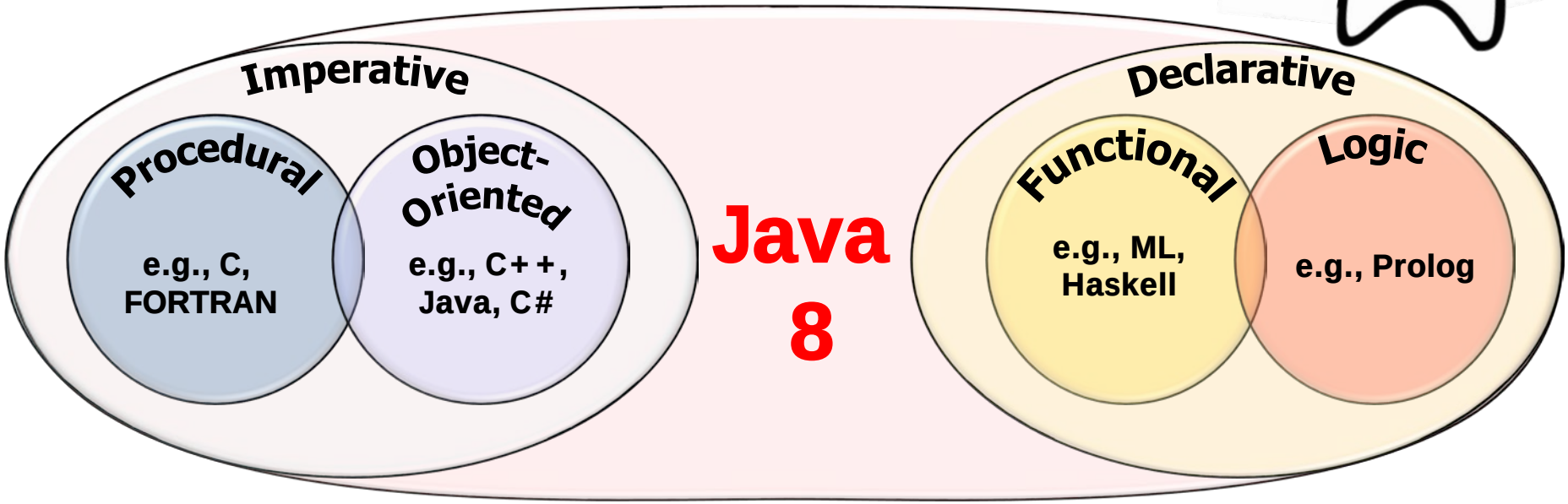
```
Tree tree = ...;
Visitor printVisitor =
   makeVisitor(...);

for(Iterator<Tree> iter =
       tree.iterator();
    iter.hasNext();)
  iter.next()
    .accept(printVisitor);
```

See en.wikipedia.org/wiki/Imperative_programming

Combining Object-Oriented (OO) & Functional Programming (FP) in Java 8

Benefits of Combining OO & FP in Java 8

- Java 8's combination of functional & object-oriented paradigms is powerful!



