Testing and JUnit
Why Test Code?

- Testing makes code better
- Find bugs in development instead of when the product has been released!
- Helps improve design of code
- Makes the code easier to understand for developers
- [https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks](https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks)
What Should We Test?

- General functionality of code - in a perfect world, does the code do what we want it to do?
- Test for “edge cases” - special cases that come up less often and may not be obvious to the programmer, but still can arise in use of the program
- Note: Even with testing, there can still be bugs!
- It is important to test early on in development, and continue through the process - “regression testing”
- Testing should be as exhaustive as possible
Test Cases

- A test case describes some action that the programmer expects their code to perform
- Test cases should be as exhaustive as possible - we want to test the full functionality of our code, including edge cases!
- Test cases should also be *unique*
Black Box Testing

- This is “functional testing”
- Tester does not know the details of the code
- Given some input, the test expects some output
- Often done by QA/people not familiar with the details of the code
White Box Testing

- Software unit testing
- Done by the developers
- Test cases are designed based on the internal structure of the code
What is JUnit?

- JUnit is a framework for writing unit tests in Java
- Unit test: test of a class
- Test case: a test of a single method in a class
Using JUnit

- In general:
  - public class OurTests {
    @Test
    protected void runOurTest() {
      // Our test goes here!
    }
  }

How Do JUnit Tests Actually Work?

- Tests do not have a return type - they are void functions
- Upon success, a test will do nothing
- Upon failure, the test will throw an AssertionError
- This error is handled by JUnit, no extra work for the programmer!
How Tests Pass or Fail

- In JUnit tests, the programmer asserts a condition
- If the assertion is true, the test passes
- If the assertion is false, the test fails
- JUnit provides many assert functions
Types of Asserts

- `assertEquals(boolean expected, boolean actual)`
- `assertTrue(boolean condition)`
- `assertFalse(boolean condition)`
- `assertNotNull(Object object)`
- `assertNull(Object object)`
- `assertSame(object1, object2)`
- `assertNotSame(object1, object 2)`
- `assertArrayEquals(expectedArray, resultArray)`
import org.junit.Test;
import static org.junit.Assert.*;

public class SampleTest {

    @Test
    public void simpleTest() {
        assertEquals(1, 1);
    }

    @Test
    public void simpleTest2() {
        assertTrue(false);
    }
}
Let’s Try It Out!