TRAMP

Testing Tutorial: JUnit & CppUnit

Alexandra Remptke & Patrice Kwemo

December 10th 2001

List of contents

- Introduction
- JUnit
  - Installation
  - TestCase
  - TestResult
  - Class Assert
  - Fixture
  - TestSuite
- CppUnit

Introduction

Unit testing

testing of individual components to find and fix errors fast and cheap at the system level

XUnit testing framework

- Java - Code: JUnit 3.7
- C++ - Code: CppUnit

JUnit 3.7

a freely available set of Java packages that allow one to readily create Java test cases for Java classes and to then run these unit tests interactively or in batch mode

JUnit 3.7 - Installation

- download JUnit 3.7 from the JUnit website http://junit.org
- set the classpath:
  - `setenv CLASSPATH .:/private/Network/Servers/macbruegge11/Volumes/raid/Users/yourname/junit/junit3.7/junit.jar`
- test the installation
  - `javac junit/samples/money/*.java`
  - `java junit.samples.money.MoneyTest`
  - `OK (21 tests)`

JUnit 3.7 - TestCase

To use a test case following objects/methods have to be implemented:

- code which creates the objects you will interact with during the test (test‘s fixture*)*
- code which exercises the objects in the fixture
- code which verifies the result

Writing a test case follows the three steps:

- create an instance of TestCase
- override the method `run()`
- if you want to check a value, call `assert()` and pass a boolean that is true if the test succeeds

JUnit 3.7 - TestCase

Class Money

```
public class Money {
  private double amount;

  public Money(double amount) {
    this.amount = amount;}

  public double getAmount() {
    return this.amount; }
}
```

Class MoneyTest

```
import junit.framework.*;

public class MoneyTest extends TestCase {
  public MoneyTest(String name) {
    super(name);}

  public void testAmount() {
    Money money = new Money(2.00);
    assertTrue(2.00 == money.getAmount());}

  public static void main(String[] args) {
    junit.swingui.TestRunner.run(MoneyTest.class);}
}
```

