JUnit

- Introduction
- A JUnit Example
- Major APIs
- Practical Tips

Unit Testing

- Test individual units of source code in isolation
  - Procedures, functions, methods, and classes
- Engineers do not only write code, but also write tests to show the code works!
- Benefits
  - Allows faults to be detected and fixed early
  - Facilitates refactoring and regression testing
  - Serves as a formal specification of the intended behavior

JUnit

- A simple, open-source framework to write and run repeatable unit tests
  - Inspired many similar frameworks, i.e., the xUnit family (NUnit, CppUnit, HttpUnit, and others)
- Major features
  - Assertions for checking expected results
  - Test fixtures for sharing common test data
  - Test runners for automatically running tests

History

- 1994: SUnit by Kent Beck
- 1997: First version of JUnit by Kent Beck and Erich Gamma (on a flight from Zurich to Atlanta)
- 1998: JUnit 1.0 (public release)
- 2006: JUnit 4.0 (a major update)
- 2017: JUnit 5 (latest release)

JUnit

- Introduction
- A JUnit Example
- Major APIs
- Practical Tips

Major Steps

- Write the class to be tested
- Write a test class
- Write test setup methods if needed
- Write test methods which are annotated with @test
- Write test teardown methods if needed
- Use a test runner to run the tests

TriangleClassifier

```
1 public class TriangleClassifier {
2     public final static int INVALID = -1;
3     public final static int SCALENE = 0;
4     public final static int ISOCELES = 1;
5     public final static int EQUILATERAL = 2;
6     private int side1;
7     private int side2;
8     private int side3;
9
10    public TriangleClassifier(int side1, int side2, int side3) {
11        this.side1 = side1;
12        this.side2 = side2;
13        this.side3 = side3;
14    }
15    public int classify () {
16        int rval = 0;
17
18        if (side1 <= 0 || side2 <= 0 || side3 <= 0) {
19            rval = INVALID;
20        }
21
22        if ((side1 + side2 <= side3) || (side1 + side3 <= side2)
23            || (side2 + side3 <= side1)) {
24               rval = INVALID;
25        }
26
27        if ((side1 != side2) && (side2 != side3) && (side1 != side3)) {
28            rval = SCALENE;
29        } else if (( side1 == side2) || (side2 == side3) || (side1 == side3)) {
30            rval = ISOCELES;
31        } else {
32            rval = EQUILATERAL;
33        }
34
35        return rval;
36    }
37 }
```

TriangleClassifierTest

```java
import org.junit.*;
import org.junit.runner.*;
import org.junit.runner.notification.Failure;
import static org.junit.Assert.*;
import java.util.List;

public class TriangleClassifierTest {
    public static void main (String[] args) {
        Result result = JUnitCore.runClasses (TriangleClassifierTest.class);
        List<Failure> failures = result.getFailures();
        for (Failure failure : failures) {
            System.out.println(failure);
        }
    }

    @Before
    public void setUp () {
    }
    @After
    public void tearDown () {
    }

    @Test
    public void checkInvalid () {
        TriangleClassifier classifier = new TriangleClassifier (3, 4, 8);
        assertEquals(TriangleClassifier.INVALID, classifier.classify ());
    }

    @Test
    public void checkScalene () {
        TriangleClassifier classifier = new TriangleClassifier (4, 5, 6);
        assertEquals(TriangleClassifier.SCALENE, classifier.classify ());
    }

    @Test
    public void checkIsoceles () {
        TriangleClassifier classifier = new TriangleClassifier (3, 3, 4);
        assertEquals(TriangleClassifier.ISOCELES, classifier.classify ());
    }

    @Test
    public void checkEquilateral () {
        TriangleClassifier classifier = new TriangleClassifier (3, 3, 3);
        assertEquals(TriangleClassifier.EQUILATERAL, classifier.classify ());
    }
}
```

Test Method Execution

- Each test method is executed in a new instance of the test class
  - Common test data cannot be shared by using instance members
- By default, test methods are executed in a deterministic, but unpredictable, manner
  - Use @FixMethodOrder to fix the execution order, e.g., in the ascending name order

JUnit

- Introduction
- A JUnit Example
- Major APIs
- Practical Tips

Core Packages

- org.junit: core classes and annotations
  - @Test, @Before, @After, @BeforeClass, @AfterClass, @Ignore
  - Assert, Assume
- org.junit.runner: classes to run and analyze tests
  - JUnitCore, Runner, Result, @RunWith
- org.junit.runner.notification: provides information about a test run
  - Failure, RunListener
- org.junit.runners: provide standard implementations of test runners
  - BlockJUnit4ClassRunner (the default runner), Suite, Parameterized

