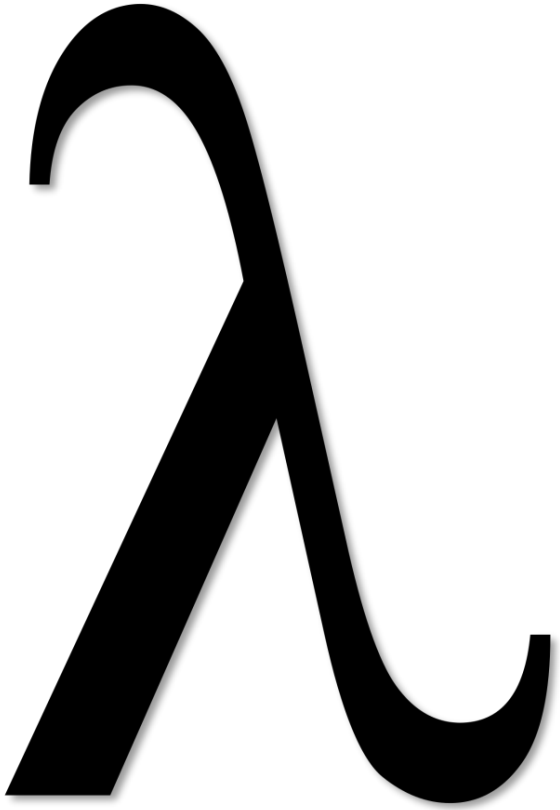
Overview of Java

Key Functional Programming Concepts & Features

Douglas C. Schmidt

Learning Objectives in This Lesson

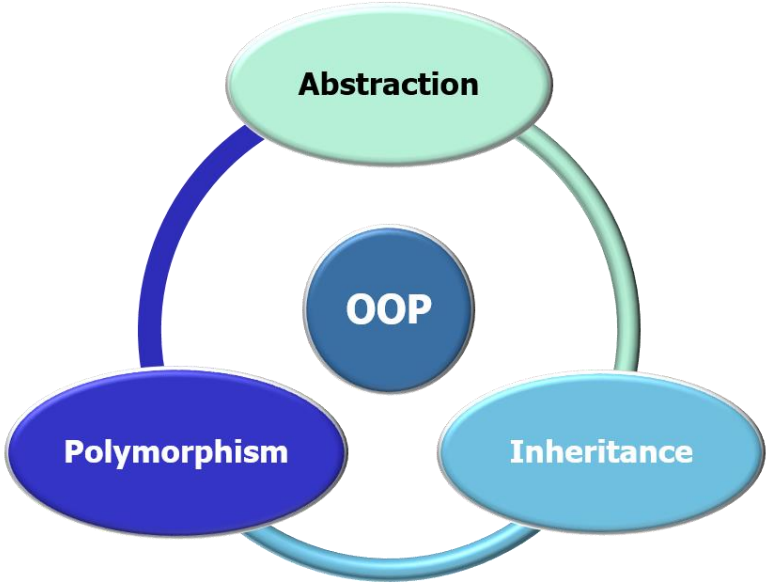
- Understand key functional programming concepts & features supported by Java.

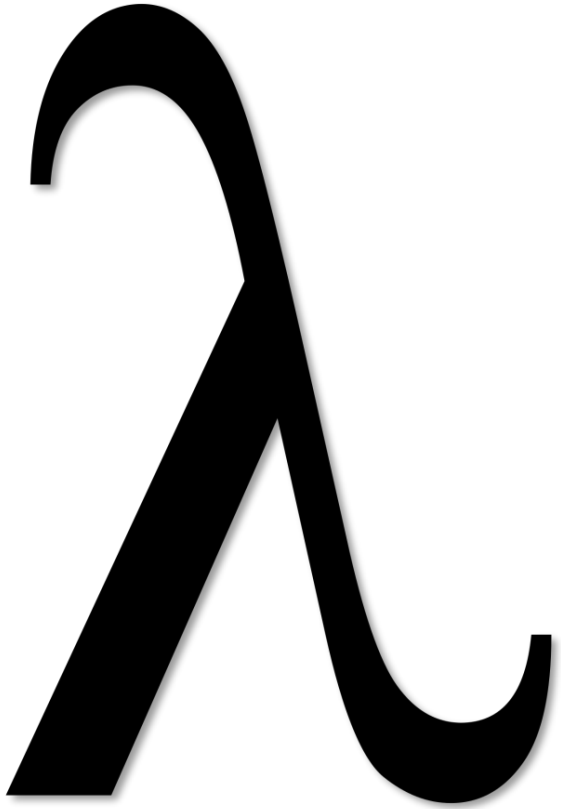


These functional programming features were added in Java 8 & expanded later.

