

JUNIT TESTING

CS 030

Abstract Models for Concrete
Problems using Java

Prof. Donald J. Patterson

JUNIT TESTING

TESTING

- Testing is the process of validating your code is working according to the specifications.
- It is done at the developer level.
- Unit testing is the testing of a single entity (class or method).
- Unit testing is critical to professional software development to ensure quality.



JUNIT TESTING

WHY TEST?

- To make sure that your code meets specifications
- To check corner cases
- Because fixing before deployment is much easier than fixing after deployment



JUNIT TESTING

WHAT IS JUNIT?

- It is a “Regression Testing Framework”
- It is used to standardize the testing process
 - so that automated tools can help you do testing
 - fast and regularly
 - within Eclipse
 - also continuous deployment environments
 - Jenkins
 - Travis-CI
- JUnit tests can also be run from the command-line



JUNIT TESTING

WHAT IS JUNIT?

- JUnit is specifically for Java
- It is open source



JUNIT TESTING

WHAT IS A UNIT TEST CASE?

- A Unit Test Case is a test code which validates that production code (method) works as expected.
- To do this quickly and efficiently it helps to have a framework: JUnit.
- A unit test case has a known input and an expected output, which is worked out **before** the test is executed.
- The known input should test a precondition and the expected output should test a postcondition.



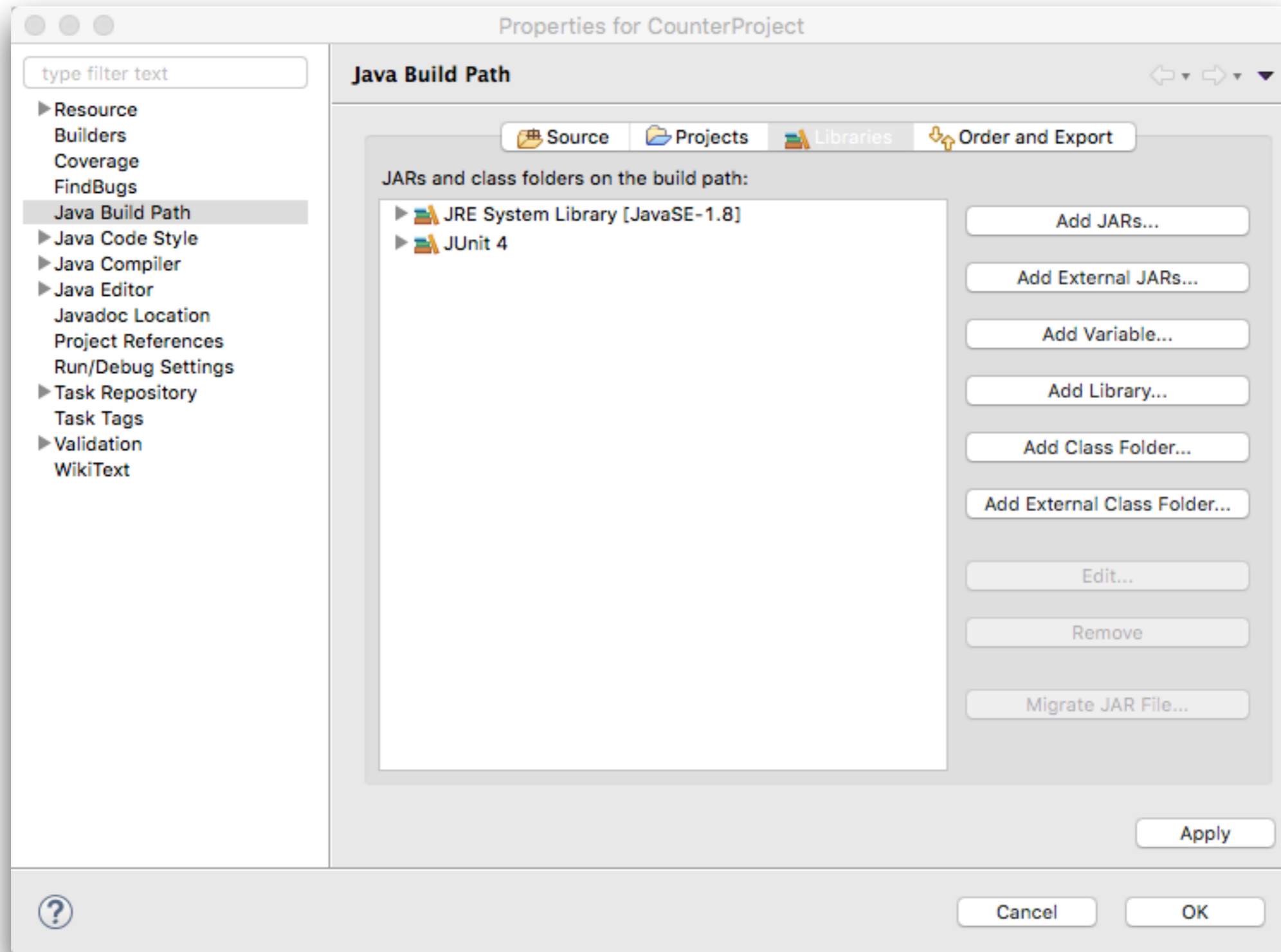
JUNIT TESTING

WHAT IS A UNIT TEST CASE?

