Understand Java’s Key Functional Programming Concepts & Features

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 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
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 evaluation of math functions

Ideal, each function is “pure,” i.e., it has no side-effects on memory or I/O

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 evaluation of math functions

Note “function composition”: the output of one function serves as the input to the next function, etc.

See [martinfowler.com/articles/collection-pipeline](http://martinfowler.com/articles/collection-pipeline)
Key Functional Programming Concepts in Java

- Functional programming has its roots in lambda calculus, e.g.,

- Computations are treated as evaluation of math functions

**Functionally compute the \( n^{th} \) factorial in parallel**

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

Key Functional Programming Concepts in Java

• Functional programming has its roots in lambda calculus, e.g.,
• Computations are treated as evaluation of math functions

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

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 evaluation of math functions

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

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 evaluation of math functions

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

Successively combine two immutable long values & produce a new one
Key Functional Programming Concepts in Java

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

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

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

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

Key Functional Programming Concepts in Java

- Functional programming has its roots in lambda calculus, e.g.,

- Computations are treated as evaluation of math functions

- Changing state & mutable shared data are discouraged to avoid various hazards

```java
class Total {
    public long mTotal = 1;
    public void mult(long n) {
        mTotal *= n;
    }
}
```

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

Key Functional Programming Concepts in Java

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

```java
class Total {
    public long mTotal = 1;
    public void mult(long n) {
        mTotal *= n;
    }
}
```

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

Run in parallel

See [docs.oracle.com/javase/tutorial/collections/streams/parallelism.html](https://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 evaluation of math functions

• Changing state & mutable shared data are discouraged to avoid various hazards

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

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

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

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 evaluation of math functions
  - Changing state & mutable shared data are discouraged to avoid various hazards

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

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

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

Beware of inconsistent memory visibility
In Java *you* must avoid these hazards, i.e., the compiler & JVM won’t save you.

---

**Key Functional Programming Concepts in Java**

- Functional programming has its roots in lambda calculus, e.g.,

  - Computations are treated as evaluation of math functions

- Changing state & mutable shared data are discouraged to avoid various hazards

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

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

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

---

*Only you can prevent concurrency hazards!*
Key Functional Programming Concepts in Java

• Functional programming has its roots in lambda calculus, e.g.,
  • Computations are treated as evaluation of math functions
  • Changing state & mutable shared data are discouraged to avoid various hazards
  • Instead, focus is on “immutable” objects

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 evaluation of math functions
  • Changing state & mutable shared data are discouraged to avoid various hazards
  • Instead, focus is on “immutable” objects
    • Immutable object state cannot change after it is constructed

final class String {
  private final char value[];
  ...
  public String(String s) {
    value = s;
    ...
  }
  public int length() {
    return value.length;
  }
  ...
}
Key Functional Programming Concepts in Java

- Functional programming has its roots in lambda calculus, e.g.,
- Computations are treated as evaluation of math functions
- Changing state & mutable shared data are discouraged to avoid various hazards
- Instead, focus is on “immutable” objects
  - Immutable object state cannot change after it is constructed
- Java String is a common example of an immutable object

```java
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 evaluation of math functions
  
  • Changing state & mutable shared data are discouraged to avoid various hazards
  
  • Instead, focus is on “immutable” objects
    
    • Immutable object state cannot change after it is constructed
  
  • Java String is a common example of an immutable object
    
    • Fields are final & only accessor methods

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

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

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

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 algorithmic actions implemented as functions

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

See en.wikipedia.org/wiki/Imperative_programming

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

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

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

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

State is often “mutable” in OO programs
End of Understand Java’s Key Functional Programming Concepts & Features