Learning Objectives in This Lesson

- Understand key functional programming concepts & features supported by Java.
- Know how to compare & contrast functional programming & object-oriented programming.



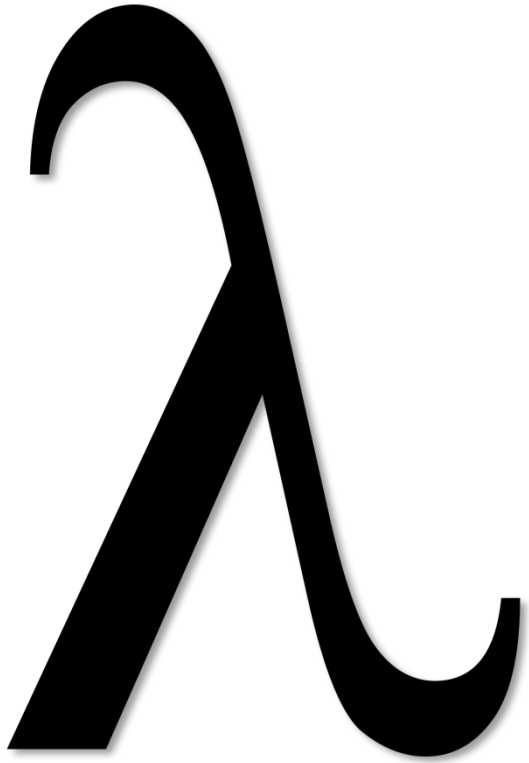


Douglas C. Schmidt

Key Functional Programming Concepts in Java

Key Functional Programming Concepts in Java

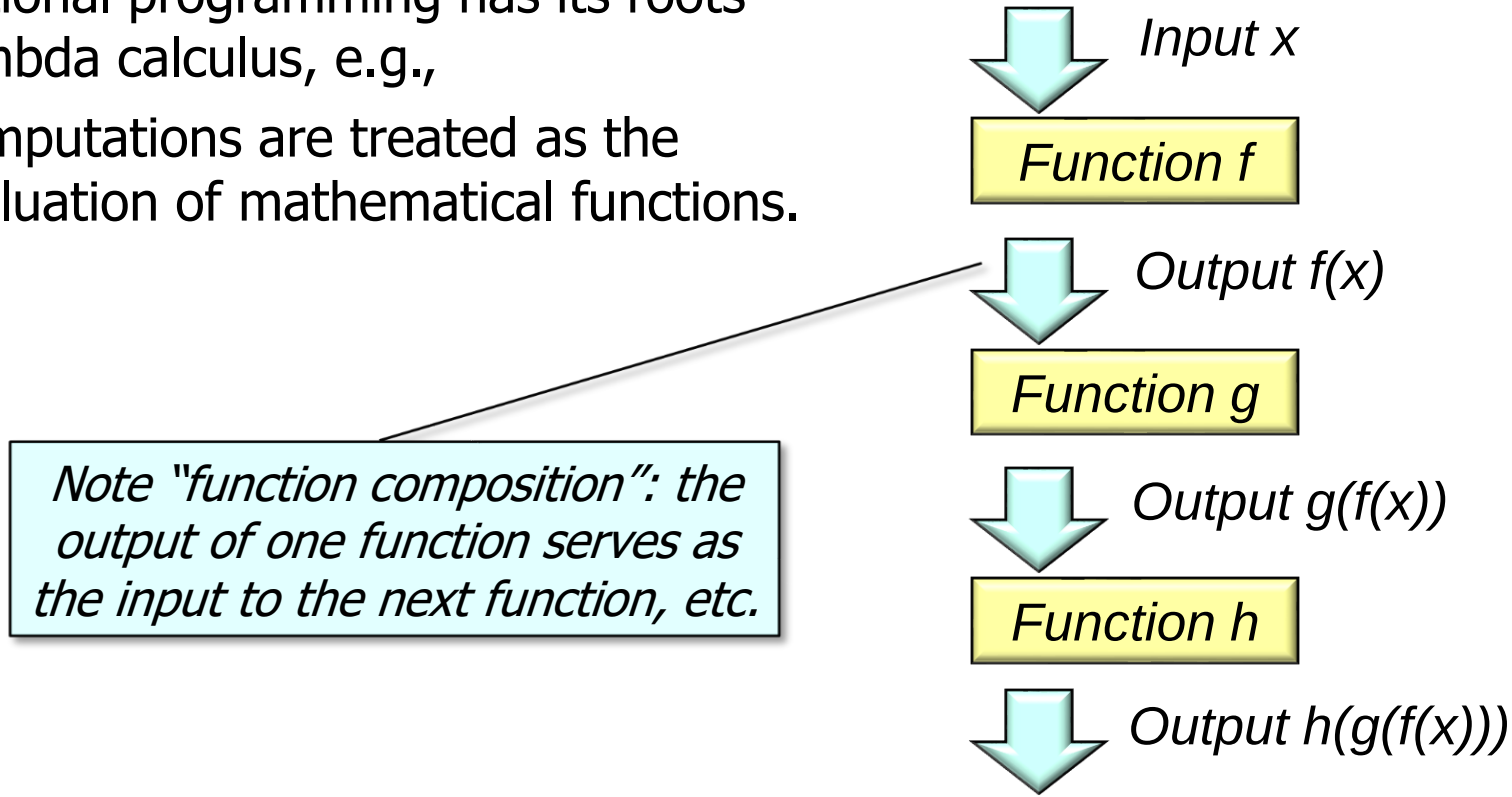
- Functional programming has its roots in lambda calculus



