EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Introduction to JUnit

Data Structures and Algorithms for Language Processing

What is JUnit

- JUnit is a small, but powerful Java framework to create and execute automatic unit tests
- Unit testing is the test of a part of a program unit in isolation of other units of your code
- With a single glance you can see if something is wrong with your code
- The size of the program unit may be a method, a class or several classes that represent a component

Why use JUnit Testing – 1

- Write code faster while increasing code quality
  - When you write JUnit tests you spend less time debugging, because you always see the effect of a change in your code.
  - You are testing small parts of your code where you can understand and track down a bug easier and faster.

Why use JUnit Testing – 2

- Immediate feedback
  - You can immediately see a problem. With `System.out.println()`, you manually have to compare actual and expected results (error prone and slow).
  - JUnit simply shows a red bar when there is a problem or a green bar when everything is ok.

Why use JUnit Testing – 3

- JUnit tests don't take much time or effort
  - Writing tests is fairly easy.
  - You simply define what is going into a method and look at the expected results.
  - Then click a button and you know if things are alright.

Create a New Test Case

- Create the class you want to test – as usual.
- To create a JUnit test for that class, you select **File→New JUnit Test Case** in DrJava's Menu.
- DrJava asks for the name of the test class
  - Use the name of your class you want to test followed by **`Test`** (do not include ".java"), e.g., if **`Purse`** is our class, name the test class **`PurseTest`**
  - DrJava creates a new document that looks like the example on the next slide.

Create a New Test Case

```
import junit.framework.TestCase;

/**
 * A JUnit test case class.
 */
public class PurseTest extends TestCase {

    /**
     * A test method
     */
    public void testX() {
    }
}
```

- The example will test parts of the **`Purse`** class from Wednesday's selftest.

Create a New Test Case

- Replace "**`X`**" in the method name **`testX()`** with a name describing the test.
- You may write as many **`testSomething()`** methods in a JUnit class as you wish.
- Every method starting with the word "**`test`**" will be called when running the test with Junit.

Compile and Run a Test Case

- Store the test with **File→Save As...**
- Click on the **Compile** button to compile the program.
- Then run **Tools→Test Current document** – or click on the **Test** button.
- The test should succeed, i.e., the `testX()` method evaluates to true because there is nothing that can fail.
- **Whenever you change something in your test case or your application: recompile and test!**

Writing Test Methods

- The following method tests an empty purse:

```
public void testEmpty() {
    Purse myPurse = new Purse();
    assertNotNull("testEmpty", myPurse);
    assertEquals(0, myPurse.getTotal(), 0.00001);
}
```

- Remember: if the difference between two floating point numbers is very small, consider them equal.

Writing Test Methods

- Testing with JUnit is implemented by doing assertion.
- **`assertNotNull("testEmpty", myPurse)`**
  asserts (checks) **`myPurse`** must not be **null**.
  - If **`myPurse`** is **null** the assertion fails and we know that there is a problem in our code.
- **`assertEquals(0, myPurse.getTotal(), 0.00001)`**
  asserts that **`myPurse.getTotal()`** must be 0, because we havent added any money to our purse yet.
  - The assertion fails if **`myPurse.getTotal()`** does not return 0.

Writing Test Methods

- Download **`PurseTest.java`** and **`Purse.java`** (see Wednesday's selftest)
- For each of the **`addX()`** methods of the **`Purse`** class, there is a corresponding test method **`testAddX()`** in **`PurseTest`** that inspects the method itself.
- Notice that at the beginning of each test, a spearate **`Purse`** variable is created:

```
Purse myPurse = new Purse();
```

Writing Test Methods

- Example: try to put money into our purse to test method **`addOneEuroCoins`**

```
public void testAddOneEuroCoins() {
    Purse myPurse = new Purse();

    myPurse.addOneEuroCoins(1);
    assertEquals(1.00, myPurse.getTotal(), 0.00001);

    myPurse.addOneEuroCoins(100);
    assertEquals(101.00, myPurse.getTotal(), 0.0001);
}
```

List of Available Assert Methods

- You can find a list of available assert methods in the JUnit documentation: http://kentbeck.github.com/junit/javadoc/latest/org/junit/Assert.html