JUnit 3.7 - TestResult

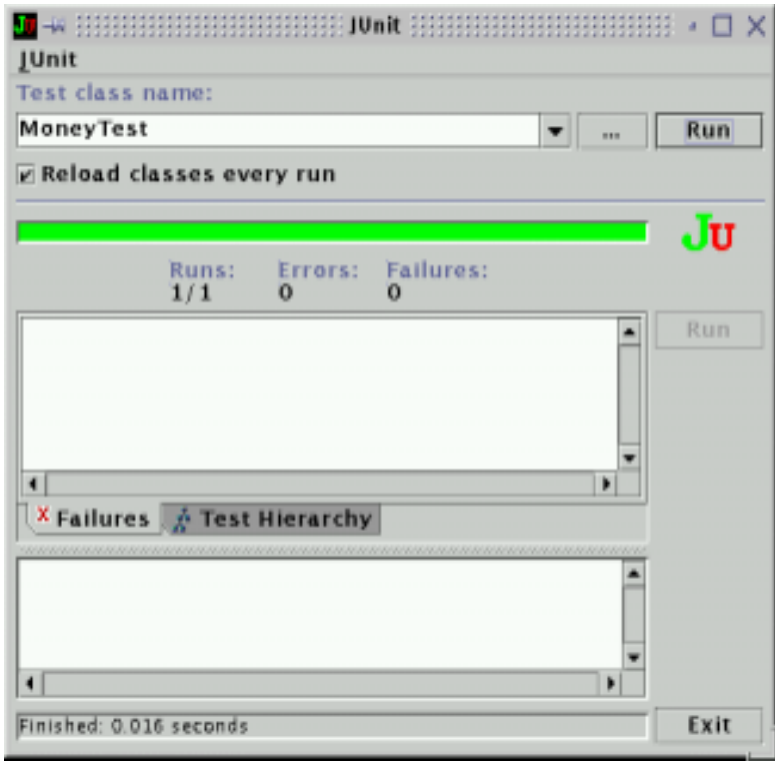
Start TestRunner

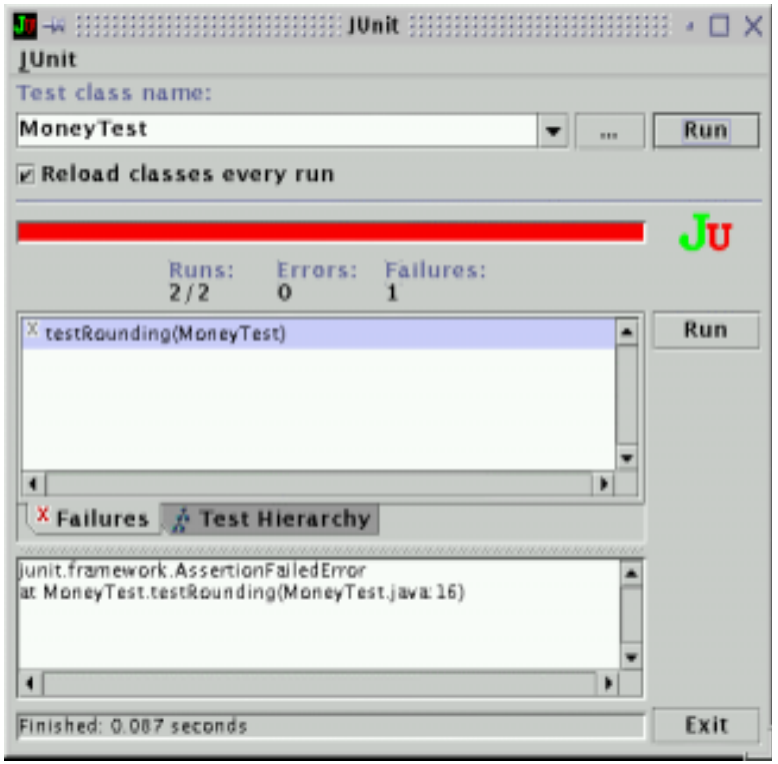
- `java MoneyTest`
- `java junit.awtui. TestRunner`
- `junit.swingui.TestRunner`

The graphical user interface presents a window with:

- a field to type in the name of a class with a suite method
- a Run button to start the test
- a progress indicator that turns from red to green in the case of a failed test
- a list of failed tests

JUnit 3.7 - TestResult





JUnit 3.7 - Class Assert

- `assertTrue(boolean condition)`
  asserts that a condition is true
- `assertEquals(Object expected, Object actual)`
  asserts that two objects are equal
- `assertEquals(int expected, int actual)`
  asserts that two int are equal - the comparison is done by the == operator
- `assertEquals(double expected, double actual, double delta)`
  asserts that two doubles are equal - the last argument gives the tolerance
- `assertNull(Object object)`
  asserts that an object is null
- `assertNotNull(Object object asserts)`
  that an object is not null
- `assertSame(Object expected, Object actual)`
  asserts that two o the same object

The `assertEquals`-method can also be used with float, long, boolean, byte, char and short.

JUnit 3.7 - Fixture

Tests need to run against the background of a known set of objects - the test fixture. Often the same fixture will be used for several different tests.

Tests set up a test fixture, run some code against the fixture, check some results, and then clean up the fixture.

What to do with a common fixture:

- create a subclass of TestCase
- add an instance variable for each part of the fixture
- override `setUp()` to initialize the variables
- override `tearDown()` to release any permanent resources you allocated in `setUp()`

