Lambda Expressions

Part 1

Chapter 6, Core Java Volume I

Contents

Functional Interfaces

- A *functional interface* is an interface with a single abstract method.
- Examples:

```
interface MySupplier
{
   double get();
}
```

```
interface MyFunction
{
   double apply(double v);
}
```

- Library interface examples:

```
interface ActionListener extends EventListener
{
   void actionPerformed(ActionEvent e);
}
```

```
interface Comparator<T>
{
   int Compare(T o1, T o2);
}
```

Functional Interfaces

- Using the **Comparator** Interface
  - When sorting strings by length, you can pass a **Comparator** object to the sort method.
  - The **compare** method had to be called repeatedly to compute

```
class LengthComparator implements Comparator<String>
{
   public int compare(Sting first, String second)
   {
       return first.length() - second.length();
    }
}
```

```
String[] list = new String[5];
...
Arrays.sort(list, new LengthComparator());
```

Functional Interfaces

- Using the **ActionListerner** Interface
  - You give the timer an **ActionListener** object and the timer calls back the **actionPerformed** method in the **ActionListener** object.

```
class TimePrinter implements ActionListener
{
   public void actionPerformed(ActionEvent event)
   {
       // printing time info.
    }
}
```

```
Timeer t = new Timer(4000, new TimePrinter());
t.start();
```

- Both examples have something in common; a block of code was passed to someone - **Arrays.sort** method or a **Timer** object and that block of code was called at some later time.
- Until java 8, we cannot pass code blocks because java is object-oriented so we create and pass *an object* of a class which has a method with the desired code.

Lambda Expressions

- Lambda Expression (in general)
  - *an anonymous function (*or block of code with parameters)
  - can be stored in variables or passed to functions as a parameter and then executed later one time or multiple times (*first-class objects*)

- Lambda Expression (in Java)
  - represents an instance of a functional interface
  - a simple way of implementing functional interfaces

- A lambda expression can be used whenever *a functional interface object* is expected:

```
Arrays.sort(words, (first, second) -> first.length() - second.length()) ;

Timer t = new Timer ( 1000, event -> System.out.println("At the tone, the time is " + new Date()) ) ;
```

The Syntax of Lambda Expressions

- General form : *parameters -> code*
- Simplest form: (parameter list) -> an expression or a method call

```
(String first, String second) -> first.length() - second.length()
event -> System.out.println("At the tone, the time is " + new Date())
```

- If the code has multiple statements, use { }:

```
(String first, String second) ->
{
   if (first.length() < second.length()) return -1;
   else if (first.length() > second.length()) return 1;
   else return 0;
}
```

- If there are no parameters, you still supply parentheses:

```
() -> Toolkit.getDefaultToolkit().beep();
```

- If parameter types can be inferred, omit them:

```
Comparator<String> comp = (first, second) -> first.length() - second.length();
```

- If there is exactly one parameter with inferred type, omit parentheses:

```
ActionListener listener = event -> Toolkit.getDefaultToolkit().beep();
```

Listing 6.6 lambda/LambdaTest.java

```
package lambda;
import java.util.*;
import javax.swing.*;
import javax.swing.Timer;
public class LambdaTest
{
   public static void main(String[] args){
      String[] planets = new String[] { "Mercury", "Venus", "Earth", "Mars",  "Jupiter", "Saturn", "Uranus", "Neptune" };
      System.out.println(Arrays.toString(planets));
      System.out.println("Sorted in dictionary order:");
      Arrays.sort(planets);
      System.out.println(Arrays.toString(planets));
      System.out.println("Sorted by length:");
      Arrays.sort(planets, (first, second) -> first.length() - second.length());
      System.out.println(Arrays.toString(planets));
      Timer t = new Timer(1000, event ->  System.out.println("The time is " + new Date()) );
      t.start();
      // keep program running until user selects "Ok"
      JOptionPane.showMessageDialog(null, "Quit program?");
      System.exit(0);
   } // end of main()
} // end of LambdaTest
```

Executing Lambda Expressions

- A lambda expression itself does not contain the information about which functional interface (target type) it is implementing. That information is *deduced from the context.*

```
MySupplier mySVal;
mySVal = () -> 98.6;        // MySupplier is the target type of the lambda expression

MyFunction myFVal;
myFVal = (n) -> 1.0/n;      // MyFunction is the target type of the lambda expression
```

- When the method defined in the functional interface is called through the target, the lambda expression is *executed*.

```
mySVal.get();               // 98.6
myFVal.apply(4.0);          // 0.25
myFVal.apply(8.0);          // 0.125
```

Executing Lambda Expressions

- How to interpret the following statement?

```
Arrays.sort(words, (first, second) -> first.length() - second.length()) ;
```

1. Compiler matches the call with sort(Sting[] a, Comparator<String> c) // type inference
2. Pass the lambda expression to the parameter c of sort
3. Sort executes the lambda expression (a comparator object) to compare two elements

```
c.compare(a[i], a[j])  // executing the lambda expression
```