See en.wikipedia.org/wiki/Functional_programming

Key Functional Programming Concepts in Java

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



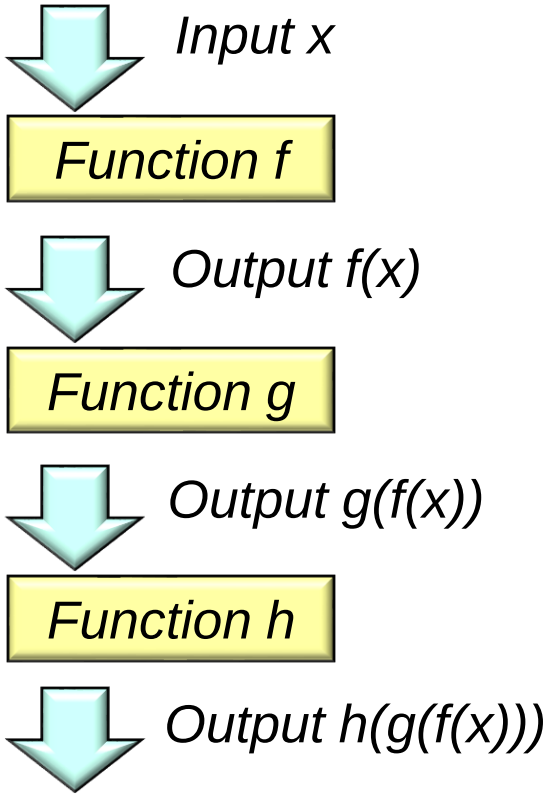
See en.wikipedia.org/wiki/Functional_programming#Pure_functions

Key Functional Programming Concepts in Java

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

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

Compute the 'nth' factorial in parallel



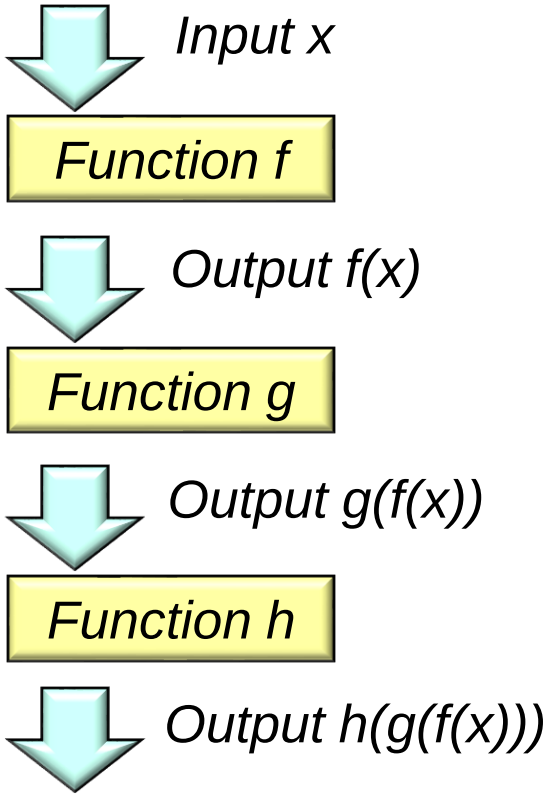
See github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex16

