Section 4:
HW5, Graphs, and Testing

Slides by Vinod Rathnam
with material from Alex Mariakakis, Krysta Yousoufian, Mike Ernst, Kellen Donohue

AGENDA
- HW5
- Graphs
- JUnit Testing
- Test Script Language (Demo)
- JavaDoc (Demo)

DEMO: HW 5 STARTER FILES

GRAPHS

Nodes
Edges

Children of A

Parents of D

Path from A to C

Shortest path from A to C?

Shortest path from A to B?
INTERNAL VS. EXTERNAL TESTING

- **Internal**: JUnit
  - How you decide to implement the object
  - Checked with implementation tests
- **External**: test script
  - Your API and specifications
  - Testing against the specification
  - Checked with specification tests

A JUNIT TEST CLASS

- A method with @Test is flagged as a JUnit test
- All @Test methods run when JUnit runs

```java
import org.junit.*;
import static org.junit.Assert.*;
public class TestSuite {
    ...
    @Test
    public void TestName1() {
        ...
    }
}
```

USING JUNIT ASSERTIONS

- Verifies that a value matches expectations
  - `assertEquals(42, meaningOfLife());`
  - `assertTrue(list.isEmpty());`
- If the value isn’t what it should be, the test fails
  - Test immediately terminates
  - Other tests in the test class are still run as normal
  - Results show details of failed tests

- And others: [http://www.junit.org/apidocs/org/junit/Assert.html](http://www.junit.org/apidocs/org/junit/Assert.html)
  - Each method can also be passed a string to display if it fails:
    - `assertEquals("message", expected, actual)`

CHECKING FOR EXCEPTIONS

- Verify that a method throws an exception when it should
- Test passes if specified exception is thrown, fails otherwise
- Only time it’s OK to write a test without a form of asserts

```java
@Test(expected=IndexOutOfBoundsException.class)
public void testGetEmptyList() {
    List<String> list = new ArrayList<String>;
    list.get(0);
}
```

SETUP AND TEARDOWN

- Methods to run before/after each test case method is called:
  ```java
  @Before
  public void name() { ... }
  @After
  public void name() { ... }
  ```
- Methods to run once before/after the entire test class runs:
  ```java
  @BeforeClass
  public static void name() { ... }
  @AfterClass
  public static void name() { ... }
  ```
SETUP AND TEARDOWN

public class Example {
    List empty;
    @Before
    public void initialize() {
        empty = new ArrayList();
    }
    @Test
    public void size() {
        ...
    }
    @Test
    public void remove() {
        ...
    }
}

DON’T REPEAT YOURSELF

✓ Can declare fields for frequently-used values or constants
  private static final String DEFAULT_NAME = "MickeyMouse";
  private static final User DEFAULT_USER = new User("lazowska", "Ed", "Lazowska");

✓ Can write helper methods, etc.
  private void eq(RatNum ratNum, String rep) {
      assertEquals(rep, ratNum.toString());
  }
  private BinaryTree getTree(int[] items) {
      // construct BinaryTree and add each element in items
  }

#1: BE DESCRIPTIVE

✓ When a test fails, JUnit tells you:
  - Name of test method
  - Message passed into failed assertion
  - Expected and actual values of failed assertion

  The more descriptive this information is, the easier it is to diagnose failures

<table>
<thead>
<tr>
<th>Level of goodness</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>testAddDaysWithinMonth()</td>
</tr>
<tr>
<td>Not so good</td>
<td>testAddDays1(), testAddDays2()</td>
</tr>
<tr>
<td>Bad</td>
<td>test1(), test2()</td>
</tr>
<tr>
<td>Overkill</td>
<td>TestAddDaysOneDayAndThenFiveDaysStartingOnJanuaryTwentySeventhAndMakeSureItRollsBackToJanuaryAfterRollingToFebruary()</td>
</tr>
</tbody>
</table>

LET’S PUT IT ALL TOGETHER!

public class DateTest {
...

