Unit Testing with JUnit

CSE 1020

moodle.yorku.ca

Unit testing

A unit test is designed to test a single unit of code, for example, a method.

Such a test should be automated as much as possible; ideally, it should require no human interaction in order to run, should assess its own results, and notify the programmer only when it fails.

A class that contains unit tests is known as a test case.

The code to be tested is known as the unit under test.

JUnit

JUnit is a Java unit testing framework written by Kent Beck and Erich Gamma.

JUnit is available at www.junit.org.

Kent Beck

Kent Beck is an American software engineer and the creator of the Extreme Programming and Test Driven Development software development. He works at Facebook.

source: Three Rivers Institute

Erich Gamma

Erich Gamma is a Swiss computer scientist and member of the "Gang of Four" who wrote the influential software engineering textbook "Design Patterns: Elements of Reusable Object-Oriented Software." He works at Microsoft.

source: Pearson

Java annotations

Annotations provide data about code that is not part of the code itself. Therefore, it is also called metadata.

In its simplest form, an annotation looks like

```
@Deprecated
```

(The annotation type `Deprecated` is part of `java.lang` and, therefore, need not be imported.)

An annotation can include elements and their values:

```
@Test(timeout=1000)
```

(The annotation type `Time` is part of `org.junit` and, therefore, needs to be imported.)

A test case

```
import org.junit.Assert;
import org.junit.Test;

public class ... {
    @Test
    public void ...() {
        ...
    }

    @Test
    public void ...() {
        ...
    }

    ...
}
```

Names of test methods

It is good practice to use descriptive names for the test methods. This makes tests more readable when they are looked at later.

Assertions in test methods

Each test method should contain (at least) one assertion: an invocation of a method of the `Assert` class of the `org.unit` package.

Do not confuse these assertions with Java's `assert` statement.

Methods of the Assert class

`assertEquals(long, long)`

assert that the two are the same.

`assertEquals(String, long, long)`

assert that the two are the same; if not, the message is used.

Methods of the Assert class

```
assertEquals(double, double, double)
assertEquals(String, double, double, double)
```

The method invocation

```
Assert.assertEquals(expectedValue, actualValue, delta)
```

asserts

$$|\texttt{expectedValue} - \texttt{actualValue}| < \texttt{delta}$$

Methods of the Assert class

`assertEquals(Object, Object)`
`assertEquals(String, Object, Object)`

asserts that the objects are equal according to the `equals` method.

`assertSame(Object, Object)`
`assertSame(String, Object, Object)`

asserts that the objects are equal according to the == operator.

Methods of the Assert class

```
assertTrue(boolean)
assertTrue(String, boolean)
```

asserts that the condition is true.

```
assertFalse(boolean)
assertFalse(String, boolean)
```

asserts that the condition is false.

Methods of the Assert class

`assertNull(Object)`
`assertNull(String, Object)`

asserts that the object is null.

`assertNotNull(Object)`
`assertNotNull(String, Object)`

asserts that the object is not null.

Timeout

Cause a test to fail if it takes longer than a specified time in milliseconds:

```
@Test(timeout=1000)
public void ...() {
    ...
}
```

Exceptions

Cause a test to fail if a specified exception is not thrown:

```
@Test(expected=NumberFormatException.class)
public void ...() {
    ...
}
```

Body of unit test method

1. Create some objects.
2. Invoke methods on them.
3. Check the results using a method of the `Assert` class.

Test case

For each method and constructor (from simplest to most complex)

1. Study its API.
2. Create unit tests.

Writing a test suite

A test suite comprises one or more tests, grouping them so that they can be run together.

Create a test suite using the `@RunWith` and `@Suite.SuiteClasses` annotations. Both annotations are part of the packages `org.junit.runner` and `org.junit.runners` and, hence, need to be imported.

Writing a test suite

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({....class, ....class})
public class ... { }
```

JUnit and Eclipse

To add JUnit to a project, select its properties (select "Properties" from the "Project" menu option) and select the "Java Build Path," "Libraries" tab. Press the "Add Library ..." button and then choose "JUnit." Click the "Next" button, and on the next dialog select "JUnit 4" from the drop-down list.

A JUnit test case class can be run by right-clicking on the test class and selecting "Run As ..." and "JUnit Test."

Fraction

Exercise

Test the `Fraction` class of the `franck.cse1020` package.

Testing a main method

```
@Test
public void test() {
  // command line arguments
  String[] args = {};
  // input given by the user via the keyboard
  String user = "...";

  // set up input and output
  ByteArrayInputStream input =
    new ByteArrayInputStream(user.getBytes());
  System.setIn(input);
  ByteArrayOutputStream output =
    new ByteArrayOutputStream();
  PrintStream stream = new PrintStream(output);
  System.setOut(stream);
```

Testing a main method

```
    // call the main method
    ClassName.main(args);

    // verify the output
    String expected = "...";
    String actual = output.toString();
    Assert.assertEquals(expected, actual);
}
```

Test 5

- When: Friday February 28, during the lab (14:30–16:00)
- Where: Lassonde building, labs 1006, 1004, 1002
- Material: Chapter 1–6 of the textbook, with a focus on Chapter 5 and 6
- What: Two programming questions
- Note: For each question, you get 1 mark for the fact that your code compiles
- Note: Your code is not only marked for correctness but also style (1 mark for each question)