Unit Testing

How do you know if your code is correct?

Testing Your Code

- Submit to AutoLab?
  - Decent for education
  - Does not exist outside of class
  - Need a way to test code on your own

- Call your methods in a main method?
  - A great start!
  - Must manually check all the printed values
  - Tedious for large projects

Testing Your Code

- Unit Testing!
  - Write code to automate testing

Unit Testing

- Run a series of tests on your code
  - Call your method with a specific input
  - Verify that it returned the correct output
- If your code returns the correct output on **ALL** the tests, it passes
- If your code fails a single test, it fails
- You want your code to be correct on ALL inputs

Unit Testing

Scenario

- You were given a programming task
  - "Write a method named addFive that takes an int as a parameter and returns the input plus five as an int"

```
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- You write this wonderful code and you want to test it to make sure it's correct

Unit Testing

Scenario

- You write a main method
- You call your method a few times
- You print the return values to the screen
- You verify with your eyes that what was printed makes sense
- You feel great about the results!

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

    public static void main(String[] args) {
        System.out.println(addFive(2));
        System.out.println(addFive(5));
        System.out.println(addFive(1));
    }

}
```

```
7
10
6
```

Unit Testing

- But what do you do when the the code is harder to verify like this
  - "Write a method that sorts 100s of Songs by title or artist"
- We want to move on to automated testing
  - Write testing code
  - Run that code to test your method

```
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

    public static void main(String[] args) {
        System.out.println(addFive(2));
        System.out.println(addFive(5));
        System.out.println(addFive(1));
    }

}
```

```
7
10
6
```

Unit Testing

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- Let's look at our first unit test
- Testing will be defined in a separate file/class

Unit Testing

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- We will use the JUnit library for testing
- JUnit does not come with java and is installed using the pom.xml file included in the project code (IntelliJ installs this automatically)

Unit Testing

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- We'll use a method from this library called assertTrue which must be imported
- import static: We want to import a static method from a class without importing the entire class - avoid typing Assert.assertTrue

Unit Testing

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- Each test you write will be defined by a public method (Note: Not a static method)
- To tell JUnit that the method defines a test, annotate it with @Test

Unit Testing

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- The @Test annotation must be included
- Common cause of errors is to miss this annotation

Unit Testing

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- Each test method has a name which will be the name of that test
- In this course - These names will appear in AutoLab to give you more information about your tests

Unit Testing

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- Finally, we can write *test cases*
  - A test case tests a single input/output pair
- This *test class* contains one *test* that has 3 *test cases*

Unit Testing

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- For each test case, call assertTrue with a boolean expression that you expect to resolve to true
- If assertTrue is ever called with a value of false, the code fails the entire test class - We want 0 bugs in our code!

Unit Testing

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- Note: There is no main method in this file
- JUnit will use this code through it's own main method
- IntelliJ understands JUnit and gives you convenient run buttons

Unit Testing

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- Is this enough testing?
  - A question that can always be asked, and never fully answered

Unit Testing

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- We have 3 small positive integers which represent *common* test cases for this method
- These are simple inputs that everyone would expect

Unit Testing

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive_noNegatives(int x){
        return Math.abs(x)+5;
    }

}
```

- Here is an **incorrect** solution for the addFive method
- Our test passes this incorrect solution!
- Not enough testing

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
        assertTrue(Adder.addFive(10) == 15);
        assertTrue(Adder.addFive(100) == 105);
    }

    @Test
    public void testAddFiveWithNegatives() {
        assertTrue(Adder.addFive(-1) == 4);
        assertTrue(Adder.addFive(-5) == 0);
        assertTrue(Adder.addFive(-10) == -5);
        assertTrue(Adder.addFive(-51) == -46);
        assertTrue(Adder.addFive(-100) == -95);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- Let's add some *uncommon* cases
- Negative inputs are a more unusual input that might expose a bug in the code

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
        assertTrue(Adder.addFive(10) == 15);
        assertTrue(Adder.addFive(100) == 105);
    }

    @Test
    public void testAddFiveWithNegatives() {
        assertTrue(Adder.addFive(-1) == 4);
        assertTrue(Adder.addFive(-5) == 0);
        assertTrue(Adder.addFive(-10) == -5);
        assertTrue(Adder.addFive(-51) == -46);
        assertTrue(Adder.addFive(-100) == -95);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- We can group our test cases into multiple Tests in the same class
- Annotate each test with @Test

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
        assertTrue(Adder.addFive(10) == 15);
        assertTrue(Adder.addFive(100) == 105);
    }

    @Test
    public void testAddFiveWithNegatives() {
        assertTrue(Adder.addFive(-1) == 4);
        assertTrue(Adder.addFive(-5) == 0);
        assertTrue(Adder.addFive(-10) == -5);
        assertTrue(Adder.addFive(-51) == -46);
        assertTrue(Adder.addFive(-100) == -95);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive_noNegatives(int x){
        return Math.abs(x)+5;
    }

}
```