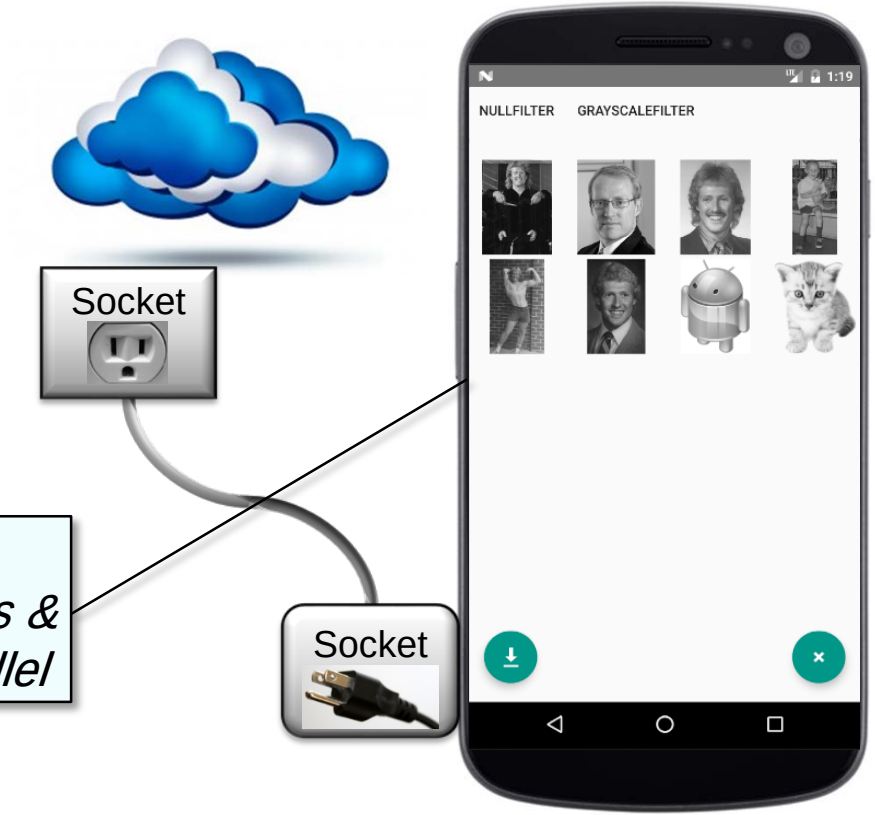
Benefits of Combining OO & FP in Java 8

- Java 8's functional features help close the gap between a program's "domain intent" & its computations

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

Benefits of Combining OO & FP in Java 8

- Java 8's functional features help close the gap between a program's "domain intent" & its computations, e.g.,
  - Domain intent defines "what"



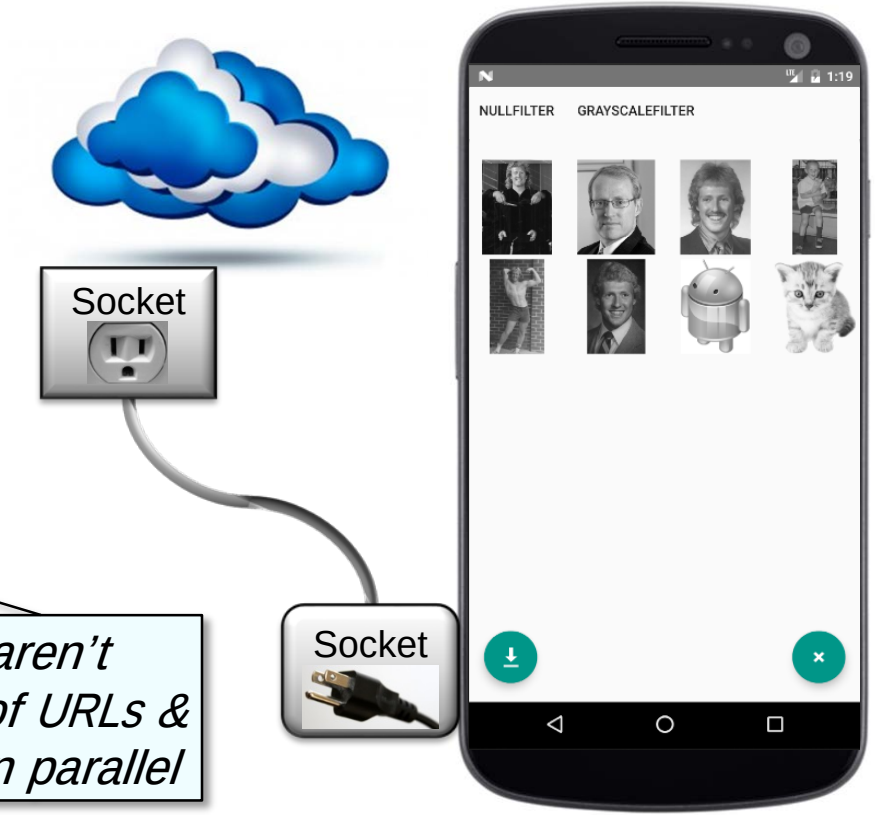
Download images that aren't already cached from a list of URLs & process/store the images in parallel

