Tutorial 8

Today we will look at more advanced JUnit testing examples, including black box and white box testing processes as well as boundary test cases.

Part A)

The following questions do not require the Java code to complete:

Question 1: What is the difference between "black box" and "white box" testing methods?

Black Box: "data driven testing." We use the program specification to find cases where the program does not conform to the design specifications, typically by testing many inputs. We do this without having any knowledge of how the program itself actually executes (we do not see the code).

White Box: Test the program's *logic*, we can look at how the code works and design tests that will ensure that the code is executing properly and without faults.

Question 2: What are the "gold standard" practices for black box and white box testing?

Black Box: Exhaustively testing inputs.

White Box: Exhaustively testing execution paths.

Part B)

The remainder of these questions require you to set up the tut8 project.

Project Setup Instructions

1. Download tut8q.jar from the course website
2. Download blackbox.jar from the course website
3. In Eclipse, start a new project
4. Right click the project and choose "Properties" from the dropdown menu
5. Click "Java Build Path" on the list to the left of the new window.
6. Choose the "Libraries" tab.
7. Use "Add Library" to add JUnit 4 to the project.
8. Use "Add External JARs" to add blackbox.jar to the project.
9. Right click on "src" and choose "import" from the dropdown menu
10. Choose General -> Archive File and press Next
11. Browse to tut8q.jar and select it
12. Press Finish

For the following questions, use the "Q1" package. Have a look at the Person.java class. You will use the code to inform how you should improve the three tests defined in PersonTest.java.

Question 1a: Alter Question 1a so that it catches the IllegalArgumentException raised by the Person class. Use the @Test "expected" annotation to indicate to JUnit which exception you need to catch.

```
@Test(expected = IllegalArgumentException.class)
public void testExpectedException1() {
	new Person("John",-1);
}
```

Question 1b: Alter Question 1b so that it catches the IllegalArgumentException raised by the Person class. Use the @Rule annotation and describe an ExpectedException object catch the exception and verify that the message returned is "Invalid age: -1"

```
@Rule
public ExpectedException exception = ExpectedException.none();

@Test
public void testExpectedException2() {
	exception.expect(IllegalArgumentException.class);
	exception.expectMessage("Invalid age: -1");

	new Person("John",-1);
}
```

Question 1c: Alter Question 1c so that it catches the IllegalArgumentException raised by the Person class. Use a traditional "Try – Catch" block to catch the exception.

```
@Test
public void testExpectedException3() {
	try {
		new Person("John",-1);
		fail("Should throw IllegalArgumentException for age: -1");
	} catch (IllegalArgumentException e) {
		assertEquals("Invalid age: -1", e.getMessage());
	}
}
```

Continued next page.

For the following questions, use the "Q2" package. Make sure you understand how the "getLetterGrade" method works, and then answer the questions below to improve the tests in GradeTest.java.

Question 2a: Implement the testB, testC and testD methods.

```
@Test
public void testB() {
	String grade = Grade.getLetterGrade(70);
	assertEquals("B", grade);
}

@Test
public void testC() {
	String grade = Grade.getLetterGrade(55);
	assertEquals("C", grade);
}

@Test
public void testD() {
	String grade = Grade.getLetterGrade(40);
	assertEquals("D", grade);
}
```

Question 2b: There has been a serious error reported about the getLetterGrade method returning the wrong grade letter for some students. Why is this? As part of your answer, write a test to expose the error and adjust the code until it passes.

For values between 50 and 55, getLetterGrade returns "A" when it should return "C-"

```
@Test
public void testCMinus() {
	String grade = Grade.getLetterGrade(50);
	assertEquals("C-", grade);
}
```

Question 2c: ***Boundary Testing*** – The getLetterGrade method has a boundary at "100". Implement the tests required to adequately test this boundary. Does the code correctly work around this boundary? Be sure to implement and catch any exceptions you think are necessary.

```
@Test
public void testUnderBoundary() {
	String grade = Grade.getLetterGrade(99);
	assertEquals("A+", grade);
}

@Test
public void testBoundary() {
	String grade = Grade.getLetterGrade(100);
	assertEquals("A+", grade);
}

@Test(expected = IllegalArgumentException.class)
public void testOverBoundary() {
	String grade = Grade.getLetterGrade(101); // Should raise exception
}
```

For the following questions, use the "Q3" package. The Performance.prime method is used to get the n^{th} prime, however it is taking a long time to run and consequently messing up the test suite.

Question 3a: Add a timeout to the test case using the "timeout" parameter and assert that the output is "7".

```
@Test(timeout = 500)
public void test() {

	int result = Performance.prime(4); // Find the third prime.
	assertEquals(7, result);

}
```

Question 3b: How can you fix the Performance.prime() method to make it work correctly?

Complete the code in the while body:

```
// Completed code:
if (foundPrimes == n)
	return prime;
```

For the following questions, use the "Q4" package. You are tasked with testing this package and ensuring that it works properly, but nobody knows what it does. Your first task is to figure out what it actually does. You can use the eclipse IDE to help understand what methods are inside the package and what their inputs and outputs are.

Question 4a: Write a test that gives BlackBox some input, and checks the output.

```
@Test
public void testRun() {

	String[] array = {"banana","pear","apple"};
	List<String> result = BlackBox.Run(array);

	assertEquals("apple",result.get(0));
	assertEquals("banana",result.get(1));
	assertEquals("pear",result.get(2));

}
```

Continued next page.

Question 4b: What happens if you give BlackBox one input? Write a test to check this condition.

```
@Test
public void testSingleInput() {

	String[] array = {"apple"};
	List<String> result = BlackBox.Run(array);

	assertEquals("apple",result.get(0));

}
```

Question 4c: Write one other test that checks some other behaviour of the BlackBox.

```
@Test
public void testSortingIndex() {

	String[] array = {"banana","lime","lemon","apple"};
	List<String> result = BlackBox.Run(array);

	// If sorting alphabetically including second index,
	// lime should come after lemon.
	assertEquals("apple",result.get(0));
	assertEquals("banana",result.get(1));
	assertEquals("lemon",result.get(2));
	assertEquals("lime",result.get(2));

}
```