Introduction to JUnit

CS 4501 / 6501
Software Testing

[Ammann and Offutt, “Introduction to Software Testing”]
Today’s Objective

• Understand JUnit test classes
• Understand anatomy of basic JUnit test methods
• JUnit assertions and other features
• Associated patterns for test programming
What is JUnit?

- An open source Java testing framework (junit.org) used to write and run repeatable automated tests

- JUnit is widely used in industry

- A structure for writing test drivers

- JUnit features include
  - Assertions to evaluate expected results
  - The ability to share common test data among tests
  - Test sets to easily organize and run tests
  - The ability to run tests from either a command line or a GUI
JUnit Tests

- For unit and integration testing
  - Entire object, part of an object (a method or some interacting methods), and interaction between several objects

- One test case in one test method

- A test class contains one or more test methods

- Test classes include
  - A collection of test methods
  - Method to set up the state before running each test (prefix)
  - Method to update the state after each test (postfix)
  - [Optional] Method to set up and update before and after all tests
Writing JUnit Tests (JUnit4)

• Download necessary jar files at junit.org

• Use the methods of the org.junit.Assert class
  • Refer to Javadoc for a complete API

• Each test method
  • Checks a condition (assertion)
  • Reports to the test runner whether the test failed or succeeded

• The test runner uses the result to report to the user

• All of the methods return void
Common Methods

- `assertTrue(boolean condition)`
  - Assert that a condition is true

- `assertTrue(String message, boolean condition)`
  - Assert that a condition is true
  - If the assertion is true, the string is ignored. Otherwise, the string is sent to the test engineer.

- `assertEquals(Object expected, Object actual)`
  - Assert that two objects are equal

- `fail(String message)`
  - If a certain situation is expected when a certain section of code is reached, the string is sent to the test engineer.
  - Often used to test exceptional behavior
JUnit – Test Classes

```java
package inclass;

import static org.junit.Assert.*;
import org.junit.Test;

public class ArrayOperationsNumZeroTest {

    @Test
    public void testNumZeroEmptyArray() {
        int[] x = {};  // zero-sized array
        int n = ArrayOperations.numZero(x);
        assertEquals("0 zeros", 0, n);
    }

    @Test
    public void testNumZeroArrayWithNoZeros() {
        int[] x = {1, 2, 3};
        int n = ArrayOperations.numZero(x);
        assertEquals("0 zeros in an array with no zeros", 0, n);
    }
}
```
JUnit – Test Methods

1) Setup test case values

```java
@Test
public void testNumZeroArrayWithNoZeros() {
    int[] x = {1, 2, 3};
    int n = ArrayOperations.numZero(x);
    assertEquals("0 zeros in an array with no zeros", 0, n);
}
```

2) Execute program under test

3) Assert expected vs. actual test outputs

- expected
- actual output
Let’s do some exercises
Wrap-up

• Test automation
• Testability, Observability, Controllability
• Components of a test case
• JUnit
  • Test class, test method
  • Common methods: assertTrue, assertEquals, fail

What’s Next?
• More JUnit