Benefits of Combining OO & FP in Java 8

- Java 8'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(urlCached()))
   .map(this::downloadImage)
   .flatMap(this::applyFilters)
   .collect(toList());
```

Download images that aren't already cached from a list of URLs & process/store the images in parallel

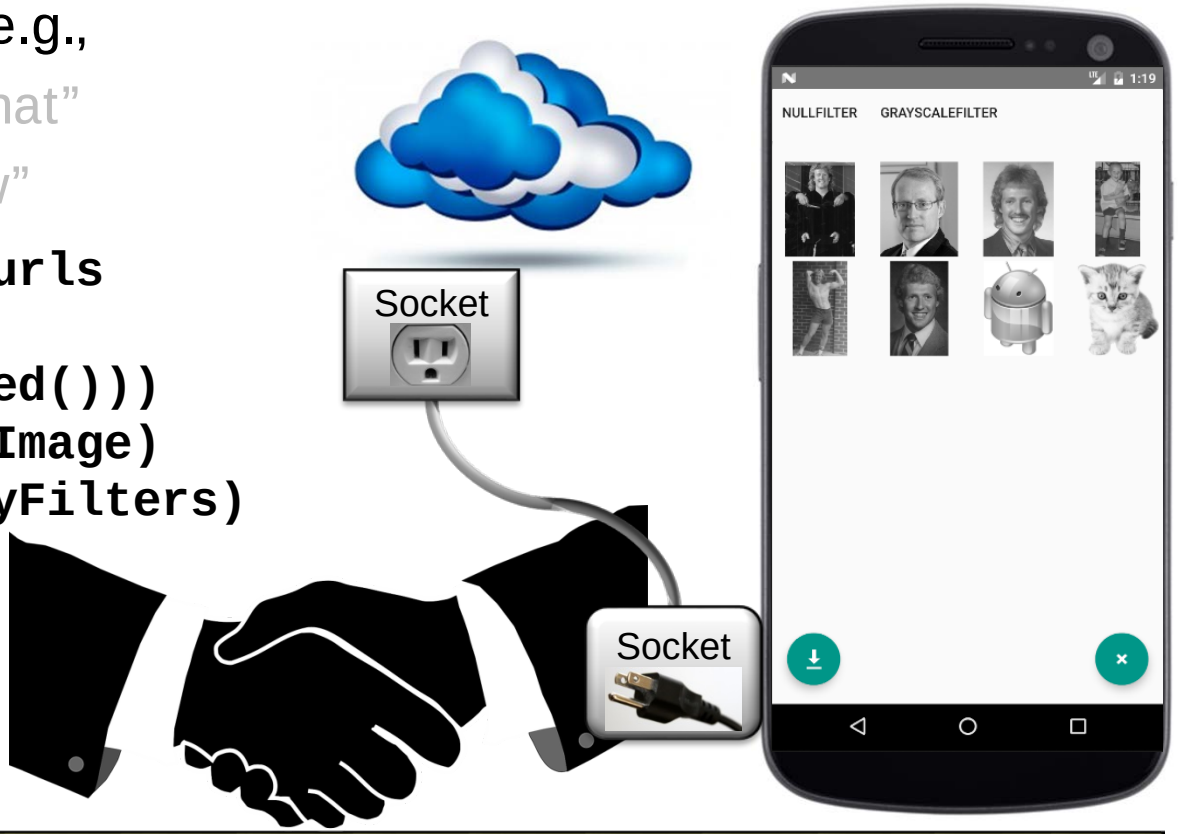


See github.com/douglascraigschmidt/LiveLessons/tree/master/ImageStreamGang

Benefits of Combining OO & FP in Java 8

- Java 8's functional features help close the gap between a program's "domain intent" & its computations, e.g.,
  - Domain intent defines "what"
  - Computations define "how"

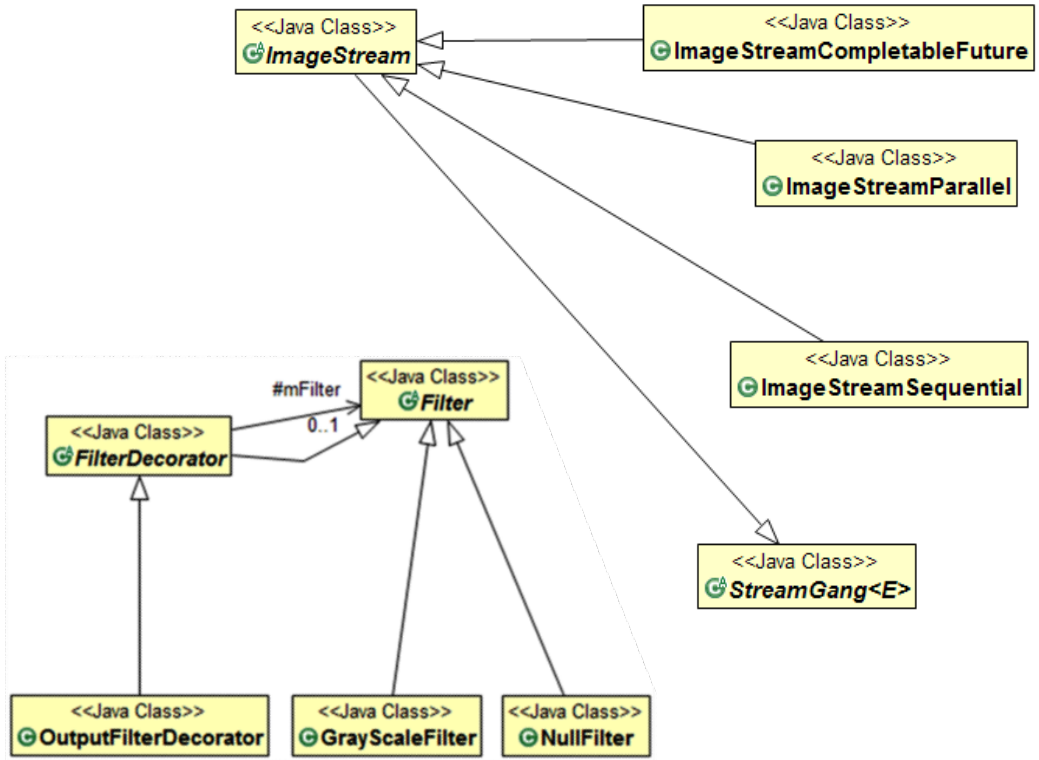
```
List<Image> images = urls
   .parallelStream()
   .filter(not(urlCached()))