Other Packages

- org.hamcrest: a 3rd-party package that contains APIs for various matchers
- org.hamcrest.core: provides implementations for fundamental matchers
- org.junit.matchers: provides additional matchers that are specific to unit testing

@Before/@After

- Indicates a method to be executed before/ after each test

```
public class Example {
   File output;
   @Before public void createOutputFile() {
        output= new File(...);
   }
   @Test public void something() {
        ...
   }
   @After public void deleteOutputFile() {
        output.delete();
   }
}
```

@BeforeClass/@AfterClass

- Indicates a method to be executed before/ after all test methods are executed

```
public class Example {
   private static DatabaseConnection database;
   @BeforeClass public static void login() {
        database= ...;
   }
   @Test public void something() {
        ...
   }
   @Test public void somethingElse() {
        ...
   }
   @AfterClass public static void logout() {
        database.logout();
   }
}
```

Assertions

- JUnit provides overloaded assertion methods for all primitive types, Object, and arrays (of primitives or Objects).
- An assertion method typically takes three parameters
  - An error message (of String type, optional), expected value, and actual value
- Typically imported through a static import
  - import static org.junit.Assert.*
  - Assert.assertEquals() vs assertEquals()

Assertions (2)

- Major groups of assertion methods
  - AssertArrayEquals
  - AssertEquals/AssertNotEquals
  - AssertSame/AssertNotSame
  - AssertNull/AssertNotNull
  - AssertTrue/AssertFalse
  - AssertThat
  - Fail

assertThat and matchers

- Allows to write more readable assertions
  - assertThat (x, is(3));
  - assertThat (x, is(not(4)));
  - assertThat (responseString, either(containsString("color").or(containsString("colour"))));
  - assertThat(myList, hasItem("3"));
- Matchers can be negated and/or combined together

assertThat and matchers (2)

```
assertTrue(responseString.contains("color") ||
                responseString.contains("colour"));
// ==> failure message:
// java.lang.AssertionError:

assertThat(responseString, either(containsString("color").or(
                containsString("colour"))));
// ==> failure message:
// java.lang.AssertionError:
// Expected: (a string containing "color" or a string containing "colour")
//  got: "Please choose a font"
```

Exception Test

- How to verify that code throws exceptions as expected?

```
@Test(expected= IndexOutOfBoundsException.class)
public void empty() {
    new ArrayList<Object>().get(0);
}
```

Exception Test (2)

```
@Test
public void testExceptionMessage() {
    try {
        new ArrayList<Object>().get(0);
        fail("Expected an IndexOutOfBoundsException to be thrown");
    } catch (IndexOutOfBoundsException ex) {
        assertEquals("Index: 0, Size: 0”, ex.getMessage());
    }
}
```

Assume

- States a condition that must be satisfied for a test to be meaningful

```
// only provides information if database is reachable.
@Test public void calculateTotalSalary() {
   DBConnection dbc = Database.connect();
   assumeNotNull(dbc);
   // ...
}
```

JUnitCore

- A façade for running tests either from command line or programmatically
  - public static void main (String… args)
  - public static Result runClasses (Class<?>… classes)

Suite Runner

- A runner that allows to build a suite that contains tests from many classes

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

@RunWith(Suite.class)
@Suite.SuiteClasses ({
  TriangleClassifierTest.class
})

// this class is just a place holder
public class SuiteTest {
}
```

Parameterized Runner

- A custom runner that implements parameterized tests.

JUnit

- Introduction
- A JUnit Example
- Major APIs
- Practical Tips

Practical Tips

- Use the right Assertions in order to make code more readable and/or get more meaningful error messages
  - assertTrue (!(a < 3)) vs assertFalse (a < 3)
  - assertTrue (a.equals(b)) vs assertEquals(a, b)
- Minimize dependencies from other tests and/or components
- Only write tests for methods that are significant and likely to change in the future
  - Typically no tests are needed for getters/setters

Practical Tips (2)

- Do not over- or under-specify tests
  - Tests that are brittle to changes or that do not provide sufficient checks
- Try to achieve sufficient test coverage
  - 100% is not necessary, but as a rule of thumb, 90% or more for functionally important classes, and 80% or more for general classes
- Update test code along with production code, but package them separately

References

- Simple SmallTalk Testing: With Patterns, http://www.xprogramming.com/testfram.htm
- JUnit 4.x Quick Tutorial, https://code.google.com/p/t2framework/wiki/JUnitQuickTutorial
- JUnit JavaDoc, http://junit.org/javadoc/latest/
- JUnit Practical Tips, http://www.deepakgaikwad.net/index.php/2009/10/21/practicaljunittips.html