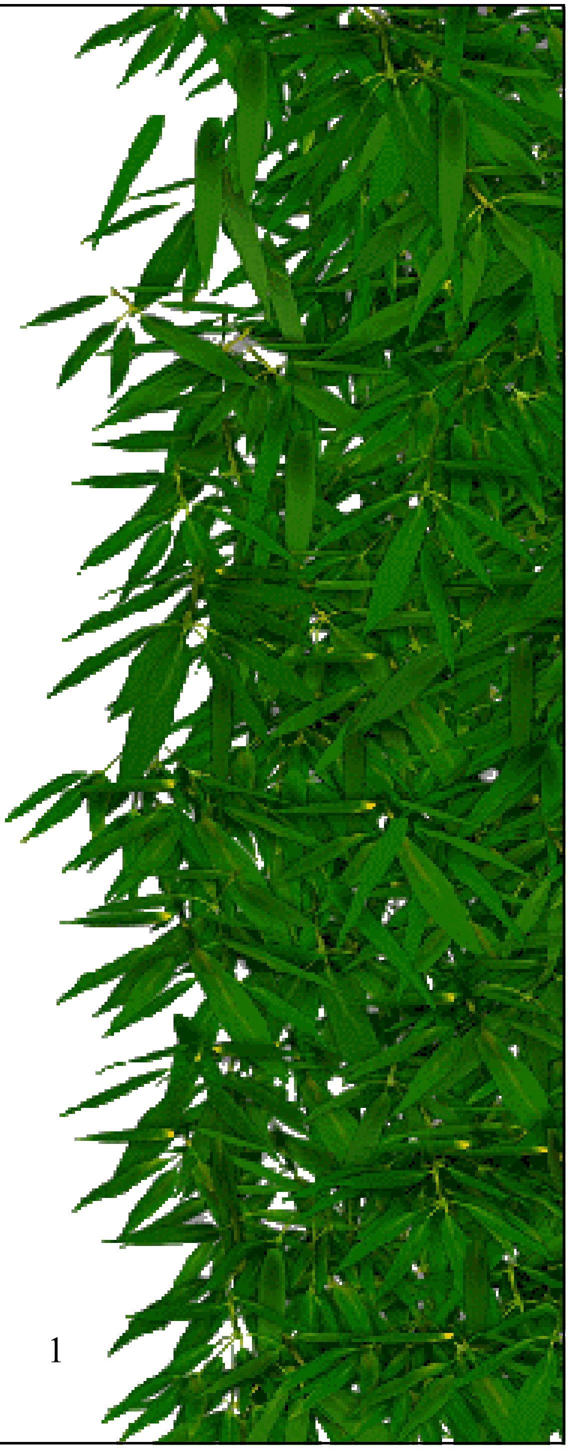
Unit Testing and SQL

Shane Cantrell
Zach Crisman



Resources

- JUnit
  - http://www.junit.org/index.htm
  - http://sourceforge.net/projects/junit
  - http://junit.sourceforge.net/doc/testinfected/testing.htm
- SQL
  - http://www.w3schools.com/sql/default.asp
  - http://shane.hydrus.net/cse403/MyDatabase.zip

Why Test?

- Gets you to think about possible modes of failure
- Allows you to easily verify that nothing has been inadvertently broken
- If something breaks, then you know right away (assuming it was covered in a test)
- Allows test code to be conveniently packaged for continued use

JUnit: Planning

- Initialize test variables
- Run the test
- Create the solution using a direct method
- Compare the results

- Classes should be designed with unit testing in mind!

JUnit: Basic Steps

- Extend class TestCase
  - Keep it in the same package as the classes to be tested, so that it can access package private methods
- Create public functions to test each case
- Create the “public static Test suite()” function, which returns a suite containing your test functions

JUnit: Example Skeleton

```
public class MyTest extends TestCase {

  protected void setUp() { ... }

  protected void tearDown() { ... }

  public static Test suite() { ... }

}
```

JUnit: Example suite()

```
public class MoneyTest extends TestCase {

  ...

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

JUnit: Class Assert

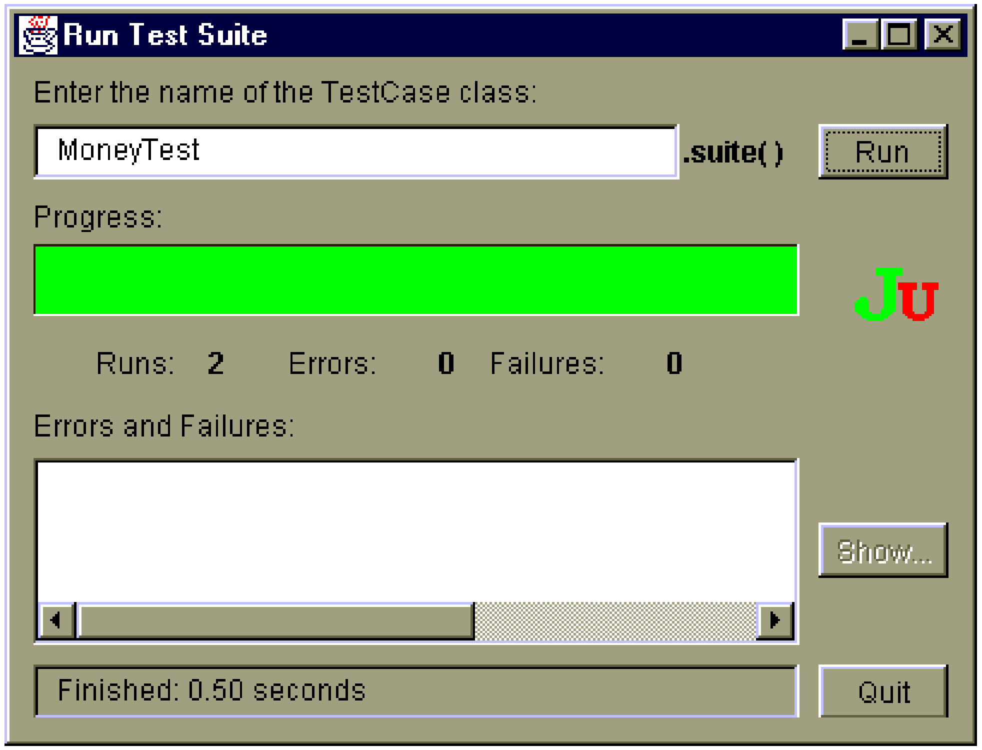
- assertEqual(*expected*, *actual*)
- assertTrue(*boolean*)
- assertFalse(*boolean*)
- assertNull(*object*)
- assertNotNull(*object*)
- assertSame(*expected*, *actual*)
- assertNotSame(*expected*, *actual*)

JUnit: Example Class

