Overview of Java

Supported Programming Paradigms

Douglas C. Schmidt
Learning Objectives in This Lesson

• Recognize the two programming paradigms supported by modern Java.

Naturally, these paradigms are also supported in versions above & beyond Java 8!
Learning Objectives in This Lesson

- Recognize the two programming paradigms supported by modern Java.
- Object-oriented programming
Learning Objectives in This Lesson

• Recognize the two programming paradigms supported by modern Java.
  • Object-oriented programming
  • Functional programming
Learning Objectives in This Lesson

• Recognize the two programming paradigms supported by modern Java.
  • Object-oriented programming
  • Functional programming

We’ll show some Java 8 code fragments that will be covered in more detail later.
Overview of Programming Paradigms in Java 8

Douglas C. Schmidt
Overview of Programming Paradigms in Java 8

- Java 8 is a “hybrid” that combines the object-oriented & functional paradigms.
Overview of Programming Paradigms in Java 8

• Object-oriented programming is an “imperative” paradigm.

See en.wikipedia.org/wiki/Imperative_programming
Overview of Programming Paradigms in Java 8

- Object-oriented programming is an “imperative” paradigm.
- e.g., a program consists of commands for the computer to perform.

Imperative programming focuses on describing how a program operates via statements that change its state.
Overview of Programming Paradigms in Java 8

- Object-oriented programming is an “imperative” paradigm.
- e.g., a program consists of commands for the computer to perform.

\[
\text{List<String> zap(List<String> lines, String omit) \{}
\]
\[
\text{List<String> res = new ArrayList<>();}
\text{for (String line : lines)}
\text{if (!omit.equals(line))}
\text{res.add(line);} \]
\[
\text{return res;}
\]

Imperatively remove a given string from a list of strings
Overview of Programming Paradigms in Java 8

- Object-oriented programming is an “imperative” paradigm.
- e.g., a program consists of commands for the computer to perform.

```java
List<String> zap(List<String> lines, String omit) {
    List<String> res = new ArrayList<>();
    for (String line : lines)
        if (!omit.equals(line))
            res.add(line);
    return res;
}
```

Create an empty list to hold results

![Diagram showing the intersection of Imperative and Object-Oriented paradigms with examples in C, C++, FORTRAN, Java, and C#]
Overview of Programming Paradigms in Java 8

- Object-oriented programming is an "imperative" paradigm.
- e.g., a program consists of commands for the computer to perform.

```java
List<String> zap(List<String> lines, String omit) {
    List<String> res = new ArrayList<>();
    for (String line : lines) {
        if (!omit.equals(line)) {
            res.add(line);
        }
    }
    return res;
}
```

Iterate sequentially through each line
Overview of Programming Paradigms in Java 8

- Object-oriented programming is an “imperative” paradigm.
- e.g., a program consists of commands for the computer to perform.

```java
List<String> zap(List<String> lines, String omit) {
    List<String> res = new ArrayList<>();
    for (String line : lines)
        if (!omit.equals(line))
            res.add(line);
    return res;
}
```

- e.g., C++, Java, C#
- e.g., C, FORTRAN

Only add lines that don’t match the `omit` string
• Object-oriented programming is an “imperative” paradigm.
• e.g., a program consists of commands for the computer to perform.

```java
List<String> zap(List<String> lines, String omit) {
    List<String> res = new ArrayList<>();
    for (String line : lines)
        if (!omit.equals(line))
            res.add(line);
    return res;
}
```

Return the list of nonmatching lines
Overview of Programming Paradigms in Java 8

- Object-oriented programming is an “imperative” paradigm.
- e.g., a program consists of commands for the computer to perform.

```java
List<String> zap(List<String> lines, String omit) {
    List<String> res = new ArrayList<>();
    for (String line : lines)
        if (!omit.equals(line))
            res.add(line);
    return res;
}
```

e.g., C, FORTRAN

e.g., C++, Java, C#
Overview of Programming Paradigms in Java 8

• Conversely, functional programming is a “declarative” paradigm.

See [en.wikipedia.org/wiki/Declarative_programming](en.wikipedia.org/wiki/Declarative_programming)
Conversely, functional programming is a “declarative” paradigm.

- e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps.

*Declarative programming focuses largely on “what” computations to perform, not “how” to compute them.*
Overview of Programming Paradigms in Java 8

• Conversely, functional programming is a “declarative” paradigm.
• e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps.

```java
List<String> zap(List<String> lines, String omit) {
    return lines
        .stream()
        .filter(not(omit::equals))
        .collect(toList());
}
```

Declaratively remove a given string from a list of strings.

See [github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex0](https://github.com/douglascraigschmidt/LiveLessons/tree/master/Java8/ex0)
Overview of Programming Paradigms in Java 8

- Conversely, functional programming is a “declarative” paradigm.
- e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps.

```java
List<String> zap(List<String> lines, String omit) {
    return lines.stream()
        .filter(not(omit::equals))
        .collect(toList());
}
```

- e.g., Prolog
- e.g., ML, Haskell

Convert list into a stream.
Overview of Programming Paradigms in Java 8

- Conversely, functional programming is a “declarative” paradigm.
- e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps.

```
List<String> zap(List<String> lines, String omit) {
    return lines
        .stream()
        .filter(not(omit::equals))
        .collect(toList());
}
```

- e.g., Prolog
- e.g., ML, Haskell

*Remove any line in the stream that matches the *omit* param.*
Overview of Programming Paradigms in Java 8

• Conversely, functional programming is a “declarative” paradigm.
• e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps.

```java
List<String> zap(List<String> lines, String omit) {
    return lines
        .stream()
        .filter(not(omit::equals))
        .collect(toList());
}
```

e.g., Prolog

Collect all nonmatching lines into a list & return it to the caller.
Overview of Programming Paradigms in Java 8

• Conversely, functional programming is a “declarative” paradigm.
• e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps.

```java
List<String> zap(List<String> lines, String omit) {
    return lines
        .stream()
        .filter(not(omit::equals))
        .collect(toList());
}
```

See [en.wikipedia.org/wiki/Fluent_interface](en.wikipedia.org/wiki/Fluent_interface)

Note the “fluent” programming style with cascading method calls.
Overview of Programming Paradigms in Java 8

- Conversely, functional programming is a “declarative” paradigm.
- e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps.

```java
List<String> zap(List<String> lines, String omit) {
    return lines
        .parallelStream()
        .filter(not(omit::equals))
        .collect(toList());
}
```

See [docs.oracle.com/javase/tutorial/collections/streams/parallelism.html](docs.oracle.com/javase/tutorial/collections/streams/parallelism.html)
Overview of Programming Paradigms in Java 8

• Conversely, functional programming is a “declarative” paradigm.
• e.g., a program expresses computational logic *without* describing control flow or explicit algorithmic steps.

```java
List<String> zap(List<String> lines, String omit) {
    return lines.parallelStream()
        .filter(not(omit::equals))
        .collect(toList());
}
```

Code was parallelized with minuscule changes since it’s declarative & stateless!