JUnit 3.7 - Fixture

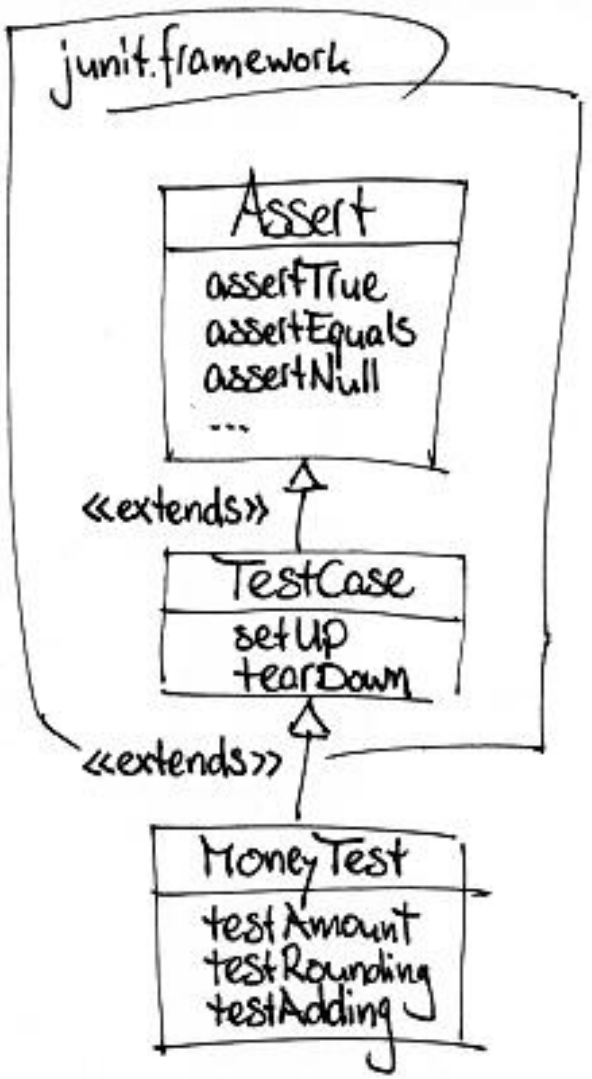
Class Money

```
public Money add(Money other) {
   return new Money(this.cents + other.cents);}
```

Class MoneyTest

```
public class MoneyTest extends TestCase {

  private Money money;

  public MoneyTest(String name) {
    super(name);}

  protected void setUp() {money = new Money(2.00);}

  protected void tearDown() {}

  public void testAmount(){
    assertTrue(2.00== money.getAmount());}

  public void testAdding() {
    Money sum = new Money(1.00).add(money);
    assertEquals("sum", 3.00, sum.getAmount(), 0.001);}

  public static void main(String[] args) {
    junit.swingui.TestRunner.run(MoneyTest.class);}
  }
```

JUnit 3.7 - UML - model



Assert

test values and conditions

TestCase

isolate a number of test cases for a common fixture from test objects

MoneyTest

test the behaviour of the devoloped class Money

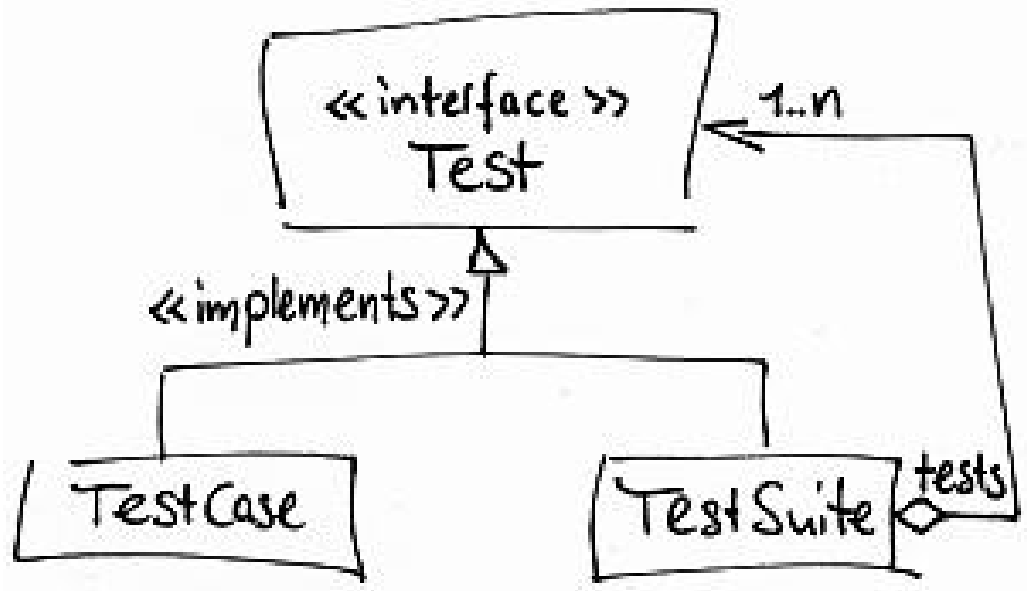
JUnit 3.7 - TestSuite

A TestSuite runs a collection of tests by definition of a static method called suite. Inside suite you add the tests to be run to a TestSuite object and return it.

```
import junit.framework.*;

  public class AllTests {
    public static Test suite() {
      TestSuite suite = new TestSuite();
      suite.addTestSuite(CustomerTest.class);
      suite.addTestSuite(MoneyTest.class);
      suite.addTestSuite(MovieTest.class);
      return suite;
    }
  }
```

JUnit 3.7 - TestSuite

TestSuite and TestCase both implement an interface called Test which defines the methods to run a test. This enables the creation of test suites by composing arbitrary TestCases and TestSuites.