   .map(this::downloadImage)
   .flatMap(this::applyFilters)
   .collect(toList());
```



Java 8 functional programming features connect domain intent & computations

Benefits of Combining OO & FP in Java 8

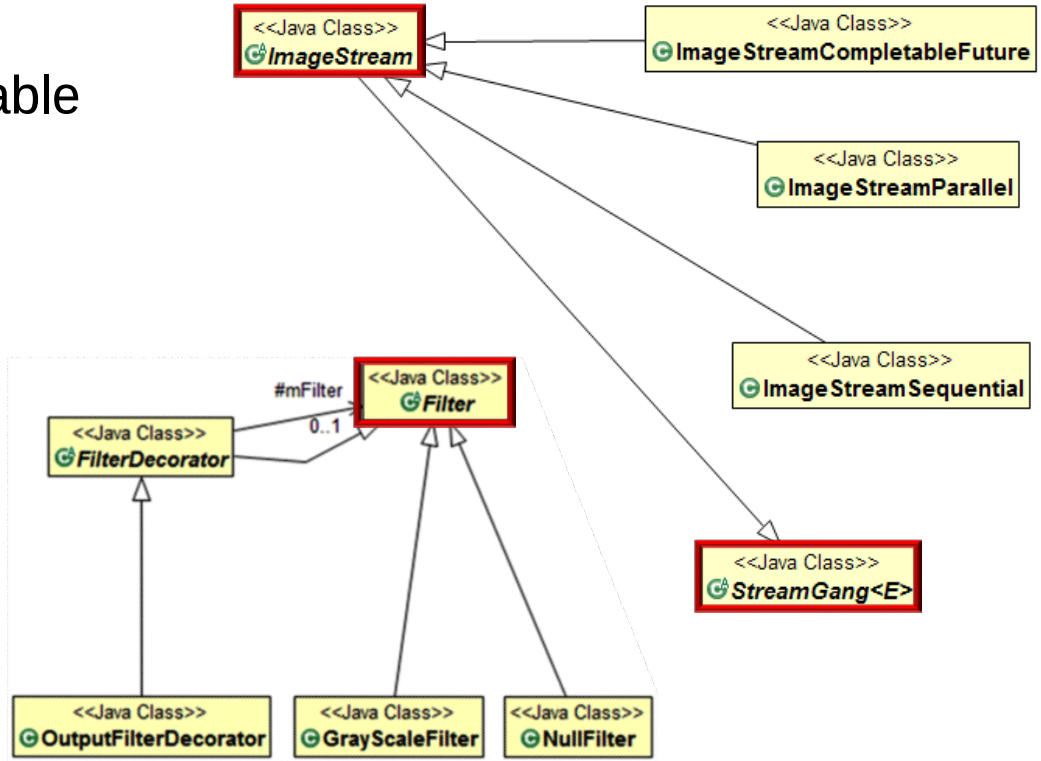
- Likewise, Java 8's object-oriented features help to structure a program's software architecture



See en.wikipedia.org/wiki/Software_architecture

Benefits of Combining OO & FP in Java 8

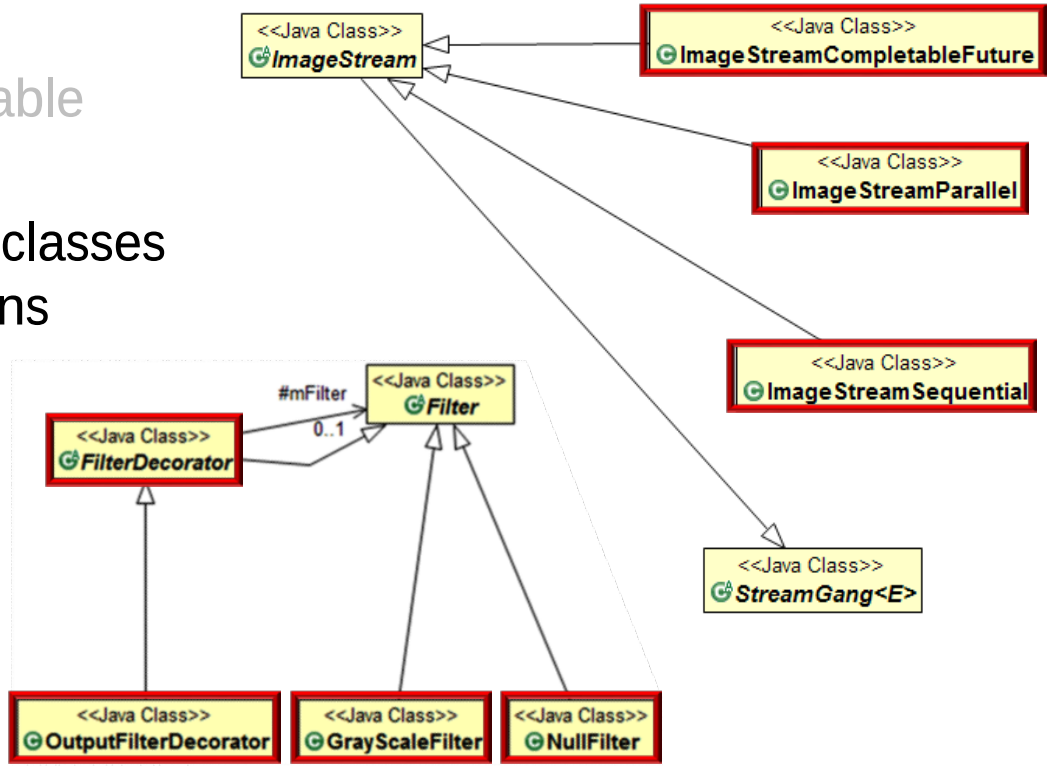
- Likewise, Java 8's object-oriented features help to structure a program's software architecture, e.g.,
  - Common classes provide a reusable foundation for extensibility



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

Benefits of Combining OO & FP in Java 8

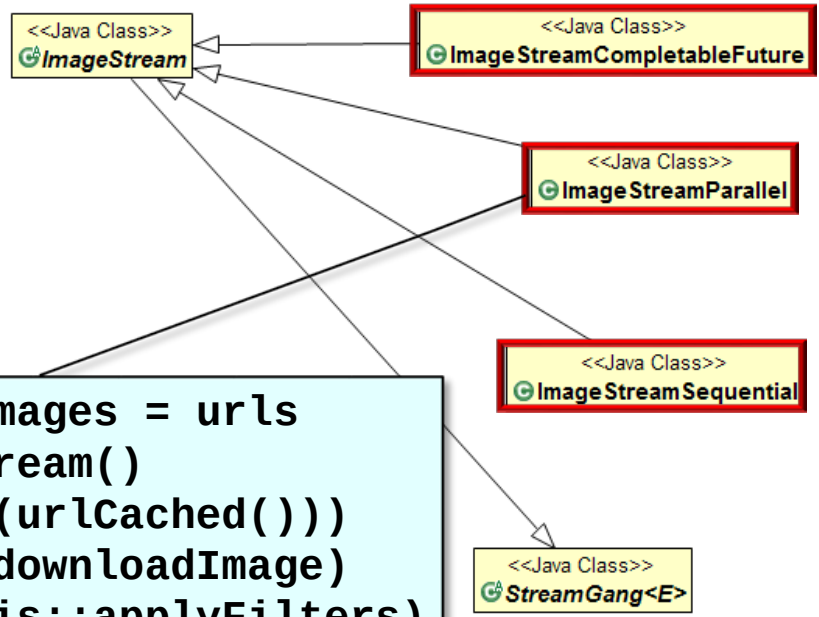