```
public class MathTest extends TestCase {
  protected double fValue1;
  protected double fValue2;

  protected void setUp() {
    fValue1 = 2.0;
    fValue2 = 3.0;
  }

  public void testAdd() {
    double result = fValue1 + fValue2;
    assertTrue(result == 5.0);
  }

  ...
}
```

JUnit: Running Tests

- java junit.textui.TestRunner junit.samples.AllTests
- java junit.swingui.TestRunner junit.samples.AllTests



Runs: 2 Errors: 0 Failures: 0

Finished: 0.50 seconds

SQL: What is it?

- SQL (Structured Query Language)
  - ANSI language for interfacing databases
  - Uses very simple text commands
- SQL Databases
  - Organization is similar to a bunch of linked spreadsheets called “tables”
  - Store text, numbers, etc.
  - Most have their own proprietary extensions

SQL: Tables

- Creating
  - CREATE TABLE *tableName* (*columnName1 type, columnName2 type, ...*)
  - Ex: CREATE TABLE objects (objID int, name varchar(80))
- Deleting
  - DROP TABLE *tableName*
  - Ex: DROP TABLE objects

SQL: Inserting Data

- Insert a Row
  - INSERT INTO *tableName* (*columnName1*, ..., *columnNameN*) VALUES (*value1*, ..., *valueN*)
  - Ex: INSERT INTO objects (objID, name) VALUES (0, ‘void’)
- Update a Cell
  - UPDATE *tableName* SET *columnName1* = *value1*, ..., *columnNameN* = *valueN* WHERE *criteria*
  - Ex: UPDATE objects SET name = ‘apple’ WHERE objID = 0
- Delete a Row
  - DELETE FROM *tableName* WHERE *criteria*
  - Ex: DELETE FROM objects WHERE objID = 0

SQL: Finding Data

- Basic Usage
  - SELECT *columnName1*, ..., *columnNameN* FROM *tableName*
  - Ex: SELECT objID, name FROM objects

- Specifiers
  - WHERE (criteria for selecting rows)
    - **WHERE *criteria***
    - **WHERE *columnName* = *value***
    - **WHERE *columnName* > *value* AND *criteria***
    - **WHERE *columnName* LIKE *value***
  - GROUP BY (criteria for grouping rows)
  - ORDER BY (criteria for ordering rows)
    - **ORDER BY *columnName1, ..., columnNameN***
    - **ORDER BY *columnName1* ASC, ..., *columnNameN* DESC**
  - INNER JOIN (merge rows from multiple tables)

SQL: What You Need

- SQL Driver
  - org.postgresql.Driver
- URL
  - jdbc:postgresql://cubist.cs.washington.edu/shanec
- Username
- Password
- Restart Tomcat to get an updated Java classpath

JDBC and SQL

- package java.sql.*
  - DriverManager
  - Connection
  - Statement
  - ResultSet
  - ResultSetMetaData
  - SQLException
- java.lang.Class

Loading the Driver

- Class.forName(JDBC_DRIVER);
- Connection connection = DriverManager.getConnection(DATABASE_URL, DATABASE_USERNAME, DATABASE_PASSWORD);

- JDBC_DRIVER = “org.postgresql.Driver”
- DATABASE_URL = “jdbc:postgresql://cubist.cs.washington.edu/shanec”
- DATABASE_USERNAME = “shanec”
- DATABASE_PASSWORD = “pwd”

Sending Commands

- Statement statement;
- statement = connection.createStatement();
- statement.execute(“ ... ”);
- ResultSet resultSet = statement.executeQuery(“ ... ”);

Important Points

- ResultSet
  - Only one per statement object
  - Close automatically with closure of statement or new statement method call
  - Must advance to the first row before accessing data
  - Column indices start with one

Other: POSTing Files

http://snowwhite.it.brighton.ac.uk/~mas/mas/courses/html/html.html

```
<FORM ENCTYPE="multipart/form-data"
      ACTION="URL"
      METHOD=POST>
Send file name:<BR>
  <INPUT NAME="message"
         TYPE="file"> <BR> <BR>
  <INPUT TYPE="submit"
         VALUE="Send file to server">
</FORM>
```

Reminder

- Unit Testing
- Logging (java.util.logging)

- Jakarta Libraries

- CVS
- E-Mail Lists
- Bug Tracking

Next Week

- Discussion on testing and debugging!
  - Think about what Ian King has to say.
  - Do you agree or disagree?
  - How does your testing compare?
  - Do you have testing stories from your past?