Testing and JUnit

Testing
- Obviously you have to test your code—right?
  - You can do ad hoc testing (running whatever tests occur to you at the moment), or
  - You can build a test suite (a thorough set of tests that can be run at any time)
- Disadvantages of a test suite
  - It’s a lot of extra programming
    - True, but use of a good test framework can help quite a bit
  - You don’t have time to do all that extra work
    - False! Experiments repeatedly show that test suites reduce debugging time more than the amount spent building the test suite
- Advantages of a test suite
  - Reduces total number of bugs in delivered code
  - Makes code much more maintainable

Types of testing
- Unit testing: Does a single object work?
- Integration testing: Do multiple objects work together?
- System testing: Does my application work?
- Performance testing: Does my application work well?
- Acceptance testing: Does the customer like my application?

JUnit
- JUnit test framework is a package of classes that lets you write tests for each method, then easily run those tests (unit testing)
- Martin Fowler: “Never in the field of software development was so much owed by so many to so few lines of code.”
- JUnit has become the standard tool for test-driven development in Java
- JUnit test generators available in Eclipse

Example – JUnit v.3
```java
import junit.framework.TestCase;
public class TestString extends TestCase {
    public void testEndsWith() { 
        assertTrue("abcabc".endsWith("abc");
    }
    public void testEndsWithEmptyString() { 
        assertTrue("abcabc".endsWith(""));
    }
}
```

Example – JUnit v.4
```java
import org.junit.Test;
import static org.junit.Assert.assertTrue;

public class TestString4 {
    @Test
    public void testEndsWith() { 
        assertTrue("abcabc".endsWith("abc");
    }
    @Test
    public void testEndsWithEmptyString() { 
        assertTrue("abcabc".endsWith(""));
    }
}
```
Digression - annotations

- Annotations: a new feature in Java v.5
- Annotations are often used by frameworks as a way of conveniently applying behavior to classes and methods

http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html

Digression – import static

- Importing static methods/variables (new feature in v.5)
- Syntax:
  ```java
  import static package.Class.staticMember;
  ```
  Importing all static members of a class
  ```java
  import static package.Class.*;
  ```
- Example:
  ```java
  import static java.lang.Math.*;
  double r = cos(PI * theta);
  // instead of: Math.cos(Math.PI * theta);
  ```
  Use sparingly! (namespace pollution)

http://java.sun.com/j2se/1.5.0/docs/guide/language/static-import.html

Example

Show the account example

Testing

Testing follows programming by contract paradigm:
- Set up preconditions
- Exercise functionality being tested
- Check postconditions

Example:
```java
public void testEmptyList() {
    ArrayList<Object> emptyList = new ArrayList<Object>();
    assertEquals("Size of an empty list should be zero.", 0, emptyList.size());
    assertTrue("An empty list should report empty.", emptyList.isEmpty());
}
```

Assert methods

- Each assert method has parameters like: message, expected-value, actual-value
- Assert methods dealing with floating point numbers get an additional argument, a tolerance
- Each assert method has an equivalent version that does not take a message – however, this is not recommended because:
  - messages helps document the tests
  - messages provide additional information when reading failure logs

- `assertTrue(String message, Boolean test)`
- `assertFalse(String message, Boolean test)`
- `assertNull(String message, Object object)`
- `assertNotNull(String message, Object object)`
- `assertSame(String message, Object expected, Object actual)` (uses == operator)
- `assertNotSame(String message, Object expected, Object actual)` (uses != operator)
Conclusions

- Ideally, you should be writing test cases at the same time as writing application code.
- Test-driven development – write tests before writing the code being tested

Resources

- The Lewis book only handles v.3 of JUnit (chapter 11.4).
- The JUnit website: http://www.junit.org
- A nice tutorial on JUnit v.4: http://www.vogella.de/articles/JUnit/article.html