Test

abstract from test cases and test suites

TestSuite

run a number of tests together

CppUnit

a C++ port of the JUnit testing framework developed by Erich Gamma and Kent Beck. Michael Feathers did the first port of JUnit to C++.

CppUnit 1.6.2 - Installation

- download CppUnit from http://sourceforge.net/projects/cppunit.
- **Installation for MacOS:**
  - Unpack the cppunit sources with
  - **gunzip cppunit-1.6.2.tar.gz**
  - Type **./configure** from the directory containing the package's source

CppUnit 1.6.2 – Installation(2)

- patch the **libtool** script generated, replacing **"-shared"** by **"-dynamiclib"** in the line starting with **"archive_cmds=".**

- Type **"make "** to compile the package, Optionally type **" make check "** to run any self-tests that come with the package.

- type **" `make install "** to install the programs and any data files and documentation.

CppUnit 1.6.2 - Testing

How to test your C++ programm?

1. When you want to test class named Student write a class (let's call it TestStudent). This class must inherit the class TestCase that is defined by the CppUnit framework.
2. Create a constructor for this class; passing a name that is representative of the set of tests for this class as the parameter.
3. Create a fixture.
4. Each 'test' you perform is represented by the implementation of a method in the test class

CppUnit 1.6.2 - Testing

5. In each test method you create, use the assertion mechanism provided by CppUnit to compare the results of running the test and the results you expected.

6. You can call each testing method in the main method of the MyMainTest.cpp by adding following lines into the main method: TestCaller<TestStudent>test("testMy ",&TestStudent::testMy);
test.run ();

7. Organize them into a suite by adding a suite() Method Including

```
TextTestRunner runner;
Runner.addTest( StudentTestCase::suite() );
  //in case you'd have another test class e.g. CourseTest class and suite() method there add
 Runner.addTest( CourseTestCase::suite());
 runner.run( "", true );    // Run all tests and wait
```

Test runner messages

The TextTestRunner will run the tests. If all the tests pass, you'll get an informative message. If any fail, you'll get the following information:

1. The name of the test case that failed
2. The name of the source file that contains the test
3. The line number where the failure occurred
4. All of the text inside the call to CPPUNIT_ASSERT which detected the failure

Example

CppUnit 1.6.2 – Class course.h

```
//************************Course.h************************
#ifndef Course_h
#define Course_h

#include <string>

class Course
{
    public:
    // Default constructor
    Course();
     // Constructor
   Course(std::string nm, int gr);
    // Method to get the name of the course
   std::string getCourseName();
     // method to get the grade of the course
    int getCourseGrade();
     private:
    std::string course_name;      // name of this course
    int grade;                    // grade of this course
  };
#endif
```

CppUnit 1.6.2 – Class course.cpp

```
//**********************Course.cpp**********************
#include "Course.h"

// Default constructor
Course::Course()
{
      course_name = "";
      grade = -1;
}
 // Constructor
Course::Course(std::string nm, int gr):course_name(nm)
{
      grade = gr;
}
 // Method to get the name of the course
std::string Course::getCourseName()
{
   return course_name;
}
 // Method to get the grade of the course
int Course::getCourseGrade()
{
   return grade;
}
```

CppUnit 1.6.2 – Class Student.h

```
//************************Student.h************************

#ifndef Student_h
#define Student_h

#include <iostream>
#include <string>
#include "Course.h"

const int MAXNUM = 20;     // Maximum number of courses allowed per
student

class Student
{
    public :
      // Constructor
      Student(std::string nm, std::string no);

      // Method to return student's name
      std::string getStuName();
```

