Lambda Expressions

Reference: Java SE8 for the Really Impatient by Cay S. Horstmann

1. Lambda expressions in Java 8

- A functional interface is an interface that defines a single abstract method. Note that, in Java 8, an interface can declare non-abstract methods.
- A lambda expression describes an instance of a functional interface, meaning an instance of the class that implements a functional interface.
- A lambda expression can be passed around and executed later once or multiple times.

2. Syntax

- Parameter → an expression or {multiple statements}
- (Parameter1, Parameter2, …) → an expression or {multiple statements}

3. Examples

Example1: Comparator

```
import java.lang.reflect.Array;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;

public class ComparatorTester
{  public static void main(String[] args)
   {
      ArrayList<String >words = new ArrayList<String>();
      words.add("apricot");
      words.add("corn");
      words.add("banana");

      Collections.sort(words,
                       new
                       Comparator<String>()
                       {  public int compare(String s1, String s2)
                          { return s1.length() - s2.length(); }
                       }
               );
      Collections.sort(words,
                        (s1,s2) -> { return s1.length() - s2.length(); }
                        );

      for (String s: words) System.out.println(s);  // corn banana apricot
   }
}
```

Example 2: Listeners

```
public class ButtonTester
{
   public static void main(String[] args)
   {
      JFrame frame = new JFrame();

      final int FIELD_WIDTH = 20;
      final JTextField textField = new JTextField(FIELD_WIDTH);
      textField.setText("Click a button!");

      JButton helloButton = new JButton("Say Hello");
      /*
       helloButton.addActionListener(new
         ActionListener()
         {
            public void actionPerformed(ActionEvent event)
            {
               textField.setText("Hello, World!");
               System.out.println(event);
            }
         });
     */

      /*
      helloButton.addActionListener( event ->
                    {
                     textField.setText("Hello, World!");
                     System.out.println(event);
                     }
               );

      */

    frame.setLayout(new FlowLayout());

    frame.add(helloButton);
    frame.add(textField);

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);
   }
}
```

4. Variable Scope

- A lambda expression can use the value of a variable in the enclosing scope.
- When a lambda expression access a local variable or parameter, then the variable must be **effectively final.** (In a lambda expression, you can only reference variables whose value does not change.)

Example: In the ButtonTester class, textField is final because it is a local variable and being used by the lambda expression.

```
final JTextField textField = new JTextField(FIELD_WIDTH);

helloButton.addActionListener(event->textField.setText("Hello, World!"));
```

- The body of lambda expression has the same scope as a nested block – The same rule for name conflicts and shadowing apply.

5. Semantics of `this` in a lambda expression

- In lambda expressions, the semantic of `this` has been changed to reference to the instance of the surrounding class which is the class that defines the lambda expression. There is no way to reference to the lambda expression from inside the lambda.
- In inner classes, `this` inside an instance of an inner class references to the instance of the inner class.

  Example:

```
public class Application()
{
   public void doWork()
   {
      Runnable runner =
          ()→ {...; System.out.println(this.toString()); …};

   }
}
```

  In the above example, this references the Application Object, not the functional instance Runnable.

6. Return statement and lambda expressions

A lambda expression may use an explicit return statement. In that, every branch should return a value.

`x -> { if (x >= 0) return 1;}` is an error.