JUnit is a simple framework for testing Java programs

- Encourages you to develop tests as you develop code.
- Makes it easy to run test suites.
- You may even want to write the test first.
- You can download JUnit from http://junit.org

Example Problem: Representing a currency

class Money {
    private int fAmount;
    private String fCurrency;

    public Money(int amount, String currency) {
        fAmount = amount;
        fCurrency = currency;
    }

    public int amount() {
        return fAmount;
    }

    public String currency() {
        return fCurrency;
    }

    public Money add(Money m) {
        return new Money(amount()+m.amount(), currency());
    }
}

Want to test the add method - code a little - test a little

- Define a MoneyTest class that extends TestCase.
- Define a method testSimpleAdd() that:
  1. creates objects to use in the test case fixtures
  2. code which exercises the objects in the fixture
  3. code which verifies the result
Let’s see what it looks like:

```java
public class MoneyTest extends TestCase {
    public void testSimpleAdd() {
        Money m12CHF= new Money(12, "CHF");  // (1)
        Money m14CHF= new Money(14, "CHF");
        Money expected= new Money(26, "CHF");
        Money result= m12CHF.add(m14CHF);    // (2)
        Assert.assertTrue(expected.equals(result));     // (3)
    }
}
```

But wait, how do we check if two Money objects are equal?

```java
public class MoneyTest extends TestCase {
    public void testSimpleAdd() {
        ...        Assert.assertTrue(expected.equals(result));     // (3)
    }
}
```

Now we have the test, let’s implement the code

```java
public class Money {
    // ... all the previous code
    public boolean equals(Object anObject) {
        if (anObject instanceof Money) {
            Money aMoney= (Money)anObject;
            return aMoney.currency().equals(currency())
                && aMoney.amount() == amount();
        }
        return false;
    }
}
```

Maybe add another case?

```java
public class MoneyTest extends TestCase {
    public void testSimpleAdd() {
        Money m12CHF= new Money(12, "CHF");
        Money m14CHF= new Money(14, "CHF");
        ...
        Assert.assertTrue(!m12CHF.equals(o)); // new test case here
    }
}
```

Note special `assertEquals` method. If not equal, tester will print `toString` of each expression.

As an aside, always define a good `toString` method for every class.

Other assertXXX variants, check out `http://junit.sourceforge.net/javadoc/junit/framework/Assert.html`
Avoiding code duplication between different tests

- Note that there is some code duplication in creating the fixtures in the two methods testSimpleAdd and testEquals.
- Can put common code into methods setUp() and tearDown.

```java
public class MoneyTest extends TestCase {
    private Money f12CHF;
    private Money f14CHF;

    protected void setUp() {
        f12CHF = new Money(12, "CHF");
        f14CHF = new Money(14, "CHF");
    }

    public void testEquals() {
        Assert.assertTrue(!f12CHF.equals(null));
        Assert.assertEquals(f12CHF, f12CHF);
        Assert.assertEquals(f12CHF, new Money(12, "CHF");
        Assert.assertTrue(!f12CHF.equals(f14CHF));
    }

    public void testSimpleAdd() {
        Money expected = new Money(26, "CHF");
        Money result = f12CHF.add(f14CHF);
        Assert.assertTrue(expected.equals(result));
    }
}
```

Putting test cases into a suite of tests

Define a static method called suite() as follows:

```java
public static Test suite() {
    TestSuite suite = new TestSuite();
    suite.addTest(new MoneyTest("testEquals");
    suite.addTest(new MoneyTest("testSimpleAdd");
    return suite;
}
```

- If you want to explicitly list the tests to include the following in MoneyTest:

```java
public static Test suite() {
    return new TestSuite(MoneyTest.class);
}
```

Ok, now let’s run the test suite.

- Make sure junit.jar is on your CLASSPATH, or explicitly give the classpath on your call to java.
- use the command-line version:
  ```java
  java junit.textui.TestRunner MoneyTest
  ```
- or use the Swing version:
  ```java
  java junit.swingui.TestRunner MoneyTest
  ```

Some general testing practices

- Martin Fowler says "Whenever you are tempted to type something into a print statement or a debugger expression, write it as a test case instead."
- At first you will have to create a lot of fixtures, but then you will find you have created all the infrastructure and new tests become easier to add.
- Try to write tests that you imagine to be useful. Look for the boundary cases.
- When to add tests:
  - During development: while you are designing your class (but before implementing).
  - During debugging: when someone discovers a defect, first write a test that should succeed if your program is working, then debug until it succeeds.
- When to run the tests:
  - All the time.
  - If you find newly introduced errors right away, then you have a good idea where the error might be.
  - Fix errors right away, keep your test suite running.