Key Functional Programming Concepts in Java

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

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

Create a stream of values from 1 to n.



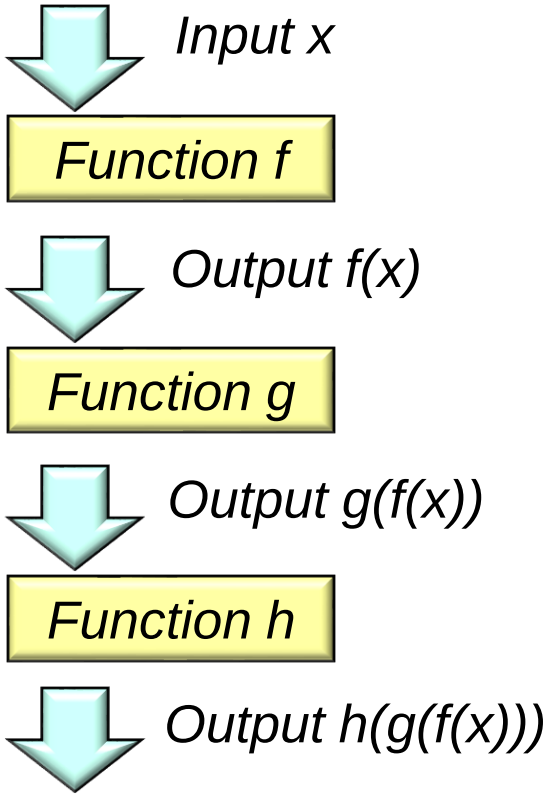
See www.baeldung.com/java-8-streams

Key Functional Programming Concepts in Java

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

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

Multiply each pair of values in the stream in parallel to make a single "reduced" result.



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

Key Functional Programming Concepts in Java

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

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

Key Functional Programming Concepts in Java

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

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

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

Shared mutable state.

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

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

Key Functional Programming Concepts in Java

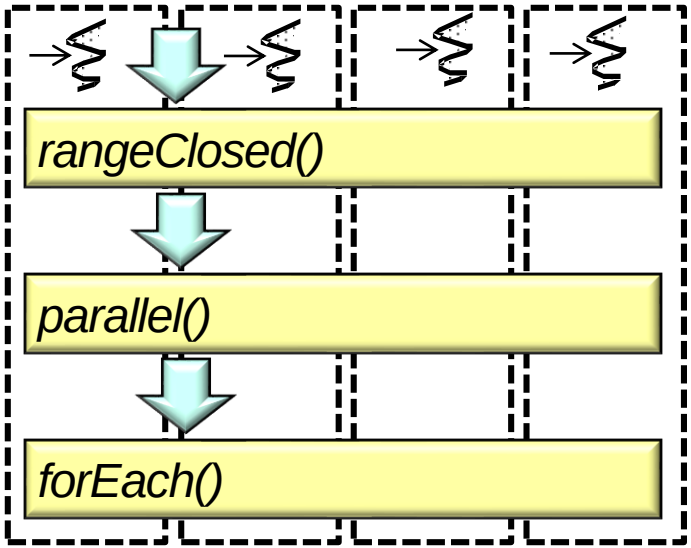
- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions.
  - Changing state & mutable shared data are discouraged to avoid various hazards.

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

Run in parallel.

