References

Object-Oriented Design Heuristics, Riel, 1996
JUnit Cookbook http://junit.sourceforge.net/doc/cookbook/cookbook.htm


JUnit Javadoc: http://www.junit.org/junit/javadoc/3.8/index.htm

Brian Marick’s Testing Web Site: http://www.testing.com/

Testing for Programmers, Brian Marick, Available at: http://www.testing.com/writings.html
Terms

Abstraction

Encapsulation

Information Hiding

Polymorphism
Heuristics

All data should be hidden within its class

public class A {
    public int x;
    public int y;
    public int z;
}

public class B {
    private int x;
    private int y;
    private int z;

    public int getX() { return x;}
    public int getY() { return y;}
    public int getZ() { return z;}
    public void setX(int value) { x = value; }
    public void setY(int value) { y = value; }
    public void setZ(int value) { z = value; }
}
Heuristics

Keep related data and behavior in one place

Beware of classes that have many accessor methods defined in their public interface

Do not create god classes/objects in your system

A class should capture one and only one key abstraction

Beware of classes that have too much noncommunicating behavior
Testing

Johnson's Law

If it is not tested it does not work

The more time between coding and testing

More effort is needed to write tests
More effort is needed to find bugs
Fewer bugs are found
Time is wasted working with buggy code
Development time increases
Quality decreases
Unit Testing

Tests individual code segments

Automated tests
What wrong with:

Using print statements

Writing driver program in main

Writing small sample programs to run code

Running program and testing it be using it
We have a QA Team, so why should I write tests?
When to Write Tests

First write the tests

Then write the code to be tested

Writing tests first saves time

   Makes you clear of the interface & functionality of the code

   Removes temptation to skip tests
What to Test

Everything that could possibly break

Test values
   Inside valid range
   Outside valid range
   On the boundary between valid/invalid

GUIs are very hard to test
   Keep GUI layer very thin
   Unit test program behind the GUI, not the GUI
Common Things Programs Handle Incorrectly

Adapted with permission from “A Short Catalog of Test Ideas” by Brian Marick,
http://www.testing.com/writings.html

Strings
Empty String

Collections
Empty Collection
Collection with one element
Collection with duplicate elements
Collections with maximum possible size

Numbers
Zero
The smallest number
Just below the smallest number
The largest number
Just above the largest number
XUnit

Free frameworks for Unit testing

SUnit originally written by Kent Beck 1994

JUnit written by Kent Beck & Erich Gamma

Available at: http://www.junit.org/

Ports to many languages at:
  http://www.xprogramming.com/software.htm
JUnit Example

Goal: Implement a Stack containing integers.

Tests:

Subclass junit.framework.TestCase

Methods starting with "test" are run by TestRunner

First tests for the constructors:

```java
import junit.framework.*;
public class TestStack extends TestCase {

    public void testDefaultConstructor() {
        Stack test = new Stack();
        assertTrue("Default constructor", test.isEmpty() );
    }

    public void testSizeConstructor() {
        Stack test = new Stack(5);
        assertTrue( test.isEmpty() );
    }

}
```
public class Stack {
    int[] elements;
    int topElement = -1;

    public Stack() {
        this(10);
    }

    public Stack(int size) {
        elements = new int[size];
    }

    public boolean isEmpty() {
        return topElement == -1;
    }
}
Running JUnit Using Eclipse

After creating your Stack Class

Select JUnit Test Case in Create Icons Menu
Running JUnit Using Eclipse

Fill in dialog window & create the test cases

Select Junit test case from the "Run as..." menu
Assert Methods

assertTrue()
assertFalse()
assertEquals()
assertNotEquals()
assertSame()
assertNotSame()
assertNull()
assertNotNull()
fail()

For a complete list see

Testing the Tests

If can be useful to modify the code to break the tests

package example;

public class Stack {
    int[] elements;
    int topElement = -1;

    etc.

    public boolean isEmpty() {
        return topElement == 1;
    }
}

Test Fixtures

Before each test setUp() is run

After each test tearDown() is run

package example;
import junit.framework.TestCase;

public class StackTest extends TestCase {
    Stack test;

    public void setUp() {
        test = new Stack(5);
        for (int k = 1; k <= 5; k++)
            test.push(k);
    }

    public void testPushPop() {
        for (int k = 5; k >= 1; k--)
            assertEquals("Pop fail on element " + k, test.pop(), k);
    }
}


Testing Exceptions

```java
public void testIndexOutOfBoundsException() {
    ArrayList list = new ArrayList(10);
    try {
        Object o = list.get(11);
        fail("Should raise an IndexOutOfBoundsException");
    } catch (IndexOutOfBoundsException success) {} 
}

Example is from the JUnit FAQ
```