CppUnit 1.6.2 – Class Student.h

```
// Method to return student's number
      std::string getStuNumber();

      // Method to assign a grade to a course
      void assignGrade(std::string co, int gr);

      // Method to return the grade of a course
      int getGrade(std::string co);
    private:
      std::string name;                  // name of the student
      std::string number;                // the student's number
      Course course_grades[MAXNUM];      // courses taken by student
      int no_of_courses;                 // the current number of courses taken
};
#endif
```

CppUnit 1.6.2 – Class Student.cpp

```
//************************Student.cpp************************
 #include "Student.h"
  // Constructor
Student::Student(std::string nm, std::string no):name(nm), number(no)
{
      no_of_courses = 0;
 }
  // Method to return student's name
std::string Student::getStuName()
{
    return name;
}
 // Method to return student's number
 std::string Student::getStuNumber()
{
   return number;
}
  // Method to assign a grade to course
void Student::assignGrade(std::string co, int gr)
```

CppUnit 1.6.2 - Class Student.cpp

```
{     // Check whether the maximum number of courses have been taken
     if (no_of_courses == MAXNUM)
    {
       std::cout << "You have exceeded the maximum number of courses !\n";
       return;
     }
     // Create a new course
     Course c(co, gr);
     course_grades[no_of_courses++] = c;
}
// Method to return the grade of a course
int Student::getGrade(std::string co)
{
      int i = 0;
            while (i < no_of_courses)
     {
       //check if course name the same as co
       if (course_grades[i].getCourseName() == co)
         return (course_grades[i].getCourseGrade());
       i++;
     }
return(-1);
}
```

CppUnit 1.6.2 – Class TestStudent.h

```
//**********************TestStudent.h************************

#ifndef TestStudent_h
#define TestStudent_h

#include <iostream>
#include <string>

 // Note 1
#include "TestCase.h"
#include "TestSuite.h"
#include "TestCaller.h"
#include "Student.h"
```

CppUnit 1.6.2 - Class TestStudent.h

```
class StudentTestCase : public TestCase
{ // Note 2
    public:
      // Constructor - Note 3
      StudentTestCase(std::string name) : TestCase(name) {}

      // Method to test the constructor
      void testConstructor();

      // method to test the assigning and retrieval of grades
      void testAssignAndRetrieveGrades();

      // Method to create a suite of tests
      static Test *suite ();
};
#endif
```

CppUnit 1.6.2 – Class TestStudent.cpp

```
//**********************TestStudent.cpp**********************

#include "TestStudent.h"

// Method to test the constructor
void StudentTestCase::testConstructor()
{  // Note 4
      // Create a student object
      Student stu("Tan Meng Chee", "94-1111B-13");

      // check that the object is constructed correctly - Note 5
      std::string student_name =stu.getStuName();
      assert(student_name == "Tan Meng Chee");
      std::string student_number = stu.getStuNumber();
      assert(student_number == "94-1111B-13");
}

// Method to test the assigning and retrieval of grades
void StudentTestCase::testAssignAndRetrieveGrades
```

CppUnit 1.6.2 – Class TestStudent.cpp

```
{
    // Create a student
    Student stu("Jimmy", "946302B");

    // Assign a few grades to this student
    stu.assignGrade("cs2102", 60);
    stu.assignGrade("cs2103", 70);
    stu.assignGrade("cs3214s", 80);

    // Verify that the assignment is correct - Note 6
    assertEquals(60, stu.getGrade("cs2102"));
    assertEquals(70, stu.getGrade("cs2103"));

    // Attempt to retrieve a course that does not exist
    assertEquals(-1, stu.getGrade("cs21002"));
}
```

CppUnit 1.6.2 – Class TestStudent.cpp

```
 // Method to create a suite of tests - Note 7
Test *StudentTestCase::suite ()
{
      TestSuite *testSuite = new TestSuite ("StudentTestCase");

      // Add the tests
      testSuite->addTest (new TestCaller <StudentTestCase>
         ("testConstructor", &StudentTestCase::testConstructor));
      testSuite->addTest (new TestCaller <StudentTestCase>
         ("testAssignAndRetrieveGrades",
          &StudentTestCase::testAssignAndRetrieveGrades));
      return testSuite;
}
```

QUESTION?