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

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



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

Key Functional Programming Concepts in Java

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

```
long factorial(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 en.wikipedia.org/wiki/Race_condition#Software

Key Functional Programming Concepts in Java

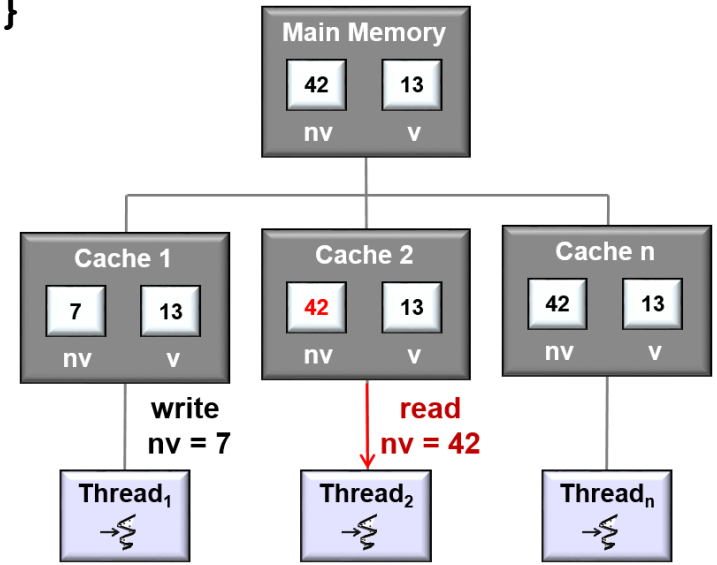
- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions.
  - Changing state & mutable shared data are discouraged to avoid various hazards.

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

Beware of inconsistent memory visibility.

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

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



See jeremymanson.blogspot.com/2007/08/atomicity-visibility-and-ordering.html

Key Functional Programming Concepts in Java

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

```
long factorial(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 concurrency hazards!

In Java, *you* must avoid these hazards, i.e., the compiler & JVM won't save you from yourself..

Key Functional Programming Concepts in Java

- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions.
  - Changing state & mutable shared data are discouraged to avoid various hazards.
- Instead, the focus is on "immutable" objects.
  - The state of these objects cannot change after they are constructed.

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

Key Functional Programming Concepts in Java

- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions.
  - Changing state & mutable shared data are discouraged to avoid various hazards.
- Instead, the focus is on "immutable" objects.
  - The state of these objects cannot change after they are constructed.

```
final class String {
  private final char value[];
  ...

  public String(String s) {
    value = s;
    ...
  }

  public int length() {
    return value.length;
  }
  ...
}
```

See www.baeldung.com/java-immutable-object

Key Functional Programming Concepts in Java

- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions.
  - Changing state & mutable shared data are discouraged to avoid various hazards.
  - Instead, the focus is on "immutable" objects.
    - The state of these objects cannot change after they are constructed.

```
final class String {
  private final char value[];
  ...

  public String(String s) {
    value = s;
    ...
  }

  public int length() {
    return value.length;
  }
  ...
}
```

See docs.oracle.com/javase/8/docs/api/java/lang/String.html

Key Functional Programming Concepts in Java

- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions.
  - Changing state & mutable shared data are discouraged to avoid various hazards.
- Instead, the focus is on "immutable" objects.
  - The state of these objects cannot change after they are constructed.
    - e.g., final fields and/or only accessor methods

```
final class String {
  private final char value[];
  ...

  public String(String s) {
    value = s;
    ...
  }

  public int length() {
    return value.length;
  }
  ...
}
```

See www.programcreek.com/2013/04/why-string-is-immutable-in-java

Key Functional Programming Concepts in Java

- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions.
  - Changing state & mutable shared data are discouraged to avoid various hazards.
  - Instead, the focus is on "immutable" objects.
  - Functional "behaviors" can be parameterized.

```
List<Thread> threads =
  Arrays.asList(
    new Thread("Larry"),
    new Thread("Curly"),
    new Thread("Moe"));
```

```
threads.sort
 (Comparator.comparing
    (Thread::getName));
```

See blog.indrek.io/articles/java-8-behavior-parameterization

Key Functional Programming Concepts in Java

- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions.
  - Changing state & mutable shared data are discouraged to avoid various hazards.
  - Instead, the focus is on "immutable" objects.
  - Functional "behaviors" can be parameterized.

```
List<Thread> threads =
  Arrays.asList(
    new Thread("Larry"),
    new Thread("Curly"),
    new Thread("Moe"));
```

Create a list of named threads.

```
threads.sort
 (Comparator.comparing
    (Thread::getName));
```

Key Functional Programming Concepts in Java

- Functional programming has its roots in lambda calculus, e.g.,
  - Computations are treated as the evaluation of mathematical functions.
  - Changing state & mutable shared data are discouraged to avoid various hazards.
  - Instead, the focus is on "immutable" objects.
- Functional "behaviors" can be parameterized.

```
List<Thread> threads =
  Arrays.asList(
    new Thread("Larry"),
    new Thread("Curly"),
    new Thread("Moe"));
```

```
threads.sort
 (Comparator.comparing
    (Thread::getName));
```

Sort the threads according to their name.

