Java 8: Overview and Quick Review of Supporting Topics

Originals of slides and source code for examples: http://www.coreservlets.com/java-8-tutorial/
Also see the general Java programming tutorial – http://courses.coreservlets.com/Course-Materials/java.html
and customized Java training courses (onsite or at public venues) – http://courses.coreservlets.com/java-training.html

For additional materials, please see http://www.coreservlets.com/. The Java tutorial section contains complete source code for all examples in this tutorial series, plus exercises and exercise solutions for each topic.

λ

Topics in This Section

- **Overview, installation, and setup**
- **References**
- **Review of basic handlers and anonymous classes**
  - Separate classes
  - Main class that implements the interface
  - Named inner classes
  - Anonymous inner classes
- **Review of making generic classes and methods**
- **A few common Eclipse techniques**

Getting Started with Java 8

The 0 km marker, starting point for distances in all of India.

Why Java 8?

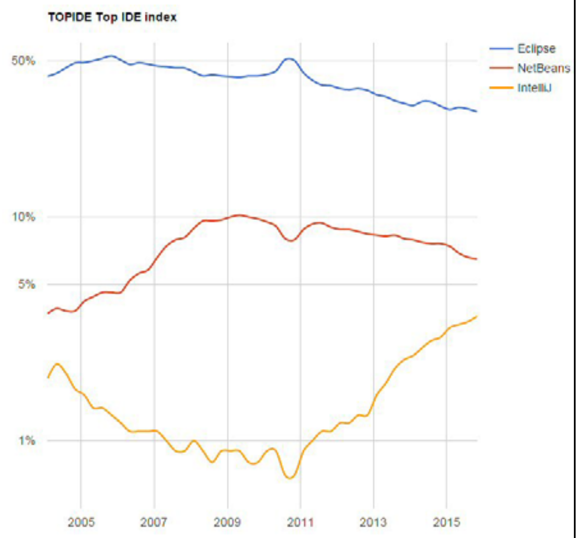
- **Make code more flexible and reusable**
  - Lambda expressions
- **Make code more convenient to write**
  - High-level methods in Stream interface
- **Make code faster and more memory efficient**
  - Lazy evaluation and automatic parallelization for streams
- **Adapt to the times**
  - Others will be using lambdas and streams, since they are standard part of Java now. So, you have to learn it simply to be able to use and modify others' code.
    - Besides, once you get a taste of the benefits, you will want to use the new features frequently

Step 1:Get Java 8 Code and JavaDocs

- **Main download page**
  - http://www.oracle.com/technetwork/java/javase/downloads/
    - JDK 1.8 itself
    - Downloadable JavaDoc API (if you want an offline copy)
- **Online API**
  - http://docs.oracle.com/javase/8/docs/api/
- **More info**
  - http://openjdk.java.net/projects/lambda/
    - Papers, FAQ, IDE info
- **Final Java SE 8 release was March 2014**
  - So, Java 8 should not be considered new, untested, or risky
  - Almost all new projects should use Java 8. Java 9 release scheduled for March 2017. Onsite Java 9 training from coreservlets.com coming soon. Email hall@coreservlets.com if interested.

Step 2: Install IDE that Supports Java 8

- **Eclipse**
  - Luna or later
    - http://www.eclipse.org/downloads/
  - Info on Java-8-specific features in Eclipse
    - https://www.eclipse.org/community/eclipse_newsletter/2014/june/article1.php
- **NetBeans**
  - NetBeans 8 and later
    - https://netbeans.org/downloads/
  - Info on Java-8-specific features in NetBeans
    - https://netbeans.org/kb/docs/java/javase-jdk8.html
- **IntelliJ IDEA**
  - IDEA 13.1 and later
    - http://www.jetbrains.com/idea/
  - Info on Java-8-specific features in IDEA
    - https://www.jetbrains.com/idea/features/code_editor.html



All three are popular IDEs (Integrated Development Environments) for Java. There is no agreement on which is better, but there is clear empirical evidence that Eclipse is more widely used.

Graph from http://pypl.github.io/IDE.html. Note the log scale.

