Test with annotation JUnit4

Advanced Programming

Installing JUnit at home

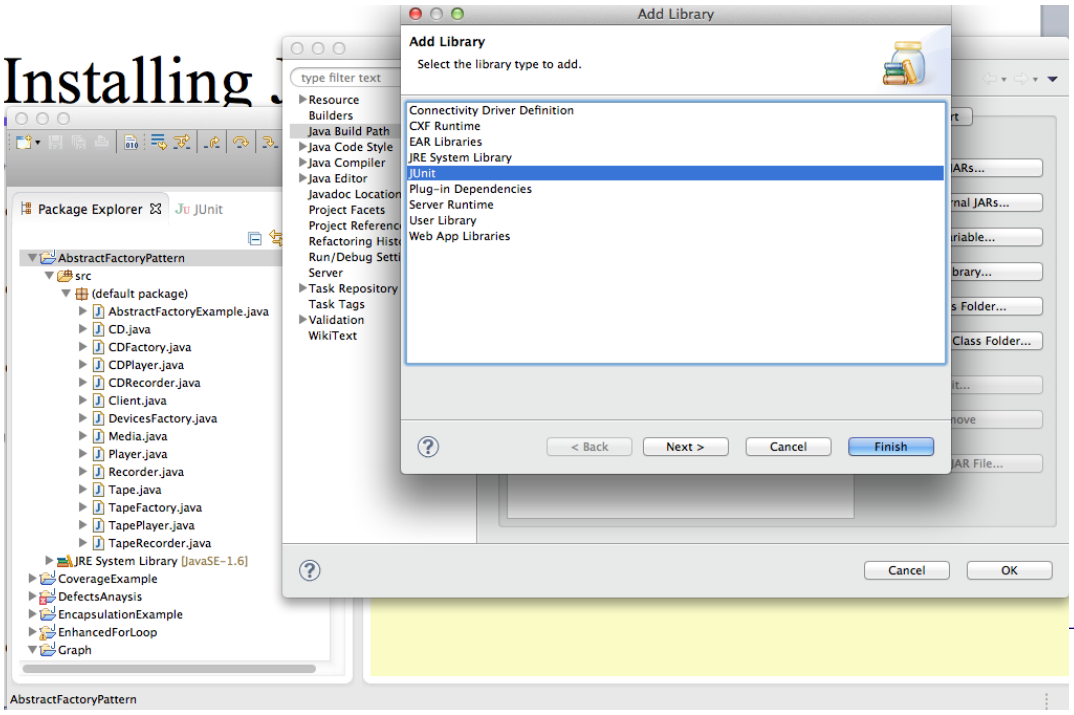
- Download from http://junit.org the jar file
- Read the API http://junit.org/junit4/javadoc/latest/
- Unzip or import the jar file
- Add to classpath of your project
  - Project → Properties → Java Build Path
  - Libraries tab
  - Add library

Installing JUnit at home

Right click on the project folder in package explorer

- BuildPath → add Libraries tab
- Or
- BuildPath →in the library tab select add library

Installing JUnit



Example

```
public class Calculator {
  public int evaluate(String expression) {
    int sum = 0;
    for (String summand: expression.split("\\+"))
      sum += Integer.valueOf(summand);
    return sum;
  }
}
```


Example

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

public class CalculatorTest {
  @Test
  public void evaluatesExpression() {
    Calculator calculator = new Calculator();
    int sum = calculator.evaluate("1+2+3");
    assertEquals(6, sum);
  }
}
```

setUp() & tearDown()

- When a test class contains multiple methods to test, you can use the **setup()** and **tearDown()** methods to initialise and release any common objects under test

Why annotations

- The traditional way to indicate test methods in JUnit 3 is by prefixing their names with “test”
- This is a very effective method for tagging certain methods in a class as having a special meaning, but the naming doesn’t scale very well and is rather inflexible (what if we want to pass additional parameters to the test?).

Benefits of Using Annotations

- Annotations were formally added to the Java language in JDK 5
- Tagging methods with annotations
  - Hence, method names are not restricted to any pattern or format.
- We can pass additional parameters to annotations

Benefits of Using Annotations

- Annotations are strongly typed, so the compiler will flag any mistakes right away
- Test classes no longer need to extend anything (such as TestCase, for JUnit 3).

Basic annotations

- **@Parameters** Describes how to pass parameters to a @Test method.
- **@Test** Marks a class or a method as a part of the test.

@Test

- It tells that the **public method that returns void** to which it is attached can be run as a test case
- JUnit first constructs a new instance of the class then invokes the annotated method
- Any exceptions thrown by the test will be reported as a failure
- If no exceptions are thrown, the test succeeds

Example

```
public class Example {
    @Test
    public void method() {
       org.junit.Assert.assertTrue(new ArrayList().isEmpty());
    }
```

Test Fixtures

- A **test fixture** is a fixed state of a set of objects used as a baseline for running tests.
- JUnit provides annotations so that test classes can have fixture run before or after every test, or one time fixtures that run before and after only once for all test methods in a class.
- There are four fixture annotations: two for class-level fixtures and two for method-level ones. At the class level, you have **@BeforeClass and @AfterClass**, and at the method (or test) level, you have **@Before and @After**

Test Fixtures

- **@BeforeClass** The annotated method will be run before any test method in the current class is invoked.
- **@AfterClass** The annotated method will be run after all the test methods in the current class have run.

@After and @Before

- Annotating a public method that returns void with @Before (or @After) causes that method to be run before (or after) the Test method
- The @Before (or @After) methods of superclasses will be run before (or after) those of the current class
- All @After methods are guaranteed to run even if a Before or Test method throws an exception

Major annotations - test fixture

- **@BeforeMethod** The annotated method will be run before each test method
- **@AfterMethod** The annotated method will be run after each test method

Example

```
public class Example {
   List myList;
   @Before
    public void initialize() {
        myList= new ArrayList();
   }
   @Test
    public void testSize() {
      //it uses myList
   }
   @Test
    public void testRemove() {
      //it uses myList
   }
}
```

Optional parameters of @Test

- **Expected and Timeout**
- **Expected:** check a test method throws the expected exception
- If it doesn't throw an exception or if it throws a different exception than the one declared, the test fails

Example: test succeeds

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

ExpectedException rule

- Careful! The @Test with expected exception will pass if any code in the method throws IndexOutOfBoundsException
- For longer tests,use the ExpectedException rule

ExpectedException rule

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

@Test
public void shouldTestExceptionMessage() throws
IndexOutOfBoundsException {
    List<Object> list = new ArrayList<Object>();
    thrown.expect(IndexOutOfBoundsException.class);
    thrown.expectMessage("Index: 0, Size: 0”);
    thrown.expectMessage(JUnitMatchers.containsString("Size: 0"));
   // JUnit matcher that finds a string in the exception message
}
```

Optional parameters of @Test

- **Timeout** causes a test to fail if it takes longer than a specified amount of clock time (measured in milliseconds)

```
@Test(timeout=100)
    public void infinity() {
        while(true);
    }
```

Annotating a class with @RunWith

- When a class is annotated with @RunWith or extends a class annotated with @RunWith, JUnit will invoke the class it references to run the tests in that class instead of the runner built into JUnit.

The test suite class with annotation

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

@RunWith(Suite.class)
@SuiteClasses(ATest.class, BTest.class, CTest.class)

public class ABCSuite {
}
```


@SuiteClasses

- The @SuiteClasses annotation specifies the classes to be executed when a class annotated with @RunWith(Suite.class) is run

- see code LECT8

Exercise

- Build test cases for class Bag