Unit Testing

CS 240 – Advanced Programming Concepts
F-22 Raptor Fighter
F-22 Raptor Fighter
• Manufactured by Lockheed Martin & Boeing
• How many parts does the F-22 have?
F-22 Raptor Fighter

• What would happen if Lockheed assembled an F-22 with "untested" parts (i.e., parts that were built but never verified)?

• It wouldn't work, and you probably would never be able to make it work
  – Cheaper and easier to just start over
Managing Implementation Complexity

• Individual parts should be verified before being integrated with other parts

• Integrated subsystems should also be verified

• If adding a new part breaks the system, the problem must be related to the recently added part

• Track down the problem and fix it

• This ultimately leads to a complete system that works
2 Approaches to Programming

• Approach #1
  – "I wrote ALL of the code, but when I tried to compile and run it, nothing seemed to work!“

• Approach #2
  – Write a little code (e.g., a method or small class)
  – Test it
  – Write a little more code
  – Test it
  – Integrate the two verified pieces of code
  – Test it
  – ...
Unit Testing

• Large programs consist of many smaller pieces
  – Classes, methods, packages, etc.

• "Unit" is a generic term for these smaller pieces

• Three important types of software testing are:
  – Unit Testing (test units in isolation)
  – Integration Testing (test integrated units)
  – System Testing (test entire system that is fully integrated)

• Unit Testing is done to test the smaller pieces in isolation before they are combined with other pieces
  – Usually done by the developers who write the code
What Unit Tests Do

- Unit tests create objects, call methods, and verify that the returned results are correct

- Actual results vs. Expected results

- Unit tests should be automated so they can be run frequently (many times a day) to ensure that changes, additions, bug fixes, etc. have not broken the code
  - Regression testing

- Notifies you when changes have introduced bugs, and helps to avoid destabilizing the system
Test Driver Program

• The tests are run by a "test driver", which is a program that just runs all of the unit test cases

• It must be easy to add new tests to the test driver

• After running the test cases, the test driver either tells you that everything worked, or gives you a list of tests that failed

• Little or no manual labor required to run tests and check the results
JUnit Testing Design

- Write a separate test method for each test
  - Marked with @Test annotation
- Set up method(s) may be executed before each test method
  - Marked with @Before
- Tear down method(s) may be executed after each test
  - Marked with @After
- Use JUnit `Assert.*()` methods to implement test cases
- Failures reported in various ways, depending on language and tool (command-line, GUI, IDE integrated)
- Example:
  - WordExtractor.java
  - WordExtractorTest.java
Running Junit Tests from IntelliJ and Android Studio

• To run a single test class, in the “Project” tool window right-click on a test class name, and select “Run Tests” or “Debug Tests”

• To run all of your unit tests, right-click on the “test/java” folder, and select “Run All Tests” or “Debug All Tests”
Running Unit Tests from The Command-Line

• Write a test driver class whose “main” method invokes the org.junit.runner.JUnitCore class to run your unit tests

• Run your test driver program from the command-line:
  
  java –cp build\classes\main;build\classes\test;libs\junit-4.12.jar;libs\sqlite-jdbc-3.16.1.jar TestDriver
JUnit 4 Unit Testing Framework

- **JUnit 4 Documentation**
- Use JUnit 4 annotations to mark test methods

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Test public void method()</td>
<td>The annotation @Test identifies that a method is a test method.</td>
</tr>
<tr>
<td>@Before public void method()</td>
<td>Will execute the method before each test. This method can prepare the test environment (e.g. read input data, initialize the class).</td>
</tr>
<tr>
<td>@After public void method()</td>
<td>Will execute the method after each test. This method can cleanup the test environment (e.g. delete temporary data, restore defaults).</td>
</tr>
</tbody>
</table>
### JUnit 4 Unit Testing Framework

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@BeforeClass</code> public void method()</td>
<td>Will execute the method once, before the start of all tests. This can be used to perform time intensive activities, for example to connect to a database.</td>
</tr>
<tr>
<td><code>@AfterClass</code> public void method()</td>
<td>Will execute the method once, after all tests have finished. This can be used to perform clean-up activities, for example to disconnect from a database.</td>
</tr>
<tr>
<td><code>@Test (expected = Exception.class)</code></td>
<td>Fails, if the method does not throw the named exception.</td>
</tr>
<tr>
<td><code>@Test(timeout=100)</code></td>
<td>Fails, if the method takes longer than 100 milliseconds.</td>
</tr>
</tbody>
</table>
Adding the JUnit Library to Your Project

• Maven

    <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
    </dependency>

• Gradle (build.gradle file)

testCompile group: 'junit', name: 'junit', version: '4.12'
A More Detailed Example

- code-example on website in the unit testing lecture notes
- Contains code for web-based spelling checker
- “Real” classes are in:
  - src/main/java/spellcheck/*.java
  - src/main/java/dataaccess/*.java
- “Test” classes are in:
  - src/test/java/spellcheck/*.java
  - src/test/java/dataaccess/*.java
Android Testing Framework

• Android provides a framework for writing automated unit tests based on Junit

• There are two types of Android unit tests
  – Local Unit Tests
    • These tests depend only on standard Java classes and can be ran on the development computer instead of on an Android device
    • You will create local unit tests for the Family Map Server project
  – Instrumented Unit Tests
    • These tests depend on Android-specific classes and must be run on an Android device
    • You will create instrumented unit tests for the Family Map Client project
Android Local Unit Tests

- **Official Documentation**
- Can run on the development computer without a device or emulator
- Module’s primary source code is located in the folder
  - `<module>/src/main/java/<package>`
- Local unit test code is located in the folder
  - `<module>/src/test/java/<package>`
Database Unit Tests

• When writing unit tests for your database code, there are additional things to think about

• Put database driver JAR file on the class path

• Each unit test should start with a pristine database so prior tests have no effect
  – Can re-create tables before each test
  – Or, you can “rollback” the effects of each test so they are undone and don’t affect later tests