Java 8 References

Online References

- **The Oracle Java tutorial**
  - Lambda expressions
    - https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html
  - Method references
    - https://docs.oracle.com/javase/tutorial/java/javaOO/methodreferences.html
  - Streams
    - https://docs.oracle.com/javase/tutorial/collections/streams/
- **Lambda project home page**
  - http://openjdk.java.net/projects/lambda/
- **The coreservlets.com Java 8 tutorial**
  - http://www.coreservlets.com/java-8-tutorial/

Books

- ***Mastering Lambdas: Java Programming in a Multicore World***
  - By Maurice Naftalin – https://www.mhprofessional.com/details.php?isbn=0071829628
- ***Java 8 Lambdas: Functional Programming for the Masses***
  - By Richard Warburton
    - http://www.r-5.org/files/books/computers/languages/java/fp/Richard_Warburton-Java_8_Lambdas-EN.pdf
    - http://shop.oreilly.com/product/0636920030713.do
- ***Functional Programming in Java: Harnessing the Power of Java 8 Lambda Expressions***
  - By Venkat Subramanian
    - http://pragprog.com/book/vsjava8/functional-programming-in-java
- ***Java 8 in Action***
  - By Raoul-Gabriel Urma and Mario Fusco – http://www.manning.com/urma/
- ***Java SE8 for the Really Impatient***
  - By Cay Horstmann – http://horstmann.com/java8/

coreservlets.com – custom onsite training

Quick Review of Pre-Lambda Handlers

Skip this section if you already know about anonymous inner classes

Main Alternatives

- **Big idea**
  - You need to designate code to be executed by other entity
    - E.g., you need code to respond a button click, to run in the background for threads, or to handle sorting comparisons
- **Pre-lambda alternatives**
  - Separate (normal) class
    - `Arrays.sort(theArray, new SeparateClass(...));`
  - Main class (which implements interface)
    - `Arrays.sort(theArray, this);`
  - Named inner class
    - `Arrays.sort(theArray, new InnerClass(...));`
  - Anonymous inner class
    - `Arrays.sort(theArray, new Comparator<String>() { ... });`

Separate Classes

```
public class StringSorter1 {
  public static void doTests() {
    String[] testStrings = {"one", "two", "three", "four"};
    System.out.print("Original: ");
    ArrayUtils.printArray(testStrings);
    Arrays.sort(testStrings, new StringLengthComparator());
    System.out.print("After sorting by length: ");
    ArrayUtils.printArray(testStrings);
    Arrays.sort(testStrings, new LastLetterComparator());
    System.out.print("After sorting by last letter: ");
    ArrayUtils.printArray(testStrings);
  }
}
```

ArrayUtils.printArray is a minor helper method that turns the array into a List, then prints the List. It is shown on an upcoming slide.

Separate Classes (Continued)

```
public class StringLengthComparator
             implements Comparator<String> {
  @Override
  public int compare(String s1, String s2) {
    return(s1.length() - s2.length());
  }
}
```

Separate Classes (Continued)

```
public class LastLetterComparator
             implements Comparator<String> {
  @Override
  public int compare(String s1, String s2) {
    return(s1.charAt(s1.length() - 1) –
           s2.charAt(s2.length() - 1));
  }
}
```

Test Class

```
public class StringSorter1Test {
  public static void main(String[] args) {
    StringSorter1.doTests();
  }
}
```

Output

```
Original: {one, two, three, four}
After sorting by length: {one, two, four, three}
After sorting by last letter: {one, three, two, four}
```

For other examples, test classes are similar and output is identical, so test classes and output won't be repeated.

Helper Class (for Printing Array)

```
public class ArrayUtils {
  public static <T> void printArray(T[] entries) {
    System.out.println(Arrays.asList(entries));
  }

  ...
}
```

Prints an array by turning it into a List, then printing the List.

Lists already have useful toString methods that print the individual entries delimited by commas and surrounded by square brackets.

Separate Classes: Pros and Cons

- **Advantages**
  - Flexible: can pass arguments to class constructor
  - More reusable: loosely coupled
- **Disadvantage**
  - Need extra step to call methods in main app. How does handler call code in the main application? It needs a reference to the main app to do so.
    - E.g., you pass main app instance to constructor
    - But, even then, the methods you call in the main app must be public

Main App Implementing Interface

```
public class StringSorter2 implements Comparator<String> {
  public void doTests() {
    String[] testStrings = {"one", "two", "three", "four"};
    System.out.print("Original: ");
    ArrayUtils.printArray(testStrings);
    Arrays.sort(testStrings, this);
    System.out.print("After sorting by length: ");
    ArrayUtils.printArray(testStrings);
    System.out.println("NO sorting by last letter.");
  }

  @Override
  public int compare(String s1, String s2) {
    return(s1.length() - s2.length());
  }
}
```

Notice that with this option, we sort strings only one way, not two ways as with the other approaches.
See next slide.

Implementing Interface: Pros and Cons

- **Advantages**
  - No extra steps needed to call methods in main app
    - The code is *part* of the main app, so it can call any method or access any instance variable, even private ones.
  - Simple
    - Widely used in real life for button handlers, when you know you will have only one button, or for threading code when you need only one method to run in the background.
- **Disadvantage**
  - Inflexible: hard to have multiple different versions since you cannot pass arguments to handler
    - For example, what if you want to sort arrays two different ways? If you supply “this” as second argument to Arrays.sort, it will refer to the same compare method both times.