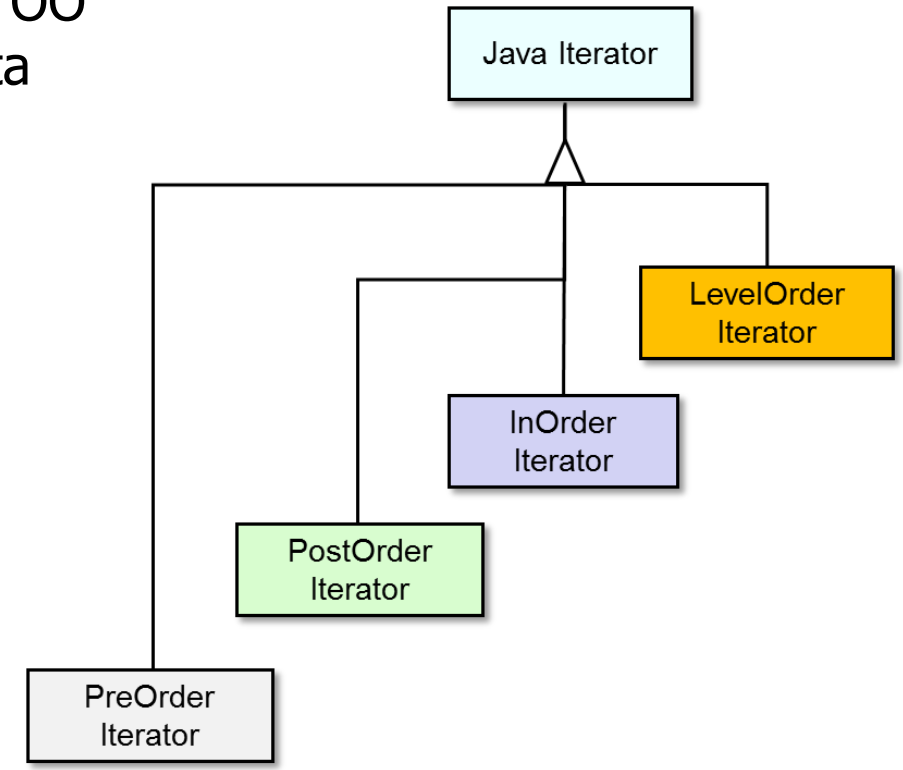
See www.baeldung.com/java-8-sort-lambda

Douglas C. Schmidt

Functional vs. Object-Oriented Programming in Java

Functional vs. Object-Oriented Programming in Java

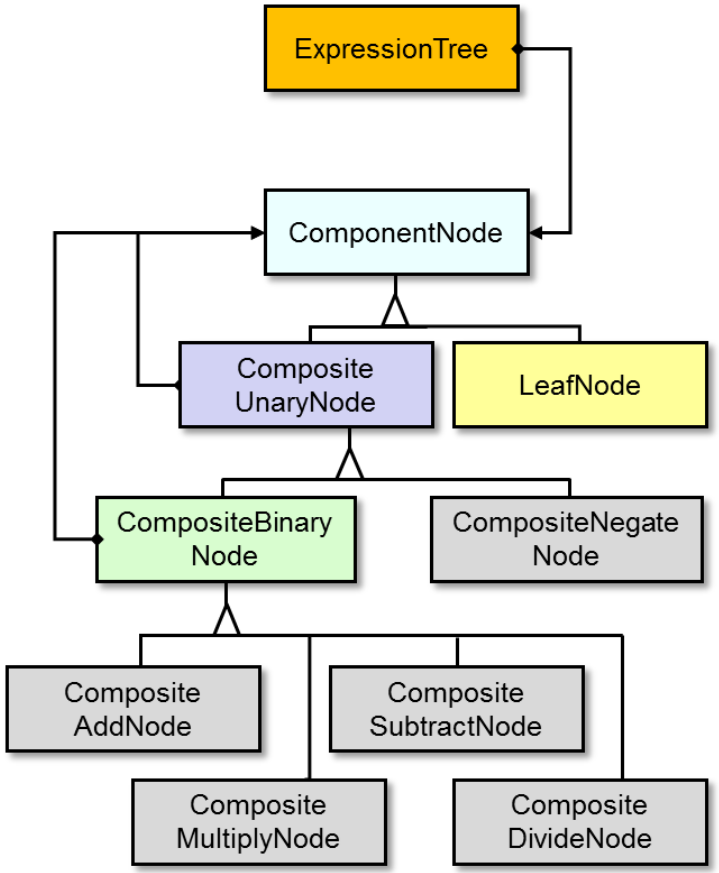
- In contrast to functional programming, OO programming employs "hierarchical data abstraction"



