CS 5154: Software Testing

Test Automation Framework

Owolabi Legunsen
Check-in and announcements

• Partner finding social is today

• TAs will hold “lab” sessions this week (and maybe next)

• In class quiz 9/8 or 9/13

• Homework 0 will be assigned soon
About this slide deck...

• These slides are posted here for your review

• We cover the material in these slides during the demo
Recall: components of a test

• Test case values (or, input values)

• Expected results

• A test oracle
JUnit: A test automation framework

• JUnit is open source: (https://junit.org)

• JUnit is widely used

• JUnit can be used
  • with standalone Java programs
  • within an IDE
  • within a build system like Maven
Some JUnit features that we saw in the demo

• Assertions for testing expected results

• Parameterized tests for sharing code among different inputs

• Test classes containing
  • a collection of test methods
  • Test fixtures: methods to set up state before, and to update state after tests

• Test runners for running the tests and reporting the results
  • e.g., org.junit.runner.JUnitCore
JUnit Test Fixtures

- A **test fixture** is the **state** of the test
  - Objects and variables that are used by more than one test
  - Initializations (*prefix* values)
  - Reset values (*postfix* values)
- Different tests can **use** the objects without sharing the state
- Objects used in test fixtures should be declared as **instance variables**
- They should be initialized in a **@Before** (or **@BeforeClass**) method
- Can be deallocated or reset in an **@After** (or **@AfterClass**) method
Simple JUnit Example

```java
public class Calc
{
    static public int add (int a, int b)
    {
        return a + b;
    }
}

import org.junit.Test;
import static org.junit.Assert.*;

public class CalcTest
{
    @Test public void testAdd()
    {
        assertTrue ("Calc sum incorrect",
                    5 == Calc.add (2, 3));
    }
}
```

Note: JUnit 4 syntax
Testing the Min Class

import java.util.*;

public class Min {
    /**
     * Returns the minimum element in a list
     * @param list Comparable list of elements to search
     * @return the minimum element in the list
     * @throws NullPointerException if list is null or
     *         if any list elements are null
     * @throws ClassCastException if list elements are
     *         not mutually comparable
     * @throws IllegalArgumentException if list is empty
     */
    ...
}
Testing the Min Class (2)

```java
public static <T extends Comparable<? super T>> T min (List<? extends T> list)
{
    if (list.size() == 0)
    {
        throw new IllegalArgumentException ("Min.min");
    }
    Iterator<? extends T> itr = list.iterator();
    T result = itr.next();

    if (result == null) throw new NullPointerException ("Min.min");

    while (itr.hasNext())
    {
        // throws NPE, CCE as needed
        T comp = itr.next();
        if (comp.compareTo (result) < 0)
        {
            result = comp;
        }
    }
    return result;
}
```
MinTest Class

Standard imports for all JUnit classes:

```java
import static org.junit.Assert.*;
import org.junit.*;
import java.util.*;
```

Test fixture and pre-test setup method (prefix):

```java
private List<String> list;   // Test fixture

// Set up - Called before each test method
@Before
public void setUp() {
    list = new ArrayList<String>();
}
```

Post test teardown method (postfix):

```java
// Tear down - Called after every test method.
@After
public void tearDown(){
    list = null;   // redundant in this example
}
Min Test Cases: NullPointerException

```java
@Test public void testForNullList()
{
    list = null;
    try {
        Min.min (list);
    } catch (NullPointerException e) {
        return;
    }
    fail ("NullPointerException expected");
}

@Test (expected = NullPointerException.class)
public void testForNullElement()
{
    list.add (null);
    list.add ("cat");
    Min.min (list);
}

@Test (expected = NullPointerException.class)
public void testForSoloNullElement()
{
    list.add (null);
    Min.min (list);
}
```
More Exception Test Cases for Min

```java
@Test (expected = ClassCastException.class)
@SuppressWarnings ("unchecked")
public void testMutuallyIncomparable()
{
    List list = new ArrayList();
    list.add ("cat");
    list.add ("dog");
    list.add (1);
    Min.min (list);
}
```

Note that Java generics don’t prevent clients from using raw types!

Special case: Testing for the empty list

```java
@Test (expected = IllegalArgumentException.class)
public void testEmptyList()
{
    Min.min (list);
}
```

```java
@Test (expected = IllegalArgumentException.class)
public void testEmptyList()
{
    Min.min (list);
}
```
Finally! A couple of “Happy Path” tests

Remaining Test Cases for Min

```java
@Test
public void testSingleElement() {
    list.add("cat");
    Object obj = Min.min(list);
    assertTrue("Single Element List", obj.equals("cat"));
}

@Test
public void testDoubleElement() {
    list.add("dog");
    list.add("cat");
    Object obj = Min.min(list);
    assertTrue("Double Element List", obj.equals("cat"));
}
```
Data-Driven Tests

- **Problem**: Testing a function multiple times with similar values
  - How to avoid test code bloat?

- **Simple example**: Adding two numbers
  - Adding a given pair of numbers is just like adding any other pair
  - You only want to write one test

- **Parameterized** unit tests call a constructor for each collection of test values
  - Same tests are then run on each set of data values
  - Collection of data values defined by method tagged with `@Parameters` annotation
Example JUnit parameterized unit test

```java
import org.junit.*;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
import static org.junit.Assert.*;
import java.util.*;

@RunWith(Parameterized.class)
public class DataDrivenCalcTest
{
  public int a, b, sum;
  public DataDrivenCalcTest (int v1, int v2, int expected)
  { this.a = v1; this.b = v2; this.sum = expected; }

  @Parameters public static Collection<Object[]]> parameters()
  { return Arrays.asList (new Object [][] {{1, 1, 2}, {2, 3, 5}});
  }

  @Test public void additionTest()
  { assertTrue ("Addition Test", sum == Calc.add (a, b)); } 
}
```

Constructor is called for each triple of values

Test 1
Test values: 1, 1
Expected: 2

Test 2
Test values: 2, 3
Expected: 5

Test method
How to Run Tests

• JUnit provides test drivers
  • Character-based test driver runs from the command line
  • GUI-based test driver-\texttt{junit.swingui.TestRunner}
    • Not covered in this course

• If a test fails, JUnit gives the location of the failure and any exceptions that were thrown
JUnit Resources

• There are many JUnit tutorials on the Internet

• JUnit: Download, Documentation
  • https://junit.org/junit4
Summary on Test Automation

• Automation makes testing **efficient** and **effective**

• Test frameworks provide very simple ways to **automate** our tests

• Not a “**silver bullet**” ... it does not solve a hard problem in testing:

  What test values to use?

• This is test design ... the purpose of **test criteria**