Named Inner Class

```
public class StringSorter3 {
  public void doTests() {
    String[] testStrings = {"one", "two", "three", "four"};
    System.out.print("Original: ");
    ArrayUtils.printArray(testStrings);
    Arrays.sort(testStrings, new StringLengthComparator2());
    System.out.print("After sorting by length: ");
    ArrayUtils.printArray(testStrings);
    Arrays.sort(testStrings, new LastLetterComparator2());
    System.out.print("After sorting by last letter: ");
    ArrayUtils.printArray(testStrings);
  }
```

Class continued on next slide

Named Inner Class (Continued)

```
  private class StringLengthComparator2 implements Comparator<String> {
    @Override
    public int compare(String s1, String s2) {
      return(s1.length() - s2.length());
    }
  }

  private class LastLetterComparator2 implements Comparator<String> {
    @Override
    public int compare(String s1, String s2) {
      return (s1.charAt(s1.length() - 1) –
              s2.charAt(s2.length() - 1));
    }
  }
}
```

This is a continuation of class shown on previous page.

Named Inner Classes: Pros and Cons

- **Advantages**
  - No extra steps needed to call methods in main app. Inner classes have full access to all methods and instance variables of surrounding class, including private ones.
    - However, if there is a name conflict, you have to use the unpopular OuterClass.this.name syntax
  - Flexible: can define constructor and pass arguments
- **Disadvantage**
  - A bit harder to understand (arguably)

Anonymous Inner Class

```
public class StringSorter4 {
  public void doTests() {
    String[] testStrings = {"one", "two", "three", "four"};
    System.out.print("Original: ");
    ArrayUtils.printArray(testStrings);
    Arrays.sort(testStrings, new Comparator<String>() {
      @Override
      public int compare(String s1, String s2) {
        return(s1.length() - s2.length());
      }
    });
    System.out.print("After sorting by length: ");
    ArrayUtils.printArray(testStrings);
```

Anonymous Inner Class (Continued)

```
    Arrays.sort(testStrings, new Comparator<String>() {
      @Override
      public int compare(String s1, String s2) {
        return(s1.charAt(s1.length() - 1) –
               s2.charAt(s2.length() - 1));
      }
    });
    System.out.print("After sorting by last letter: ");
    ArrayUtils.printArray(testStrings);
  }
}
```

Anonymous Inner Classes: Pros and Cons

- **Advantages**
  - As with named inner classes, full access to code of surrounding class (even private methods and variables)
  - Slightly more concise than named inner classes
    - But still bulky and verbose
- **Disadvantage**
  - Harder to understand (arguably)
  - Not reusable
    - Cannot use the same anonymous class definition in more than one place

Preview of Lambdas

- **Desired features**
  - Full access to code from surrounding class
  - No confusion about meaning of "this"
  - Much more concise, succinct, and readable
  - Encourage a functional programming style
- **Quick peek**
  - Anonymous inner class

```
Arrays.sort(testStrings, new Comparator<String>() {
  @Override
  public int compare(String s1, String s2) {
    return(s1.length() - s2.length());
  }
});
```

  - Lambda

```
Arrays.sort(testStrings, (s1, s2) -> s1.length() - s2.length());
```

Building Generic Methods and Classes: Overview

Skip this section if you already know how to make your own methods and classes that support generic types

Using Existing Generic Methods and Classes

- **Basic capability**
  - Even beginning Java programmers need to know how to *use* classes that support generics
  - You cannot properly use Lists, Maps, Sets, etc. without this
  - Covered in earlier section

```
List<Employee> workers = ...;
Map<String,Employee> workerDatabase = ...;
```

Creating Your Own Generic Methods and Classes

- **Intermediate capability**
  - Intermediate Java developers should also to be able to *define* classes or methods that support generics
  - In Java 7 and earlier, being able to do this was mostly reserved for advanced developers, but it is done much more commonly in Java 8
    - Because lambda functions and generic types work together for same goal: to make code more reusable

```
public interface Map<K,V> { ... }
public static <T> T lastElement(List<T> elements) { ... }
```

Generic Classes and Methods: Syntax Overview

- **Using <TypeVariable>**
  - If you put variables in angle brackets in the class or method definition, it tells Java that uses of those variables refer to types, not to values
  - It is conventional to use short names in upper case, such as T, R (input type, result type) or T1, T2 (type1, type2), or E (element type)
- **Examples**

```
public class ArrayList<E> ... {
  ...
}

public static <T> T randomElement(T[] array) {
  ...
}
```

Generic Methods

Generic Classes and Methods: Syntax Details

- **Declaring methods that support generics**

```
public static <T> T best(List<T> entries, ...) { ... }
```

  - This says that the best method takes a List of T's and returns a T. The <T> at the beginning means T is not a real type, but a type that Java will figure out from the method call.