- Likewise, Java 8's object-oriented features help to structure a program's software architecture, e.g.,
  - Common classes provide a reusable foundation for extensibility
  - Subclasses extend the common classes to create various custom solutions



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

Benefits of Combining OO & FP in Java 8

- Likewise, Java 8's object-oriented features help to structure a program's software architecture, e.g.,
  - Common classes provide a reusable foundation for extensibility
  - Subclasses extend the common classes to create various custom solutions
- Java 8's FP features are most effective when used to simplify computations within the context of an OO software architecture

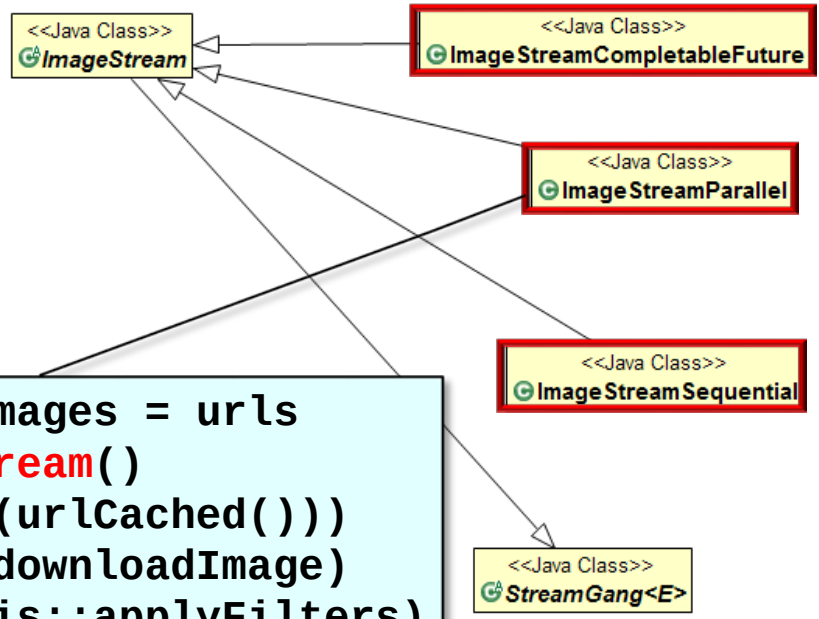



```
List<Image> images = urls
  .parallelStream()
  .filter(not(urlCached()))
  .map(this::downloadImage)
  .flatMap(this::applyFilters)
  .collect(toList());
```

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

Benefits of Combining OO & FP in Java 8

- Likewise, Java 8's object-oriented features help to structure a program's software architecture, e.g.,
  - Common classes provide a reusable foundation for extensibility
  - Subclasses extend the common classes to create various custom solutions
- Java 8'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(urlCached()))
  .map(this::downloadImage)
  .flatMap(this::applyFilters)
  .collect(toList());
```

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

End of Overview of Java 8 Foundations