    // Test addsDays when it causes a rollover between months
    @Test
    public void testAddDaysWrapToNextMonth() {
        Date actual = new Date(2050, 2, 15);
        actual.addDays(14);
        Date expected = new Date(2050, 3, 1);
        assertEquals("date after +14 days", expected, actual);
    }
}
public class DateTest {
    ...
    // Test addDays when it causes a rollover between months
    @Test
    public void testAddDaysWrapToNextMonth() {
        Date actual = new Date(2050, 2, 15);
        actual.addDays(14);
        Date expected = new Date(2050, 3, 1);
        assertEquals("date after +14 days", expected, actual);
    }
}

#2: KEEP TESTS SMALL

- Ideally, test one thing at a time
  - "Thing" usually means one method under one input condition
  - Not always possible – but if you test x() using y(), try to test y() in isolation in another test
- Low-granularity tests help you isolate bugs
  - Tell you exactly what failed and what didn’t
- Only a few (likely one) assert statements per test
  - Test halts after first failed assertion
  - Don’t know whether later assertions would have failed

#3: BE THOROUGH

- Consider each equivalence class
  - Items in a collection: none, one, many
- Consider common input categories
  - Math.abs(): negative, zero, positive values
- Consider boundary cases
  - Inputs on the boundary between equivalence classes
    - Person.isMinor(): age < 18, age == 18, age > 18
- Consider edge cases
  - -1, 0, 1, empty list, arr.length, arr.length-1
- Consider error cases
  - Empty list, null object
  
How To Create JUnit Test Classes

- Right-click hw5.test -> New -> JUnit Test Case
- Important: Put class name in ImplementationTests.java
- Demo
JUnit Asserts vs. Java Asserts

- We've just been discussing JUnit assertions so far
- Java itself has assertions

```java
public class LitterBox {
    ArrayList<Kitten> kittens;
    public Kitten getKitten(int n) {
        assert(n >= 0);
        return kittens.get(n);
    }
}
```

Assertions vs. Exceptions

- Assertions should check for things that should never happen
- Exceptions should check for things that might happen
- “Exceptions address the robustness of your code, while assertions address its correctness”

```java
public class LitterBox {
    ArrayList<Kitten> kittens;
    public Kitten getKitten(int n) {
        try {
            return kittens.get(n);
        } catch(Exception e) {
        }
    }
}
```

Reminder: Enabling Asserts in Eclipse

To enable asserts:
- Go to Run -> Run Configurations… -> Arguments tab -> input -ea in VM arguments section
- Do this for every test file

Demo!

Expensive CheckReps

- So, before your final commit, a nice thing to do is to remove the checking of expensive parts of your checkRep or the checking of your checkRep entirely
- For example, one thing you can do is have a boolean flag and structure your checkRep as so:

```java
private void checkRep() {
    cheap-stuff
    if (DEBUG_FLAG) { // or can have this for entire checkRep
        expensive-stuff
    }
    cheap-stuff
}
```

External Tests: Test Script Language

Expensive CheckReps

- Ant Validate and Staff Grading will have assertions enabled
- But sometimes a checkRep can be expensive
- For example, looking at each node in a Graph with a large number of nodes
- This could cause the grading scripts to timeout
TEST SCRIPT LANGUAGE
- Text file with one command listed per line
- First word is always the command name
- Remaining words are arguments
- Commands will correspond to methods in your code

Example:
- CreateGraph graph1
- AddNode graph1 n1
- AddNode graph1 n2
- AddEdge graph1 n1 n2 e1
- ListNodes graph1
- ListChildren graph1 n1

JAVADOC API
- Now you can generate the JavaDoc API for your code (Optional)
- Instructions online: [http://courses.cs.washington.edu/courses/cse331/15sp/tools/editing-compiling.html#javadoc](http://courses.cs.washington.edu/courses/cse331/15sp/tools/editing-compiling.html#javadoc)
- Demo: Generate JavaDocs

DEMO: TEST SCRIPT LANGUAGE

How To Create Specification Tests
- Create .test and .expected file pairs under hw5.test
- Find correct format for expected output in hw5 instructions
- Implement parts of HWSTestDriver
  - driver connects commands from .test file to your Graph implementation to the output which is matched with .expected file
- Run all tests by running SpecificationTests.java
  - Note: staff will have our own .test and .expected pairs to run with your code
  - So do not hardcode .test/.expected pairs to pass, but instead make sure the format in hw5 instructions is correctly followed

DEMO: TEST SCRIPT LANGUAGE

JAVADOC API
- Now you can generate the JavaDoc API for your code (Optional)
- Instructions online: [http://courses.cs.washington.edu/courses/cse331/15sp/tools/editing-compiling.html#javadoc](http://courses.cs.washington.edu/courses/cse331/15sp/tools/editing-compiling.html#javadoc)
- Demo: Generate JavaDocs