- **Java will figure out the type of T by looking at parameters to the method call**

```
List<Person> people = ...;
Person bestPerson = Utils.best(people, ...);
List<Car> cars = ...;
Car bestCar = Utils.best(cars, ...);
```

Partial Example: randomElement

```
public class RandomUtils {
  ...

  public static <T> T randomElement(T[] array) {
    return(array[randomIndex(array)]);
  }
}
```

- In rest of method, T refers to a type.
- Java will figure out what type T is by looking at the parameters of the method call.
- Even if there is an existing class actually called T, it is irrelevant here.

This says that the method takes in an array of T's and returns a T. For example, if you pass in an array of Strings, you get out a String; if you pass in an array of Employees, you get out an Employee. No typecasts required in any of the cases.

Complete Example: randomElement

```
public class RandomUtils {
  private static Random r = new Random();

  public static int randomInt(int range) {
    return(r.nextInt(range));
  }

  public static int randomIndex(Object[] array) {
    return(randomInt(array.length));
  }

  public static <T> T randomElement(T[] array) {
    return(array[randomIndex(array)]);
  }
}
```

Using RandomUtils

- **Examples**

```
String[] names = { "Joe", "John", "Jane" };
String name = RandomUtils.randomElement(names);
Color[] colors = { Color.RED, Color.GREEN, Color.BLUE };
Color color = RandomUtils.randomElement(colors);
Person[] people =
  { new Person("Larry", "Page"), new Person("Larry", "Ellison"),
    new Person("Larry", "Bird"), new Person("Larry", "King") };
Person person = RandomUtils.randomElement(people);
Integer[] nums = { 1, 2, 3, 4 };    // Integer[], not int[]
int num = RandomUtils.randomElement(nums);
```

- **Points**
  - No typecast required to convert to String, Color, Person, Integer
  - Autoboxing lets you assign entry from Integer[] to an int, but array passed to randomElement must be Integer[] not int[], since generics work only with Object types, not primitive types

Generic Classes or Interfaces

Generic Classes and Methods: Syntax Details

- **Declaring classes or interfaces that support generics**

```
public class SomeClass<T> { ... }
```

- **Methods in the class can now refer to T both for arguments and for return values**

```
public T getSomeValue(int index) { ... }
```

- **Java will figure out the type of T by your declaration**

```
SomeClass<Person> blah = new SomeClass<>();
```

Example: Generic Class (Simplified)

```
public class ArrayList<E> {

  public E get(int index) { ... }

  public boolean add(E element) { ... }

  ...
}
```

In rest of class, E does not refer to an existing type. Instead, it refers to whatever type was defined when you created the list. E.g., if you did ArrayList<**String**> words = ...; then E refers to String.

This says that get returns an E. So, if you created ArrayList<**Employee**>, get returns an Employee. No typecast required in the code that calls get.

This says that add takes an E as a parameter. So, if you created ArrayList<**Circle**>, add can take only a Circle.

This is a highly simplified version of the real java.util.ArrayList class. That class implements multiple interfaces, and the generic support comes from the interfaces.

Some Eclipse Shortcuts

A Few Eclipse Tricks

- **Making a new project**
  - File → New → Project → Java → Java Project
- **Making new package**
  - R-click project, New → Package
- **Making a new class**
  - R-click package, New → Class
- **Autocompletion**
  - Type part of a class or method name, Control-Space
- **Inserting main method**
  - Type the word “main”, then Control-Space
- **Inserting System.out.println**
  - Type the word “sysout”, then Control-Space
- **Renaming a class, variable, or method**
  - Select class, variable, or method, R-click, Refactor → Rename
    - Will also change all places that refer to it

Wrap-Up

Summary

- **Install Java 8**
  - http://www.oracle.com/technetwork/java/javase/downloads/
- **Bookmark the Java 8 JavaDocs API**
  - http://docs.oracle.com/javase/8/docs/api/
- **Get an IDE**
  - Eclipse, NetBeans, IntelliJ IDEA, etc.
- **Review options for handlers**
  - Especially anonymous inner classes.
    - This is the main alternative that Java 8 lambdas replace.
- **Review making generic classes and methods**
  - Techniques used frequently with lambdas

Questions?