Overview of Java 8
Programming Paradigms

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 programming paradigms supported in Java 8
Overview of Programming Paradigms in Java 8
Java 8 was released in March 2014

What's New in JDK 8

Java Platform, Standard Edition 8 is a major feature release. This document summarizes features and enhancements in Java SE 8 and in JDK 8, Oracle's implementation of Java SE 8. Click the component name for a more detailed description of the enhancements for that component.

- Java Programming Language
  - Lambda Expressions, a new language feature, has been introduced in this release. They enable you to treat functionality as a method argument, or code as data. Lambda expressions let you express instances of single-method interfaces (referred to as functional interfaces) more compactly.
  - Method references provide easy-to-read lambda expressions for methods that already have a name.
  - Default methods enable new functionality to be added to the interfaces of libraries and ensure binary compatibility with code written for older versions of those interfaces.
  - Repeating Annotations provide the ability to apply the same annotation type more than once to the same declaration or type use.
  - Type Annotations provide the ability to apply an annotation anywhere a type is used, not just on a declaration. Used with a pluggable type system, this feature enables improved type checking of your code.
  - Improved type inference.
  - Method parameter reflection.
- Collections
  - Classes in the new java.util.stream package provide a Stream API to support functional-style operations on streams of elements. The Stream API is integrated into the Collections API, which enables bulk operations on collections, such as sequential or parallel map-reduce transformations.
  - Performance Improvement for HashMaps with Key Collisions

Java 8 was released in March 2014

What’s New in JDK 8

Java Platform, Standard Edition 8 is a major feature release. This document summarizes features and enhancements in Java SE 8 and in JDK 8, Oracle’s implementation of Java SE 8. Click the component name for a more detailed description of the enhancements for that component.

- Java Programming Language
  - Lambda Expressions, a new language feature, has been introduced in this release. They enable you to treat functionality as a method argument, or code as data. Lambda expressions let you express instances of single-method interfaces (referred to as functional interfaces) more compactly.
  - Method references provide easy-to-read lambda expressions for methods that already have a name.
  - Default methods enable new functionality to be added to the interfaces of libraries and ensure binary compatibility with code written for older versions of those interfaces.
  - Repeating Annotations provide the ability to apply the same annotation type more than once to the same declaration or type use.
  - Type Annotations provide the ability to apply an annotation anywhere a type is used, not just on a declaration. Used with a pluggable type system, this feature enables improved type checking of your code.
  - Improved type inference.
  - Method parameter reflection.
- Collections
  - Classes in the new java.util.stream package provide a Stream API to support functional-style operations on streams of elements. The Stream API is integrated into the Collections API, which enables bulk operations on collections, such as sequential or parallel map-reduce transformations.
  - Performance Improvement for HashMaps with Key Collisions.

See docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
Java 8 was released in March 2014

What's New in JDK 8

Java Platform, Standard Edition 8 is a major feature release. This document summarizes features and enhancements in Java SE 8 and in JDK 8, Oracle's implementation of Java SE 8. Click the component name for a more detailed description of the enhancements for that component.

- Java Programming Language
  - Lambda Expressions, a new language feature, has been introduced in this release. They enable you to treat functionality as a method argument, or code as data. Lambda expressions let you express instances of single-method interfaces (referred to as functional interfaces) more compactly.
  - Method references provide easy-to-read lambda expressions for methods that already have a name.
  - Default methods enable new functionality to be added to the interfaces of libraries and ensure binary compatibility with code written for older versions of those interfaces.
  - Repeating Annotations provide the ability to apply the same annotation type more than once to the same declaration or type use.
  - Type Annotations provide the ability to apply an annotation anywhere a type is used, not just on a declaration. Used with a pluggable type system, this feature enables improved type checking of your code.
  - Improved type inference.
  - Method parameter reflection.

- Collections
  - Classes in the new java.util.stream package provide a Stream API to support functional-style operations on streams of elements. The Stream API is integrated into the Collections API, which enables bulk operations on collections, such as sequential or parallel map-reduce transformations.
  - Performance Improvement for HashMaps with Key Collisions

See docs.oracle.com/javase/tutorial/collections/streams
Overview of Programming Paradigms in Java 8

- Java 8 is a “hybrid” that combines the object-oriented & functional paradigms

See www.deadcoderising.com/why-you-should-embrace-lambdas-in-java-8
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

```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

Imperatively remove a designated string from a list of strings

Note how this code is inherently sequential..
Overview of Programming Paradigms in Java 8

• Conversely, functional programming is a “declarative” paradigm

See en.wikipedia.org/wiki/Declarative_programming
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

*Declarative programming focuses on “what” computations to perform, not “how” to compute them*
Conversely, functional programming is a “declarative” paradigm. A program expresses computational logic \textit{without} describing control flow or explicit algorithmic steps.

```java
def zap(List<String> lines, String omit) {
    return lines
        .stream()
        .filter(line -> !omit.equals(line))
        .collect(toList());
}
```

Examples include Prolog, ML, and Haskell.
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(line -> !omit.equals(line))
        .collect(toList());
}
```

Note “fluent” programming style with cascading method calls

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

• Conversely, functional programming is a “declarative” paradigm

• e.g., a program expresses computational logic \textit{without} describing control flow or explicit algorithmic steps

\begin{verbatim}
List<String> zap(List<String> lines, String omit) {
    return lines
        .parallelStream()
        .filter(line -> !omit.equals(line))
        .collect(toList());
}
\end{verbatim}

Perform filtering \textit{in parallel}
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(line -> !omit.equals(line))
        .collect(toList());
}
```

Perform filtering in parallel

Note how this code is can be parallelized with miniscule changes..
End of Overview of Java 8