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

Functional vs. Object-Oriented Programming in Java

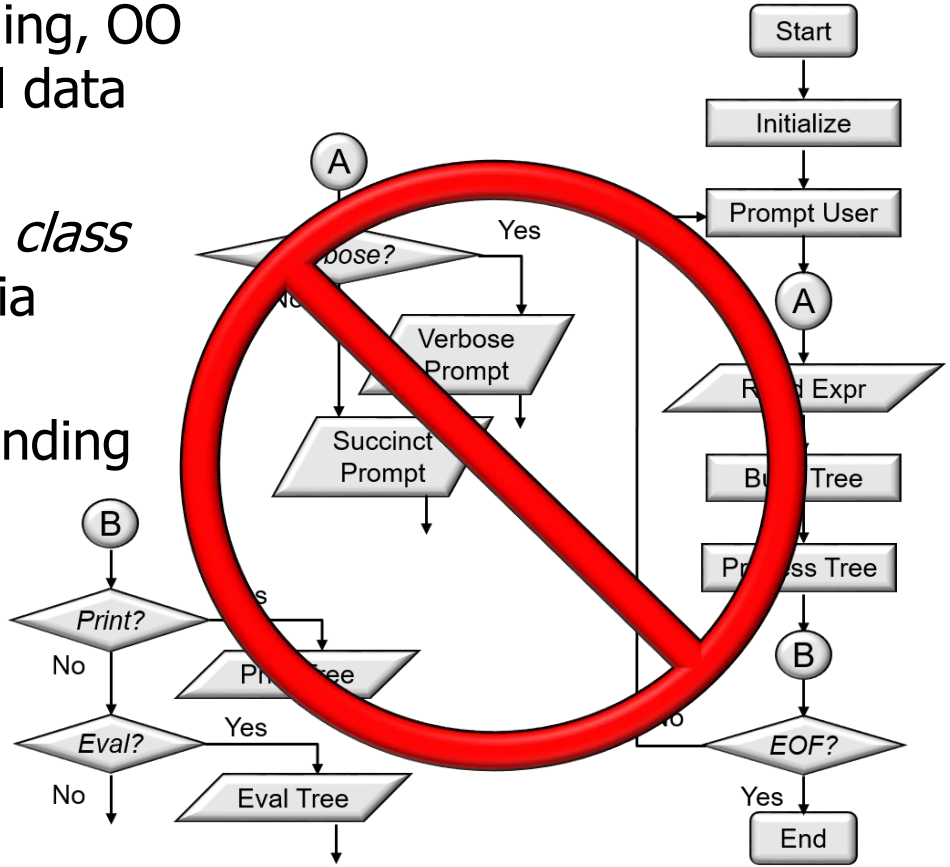
- In contrast to functional programming, OO programming employs "hierarchical data abstraction," e.g.,
  - Components are based on stable *class* roles & relationships extensible via inheritance & dynamic binding.



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

Functional vs. Object-Oriented Programming in Java

- In contrast to functional programming, OO programming employs "hierarchical data abstraction," e.g.,
  - Components are based on stable *class* roles & relationships extensible via inheritance & dynamic binding.
    - Rather than functions corresponding to algorithmic actions.



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

Functional vs. Object-Oriented Programming in Java

- In contrast to functional programming, OO 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

Functional vs. Object-Oriented Programming in Java

- In contrast to functional programming, OO 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);
```

Access & update internal iterator state

State is often "mutable" in OO programs.