- This test class:
  - Passes the correct solution
  - Rejects this incorrect solution

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
        assertTrue(Adder.addFive(10) == 15);
        assertTrue(Adder.addFive(100) == 105);
    }

    @Test
    public void testAddFiveWithNegatives() {
        assertTrue(Adder.addFive(-1) == 4);
        assertTrue(Adder.addFive(-5) == 0);
        assertTrue(Adder.addFive(-10) == -5);
        assertTrue(Adder.addFive(-51) == -46);
        assertTrue(Adder.addFive(-100) == -95);
    }

}
```

```java
public static int addFive_badOnZero(int x) {
    if(x > 0){
        return Math.abs(x) + 5;
    }else if(x < 0){
        return -Math.abs(x) + 5;
    }else{
        return x;
    }
}
```

- And passes this incorrect solution

- Not enough testing!
  - Your tests should be able to identify an incorrect solution by containing at least one test case where the solution fails

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
        assertTrue(Adder.addFive(10) == 15);
        assertTrue(Adder.addFive(100) == 105);
    }

    @Test
    public void testAddFiveWithNegatives() {
        assertTrue(Adder.addFive(-1) == 4);
        assertTrue(Adder.addFive(-5) == 0);
        assertTrue(Adder.addFive(-10) == -5);
        assertTrue(Adder.addFive(-51) == -46);
        assertTrue(Adder.addFive(-100) == -95);
    }

    @Test
    public void testAddFiveEdgeCase() {
        assertTrue(Adder.addFive(0) == 5);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- We also want to check the *edge* cases
- These are any inputs that can expose unique bugs in the code
- Typical edge case inputs: 0, "", an empty ArrayList, an empty HashMap

```java
package week2;

import org.junit.Test;

import static org.junit.Assert.assertTrue;

public class AddTest {

    @Test
    public void testAddFive() {
        assertTrue(Adder.addFive(2) == 7);
        assertTrue(Adder.addFive(5) == 10);
        assertTrue(Adder.addFive(1) == 6);
        assertTrue(Adder.addFive(10) == 15);
        assertTrue(Adder.addFive(100) == 105);
    }

    @Test
    public void testAddFiveWithNegatives() {
        assertTrue(Adder.addFive(-1) == 4);
        assertTrue(Adder.addFive(-5) == 0);
        assertTrue(Adder.addFive(-10) == -5);
        assertTrue(Adder.addFive(-51) == -46);
        assertTrue(Adder.addFive(-100) == -95);
    }

    @Test
    public void testAddFiveEdgeCase() {
        assertTrue(Adder.addFive(0) == 5);
    }

}
```

```java
package week2;

public class Adder {

    public static int addFive(int x){
        return x+5;
    }

}
```

- Your goal when testing:
  - Write at least one test case that will expose any possibly bug that could exist in the code being tested
- Unsure exactly how to do that?
  - Write **LOTS** of tests!
- Testing will often contain more code than what's being tested!

Testing in CSE116

Testing in CSE116

- When a programming task requires test, your tests are ran:
  - Against a correct solution stored on the server
  - Against a variety of incorrect solutions stored on the server

- Your test suite must pass the correct solution
  - If your tests reject the correct solution, there is something wrong with what you're testing that you must correct before moving on

- Your test suite should fail **all** the incorrect solutions
  - Your tests should be thorough enough to correctly fail/reject every incorrect solution
  - It is enough to have a single test case fail a solution to reject the entire solution

Testing Tips

```
@Test
public void testAddFiveEdgeCase() {
    assertTrue(Adder.addFive(0) == 5);
}
```

- assertTrue is very versatile
  - Behaves like a conditional
  - You can test any boolean expression you can write

Testing Tips

```java
@Test
public void testAddFiveEdgeCase() {
    assertTrue(Adder.addFive(0) == 5);
}
```

```java
@Test
public void testAddFiveEdgeCase_withAssertEquals() {
    int result = Adder.addFive(0);
    assertEquals(result, 5);
}
```

- If you are checking 2 values for equality
  - It's better to use assertEquals
  - Takes 2 arguments and passes if they are equivalent
  - Provides helpful output if the test case fails

Testing Tips

```java
@Test
public void testAddFiveEdgeCase() {
    assertTrue(Adder.addFive(0) == 5);
}
```

```java
@Test
public void testAddFiveEdgeCase_withAssertEquals() {
    int result = Adder.addFive(0);
    assertEquals(result, 5);
}
```

```java
@Test
public void testAddFiveEdgeCase_withHintText() {
    int result = Adder.addFive(0);
    assertEquals("Expected 5 on input 0, got: " + result, result, 5);
}
```

- You can add hint text to any test case
- Add a argument that's a String before all other arguments
- This String is printed if the test case fails
  - Gives you a hint of where to look for the bug