FACULDADE ENGENHARIA

Departamento de Informática

Unit Testing with JUnit

Why unit testing?

- **Unit testing**: Looking for errors in a subsystem in isolation.
  - Generally a "subsystem" means a particular class or object.
  - The Java library **JUnit** helps us to easily perform unit testing.
- The basic idea:
  - For a given class Foo, create another class FooTest to test it, containing various "test case" methods to run.
  - Each method looks for particular results and passes / fails.
- JUnit provides **"assert"** commands to help us write tests.
  - The idea: Put assertion calls in your test methods to check things you expect to be true. If they aren't, the test will fail.

A JUnit test class

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

public class name {
    ...

    @Test
    public void name() {  // a test case method
        ...
    }
}
```

- A method with `@Test` is flagged as a JUnit test case.
  - All `@Test` methods run when JUnit runs your test class.

JUnit assertion methods

`assertTrue(`**test**`)`	fails if the boolean test is `false`
`assertFalse(`**test**`)`	fails if the boolean test is `true`
`assertEquals(`**expected**`, `**actual**`)`	fails if the values are not equal
`assertNull(`**value**`)`	fails if the given value is *not* `null`
`assertNotNull(`**value**`)`	fails if the given value is `null`
`fail()`	causes current test to immediately fail

- Each method can also be passed a string to display if it fails:
  - e.g. `assertEquals("message", expected, actual)`

```
@Test
public void ArrayIntTest(){
  ArrayList<Integer> list = new ArrayList<Integer>();

  assertTrue("Asserts that list is empty", list.isEmpty());
  list.add(20);
  assertFalse("Asserts that list is not empty", list.isEmpty());
  assertEquals("Asserts that list has size 1", 1, list.size());
}
```

What's wrong with this?

```
public class DateTest {
    @Test
    public void test1() {
        Date d = new Date(2050, 2, 15);
        d.addDays(4);
        assertEquals(d.getYear(), 2050);
        assertEquals(d.getMonth(), 2);
        assertEquals(d.getDay(), 19);
    }

    @Test
    public void test2() {
        Date d = new Date(2050, 2, 15);
        d.addDays(14);
        assertEquals(d.getYear(), 2050);
        assertEquals(d.getMonth(), 3);
        assertEquals(d.getDay(), 1);
    }
}
```

Well-structured assertions

```
public class DateTest {
    @Test
    public void test1() {
        Date d = new Date(2050, 2, 15);
        d.addDays(4);
        Date expected = new Date(2050, 2, 19);
        assertEquals(expected, d);  // use an expected answer
    }                               // object to minimize tests

                                    // (Date must have toString
    @Test                           //  and equals methods)
    public void test2() {
        Date d = new Date(2050, 2, 15);
        d.addDays(14);
        Date expected = new Date(2050, 3, 1);
        assertEquals("date after +14 days", expected, d);
    }
}
```


HOW
WHY
WHERE
WHEN
WHERE
WHY
WHERE
WHO
WHEN
WHAT
WHAT
WHY
WHEN
WHAT
WHO
WHEN
WHERE
WHY
HOW