Benton Luu

CPSC 501 – Assignment 2: Refactoring

Link to GitHub Repository: https://github.com/bentonluu/CPSC501-Assignment2-JavaReflection

Refactoring 1: Long Method -> Extract Method
(GitHub SHA: 450d1cc736b15c22f2e222006446f031c1f3a567)

- In the method 'inspectClass', all the print statements and the reflection code to inspect a class and object was all done in that method which resulted in making it a long method.
- Using the refactoring method of Extract Method, extracted 6 methods from the 'inspectClass' method: 'printClass', 'printClass', 'printInterface', 'printConstructor', 'printMethods', and 'printFields'. Each of these methods correspond to the individual inspection component information that was required to appear in the script files.
- To test that the implementation of the refactoring worked correctly, created 6 JUnit tests ('printClassTest', 'printSuperclassTest', printInterfaceTest', 'printConstructorTest', 'printMethodTest', and 'printFieldTest') to verify that each method was printing out the correct information from inspecting a class and object.
- The improvements from this refactoring was that it reduced the complexity of the method 'inspectClass' and made it easier to troubleshoot an issue pertaining to an inspection section of code. It separated the responsible into their own methods which overall made the code more understandable in what each method does.

Benton Luu

BEFORE (Class and Superclass)

Inspector.java

```
public class Inspection {
   private void inspectClass(Class c, Object obj, boolean recursive, int depth) {
      numOfTabs(depth);
      System.out.println("Class: " + c)

      Class superclass = c.getSuperclass();
      numOfTabs(depth);
      System.out.println("Class: " + c)
      …
   }
   …
}
```

AFTER (Class and Superclass)

Inspector.java

```
public class Inspection {
   private void inspectClass(Class c, Object obj, boolean recursive, int depth) {
      printClass(c, depth);
      printSuperClass(c, obj, recursive, depth);
      …
      }

   public void printClass(Class c, int depth) {
      formatPrint("CLASS", depth);
      formatPrint("Class: " + c + "\n", depth);
   }

   public void printSuperclass(Class c, Object obj, boolean recursive, int depth) {
      formatPrint("SUPERCLASS (" + c.getName() + ")", depth);
      Class superclass = c.getSuperclass();
      if (superclass != null) {
         formatPrint("Superclass: " + superclass + "\n", depth);
         inspectClass(c.getSuperclass(),obj,recursive, depth + 1);
      }
      else {
         formatPrint("Superclass: null\n", depth);
      }
   }
   …
}
```

Benton Luu

Refactoring 2: Duplicate Code -> Replace Parameter with Method
(GitHub SHA: 450d1cc736b15c22f2e222006446f031c1f3a567)

- There was redundant 'System.out.println' methods scatted throughout the code used to format the content in the script files.
- Using the refactoring method Replace Parameter with Method, replaced the long System.out.println parameter list with a new method called 'printFormat' which took the content being printed and the depth level as parameters and did all the formatting within that method instead of within the individual inspect section methods and using multiple print methods.
- Created JUnit tests 'printFormatTest()' and 'tabIndicationTest()' which were used to check that the formatting of the printed content to console was configurated correctly depending what string content was given and the depth level of recursion.
- The enhancement to the code from the refactoring method being used was that it removed unnecessary print methods from the code making it shorter as well as simplified formatting the content based on the depth level. By having the formatting done in one method instead of all over the code, it makes it much easier to resolve an issue since the formatting is encompassed into a single method that is used throughout the code.

BEFORE (Class)

Inspector.java

```
public class Inspection {
   private void inspectClass(Class c, Object obj, boolean recursive, int depth) {
      numOfTabs(depth);
      System.out.println("Class: " + c)
      …
   }
   …
}
```

AFTER (Class)

Inspector.java

```
public class Inspection {
   private void inspectClass(Class c, Object obj, boolean recursive, int depth) {
      printClass(c, depth);
      …
      }

   public void printClass(Class c, int depth) {
      formatPrint("CLASS", depth);
      formatPrint("Class: " + c + "\n", depth);
   }
   …
}
```