COMP1035 – Coursework 2 – JUnit Input/Output

Optional Exercise 1 – Checking What's Printed Out

This can be hard! Because your test needs to know what is printing out. Experience Pro Tester, Racheal Zane, has wants to help you – these exercises should help you do more advanced tests in your project. Here is a way how – it is a bit complicated for now, and you will learn more about printstreams and byte arrays in the future.

NOTE – there is no need to put this in your CW2 project or repository – just set up a separate Java project for this.

1. Create a `InOutSystem` Java file with the following code:

```
import java.util.Scanner;

public class InOutSystem {
    public InOutSystem() { }
    public void printOutput() {
         System.out.print("Hello");
    }
}
```

2. Create a JUnit test file for it.

3. We need to change `System.out` is going to – rather than to print out, we'll create a place for it to go. Inside your test class, add the following variable.

```
private static final ByteArrayOutputStream outContent
  = new ByteArrayOutputStream();
```

   it will ask you if you want to `import java.io.ByteArrayOutputSystem` – you do!

4. We need to set that up with a `@Before` – `System.setOut` tells `System.out` to send its output somewhere else.

```
@BeforeAll
public static void setUpStreams() {
    System.setOut(new PrintStream(outContent));
}
```

5. We also need to unset it for the future with a `@After`.

```
@AfterAll
public static void cleanUpStreams() {
    System.setOut(null);
}
```

6. Now your test can create the object, then ask it to print "Hello", because you told it to go `outContent`.

```
@Test
public void testPrintOutput() {
```

```
    InOutSystem ios = new InOutSystem();
    ios.printOutput();
    assertEquals("Hello", outContent.toString());
    // outContent.reset();
}
```

(You can change "Hello" to something else to see that it is different, and the test will fail.)

Optional Exercise 2 – Pretending to Enter Inputs Like a User

This is also hard, but the same principle applies. Make sure to comment out the test in Exercise 1 before executing the test.

7. Add a new method to your original class, to print what the user enters in.

```
public void printInput() {
    Scanner in = new Scanner(System.in);
    String input = in.nextLine();
    System.out.print(input);
    in.close();
}
```

8. Create a new test in your JUnit test file. This creates the object, then creates the string the user will enter. Then you can create an `InputStream` (the opposite to the last exercise – `java.io` package). And set `System.setIn` to be this stream. Then you call your new method that prints what the "user" enters. And like the last test, checks what your fake system output now contains.

```
@Test
public void testPrintInput() {
    // outContent.reset();
    InOutSystem ios = new InOutSystem();
    String input = "CW2 is so easy";
    InputStream in = new ByteArrayInputStream(input.getBytes());
    System.setIn(in);
    ios.printInput();
    assertEquals("CW2 is so easy", outContent.toString());
}
```

Optional Exercise 3 – How to Test the Private Methods

This could be difficult, an article by Bill Venners listed all the possible options to test the private methods, but we will be using the "reflection technique".

9. Let's add a private method printSecret(String s) in your `InOutSystem` java file.

```
private String printSecret(String s) {
    return s;
}
```

10. Now, in your JUnit test file, add in the following code to test,

```
@Test
public void testSecret() throws IllegalAccessException,
```

```
  IllegalArgumentException,
  InvocationTargetException,
  NoSuchMethodException,
  SecurityException {
    InOutSystem ios = new InOutSystem();

    // For JUnit5, use getClass instead of class
    Method m = InOutSystem.class.getDeclaredMethod("printSecret",
    String.class);
    m.setAccessible(true);

    String ot = (String)m.invoke(ios, "I am Iron Man");

    assertEquals("I am Iron Man", ot));
}
```

If you have succeeded, try to figure out how do you modify the coding above to test the following private methods:

- `private void printSecret()` `// no return`
- `private int printSecret(int num)` `// return type is int`
- `private String printSecret(String s1, String s2)` `// two arguments`

Coursework 3 Advice

The above gives you all the right types of tools to get some difficult mark for CW2. But you will still have to figure some harder bits to be able to test the most complicated part of the system!

Hint 1: Could always refactor Louis Litt's bad-smelling code to make it easier to test (as if it had been built to prove tests in the first place!).

Hint 2: The example above uses `print()` rather than `println()` – the latter adds a `System.lineSeparator()` to a string.

So if changes the `printOutput()` method to use `println()` instead of `print()`,needs to use

```
assertEquals("Hello" + System.lineSeparator(), outContent.toString());
```
