Today
- Testing

Homework
- HW 5 (out)

Reading
- Ch 7: 235-251

Reminder
- Exam 2 next Thursday
- Project updates following Tuesday
Basic Testing Concepts

“Verification”
- does the software satisfy the specification?
- “is it built correctly?”

“Validation”
- does the software fulfill its intended purpose?
- “was it the right thing to build?”

Failure
- behavior deviates from what is expected

Defect (aka bug)
- the cause of a failure
- some part of the code that is incorrectly implemented

The two parts of “Testing”
- finding bugs (defects)
- fixing bugs
As developers and testers

- want to find defects in the system
- so we can fix them
- and verify our system works correctly

Q: What % of software has bugs?
- 100% !!!

Q: Why care about testing?

- Software Reliability
  - probability of failure-free operation in a specified environment for a specified time
  - tied to some level of failure-freeness (some failures may be fine)
- Reliability is an important aspect of quality
  - others: usability, efficiency, maintainability, complete, documented

Help ensure reliability through testing
Different aspects of a system are tested ...

Lots and lots of tests!  
... Need to Keep track of all this data

A slightly different view

- Look inside the system
- Code, DBs, algorithms, etc.
- E.g., test all paths through code (not practical!)
- Tests based on how system designed and coded  
  (mainly unit tests)

- Do not look inside system
- Test inputs and outputs
- What the user “sees”  
  (functional & acceptance tests)
Regression Testing

Good (common) programming practice ...

- write some (often small) portion of the application
- define some tests and get them to pass (succeed)
- modify the code (i.e., refactor and/or extend)

How do we ensure the changes didn’t mess up previous code?

- By performing regression testing
  - rerun our (previous) tests to ensure they still pass
  - helps ensure changes didn’t introduce bugs in previous work

Unit testing tools are extremely valuable for regression tests!
Unit Test (white box)

Test individual “components” (or “units”) of the code

- notion of a unit somewhat open-ended
- typically a method
- could be an entire class

Test each unit in “isolation”

- e.g., each method of a class is tested individually

Historically starts by analyzing code

- then determine the appropriate “test cases”
- we’ll see a different approach later (TDD)

The nature of test cases can vary

- often just giving inputs to check outputs
- but depends on the unit “under test”
- for example, for a method could be
  - whether it succeeds for normal use
  - whether it handles abnormal use (like throws Exceptions)
  - whether it calls other methods properly
Automated Testing

In automated testing ...

- sets of test cases ("test suites") executed and verified by a program
- contrast to “manual” testing ... where you have to execute and verify test cases by hand

Automated tests are written as code

- requires writing “test code” to verify program behavior
- can lead to lost of additional code (for testing)
- and this code is not part of the production system

Q: What are some possible disadvantages over manual tests?

- slows development speed because involves writing (more) code
- can be tricky to write for some cases (e.g., GUIs)

Q: What are some possible advantages over manual tests?

- much easier to do regression testing!
- tests can be run quickly
- computer doesn’t get bored
- tests are run exactly the same way each time
- code serves to document the test (and possibly API)

In practice, code without automated tests is viewed as highly suspect!
Automated Testing Frameworks

Lots of tools exist for automated testing …

- We’re going to look at JUnit (v4)
- a widely used unit testing framework
- download at: www.junit.org (under “Releases”)

Tools like JUnit try to make unit tests as painless as possible

- to write and run tests
- most of these tools are very similar
- e.g., NUnit (.Net), CppUnit (C++), PyUnit (Python), ...
JUnit Example

Assume we have this (really) simple Java class:

```java
public class Stock {
    private int shares;
    private String symbol;
    public Stock(int theShares, String theSymbol) {
        shares = theShares;
        symbol = theSymbol;
    }
    public int calculateValue(int sharePrice) {
        return sharePrice * shares;
    }
    public void sell(int amount) {
        shares = shares - amount;
    }
    public String getSymbol() {
        return symbol;
    }
}
```

Q: What are some possible test cases for this class?

- make sure it calculates the correct value
- make sure a sell is recorded correctly
- make sure the symbol is stored/retrieved correctly
Let’s write unit tests for these:

- Create 10 shares of ORCL
- The value of the stock at $30/share should be $300
- Create 10 shares of ORCL
- After selling 5 shares, the value at $30/share should be $150
- Create 10 shares of ORCL
- After selling 10 shares, the value at $30/share should be $0

Before we get started ...

- Need to download the JUnit jar file (Java Archive)
  - I’m using junit-4.10.jar
  - put the jar in the same dir as Stock.java
  - we need the jar file to run our tests
- All our tests are going to go into a testing class
  - e.g., called StockTest
Each test is placed in a separate method in the test class:

```java
import org.junit.*;
import static org.junit.Assert.*;

public class StockTest {
    @Test
    public void shouldGive300DollarsFor10Shares() {
        Stock stock = new Stock(10, "ORCL");
        assertTrue(stock.calculateValue(30) == 300);
    }
}
```

To **compile**, we need to include the jar file ...

```
javac -cp .:junit-4.8.2.jar *.java
```

– on windows, you would use ";" instead of ";"

To **run** our test we use the JUnitCore class ...

```
$ java -cp .:junit-4.8.2.jar org.junit.runner.JUnitCore StockTest
JUnit version 4.10
.
Time: 0.005
OK (1 test)
```

• the ‘.’ means we ran one test and it passed!
Now add another unit test (another test method)

```java
import org.junit.*;
import static org.junit.Assert.*;

public class StockTest {
    @Test
    public void shouldGive300DollarsFor10Shares() {
        ...
    }

    @Test
    public void shouldGive150DollarsAfterSelling5Shares() {
        Stock s = new Stock(10, "ORCL");
        s.sell(5);
        assertTrue(s.calculateValue(30) == 150);
    }
}
```

Then compile again:

```
javac -cp .:junit-4.10.jar *.java
```

And run again:

```
$ java -cp .:junit-4.8.2.jar org.junit.runner.JUnitCore StockTest

JUnit version 4.10
.
Time: 0.006

OK (2 tests)
```
And now the third unit test

```java
import org.junit.*;
import static org.junit.Assert.*;

public class StockTest {
    @Test
    public void shouldGive300DollarsFor10Shares() { ... }
    @Test
    public void shouldGive150DollarsAfterSelling5Shares() { ... }
    @Test
    public void shouldGive0DollarsAfterSellingOver5Shares() {
        Stock s = new Stock(5, "ORCL");
        s.sell(10);
        assertTrue(s.calculateValue(30) == 0);
    }
}
```

Then compile and run again:

```bash
javac -cp .:junit-4.10.jar *.java
$ java -cp .:junit-4.8.2.jar org.junit.runner.JUnitCore StockTest
JUnit version 4.10
...E
Time: 0.075
There was 1 failure:
1) shouldGive0DollarsAfterSellingOver5Shares(StockTest)
java.lang.AssertionError
    ... exception message here ...

FAILURES!!!
Tests run: 3, Failures: 1