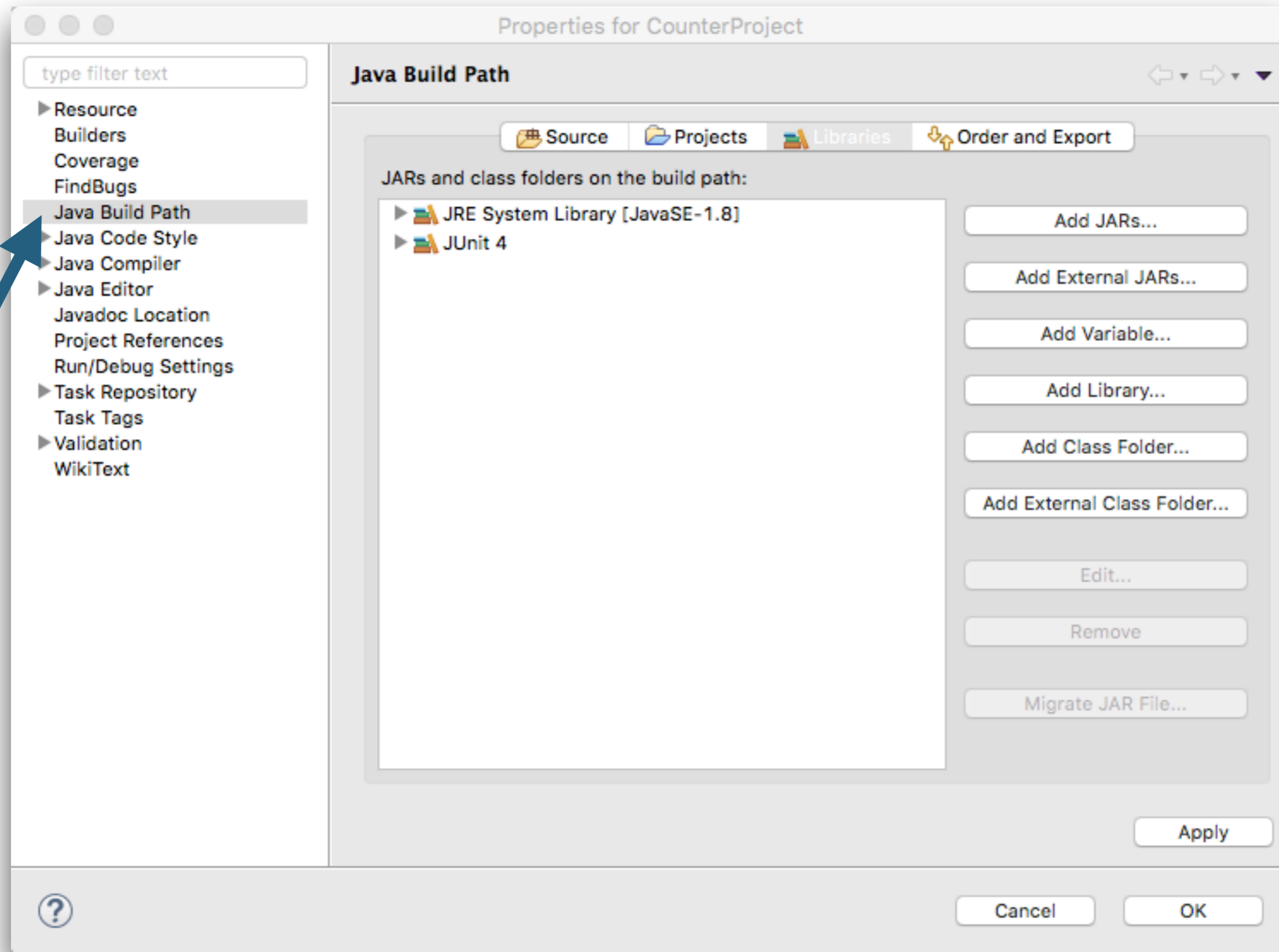
- Ideally you would have enough test cases that all possible paths through your code would be tested.
- In practice there is never enough time or money to do this
- Aim for 70-80% test coverage and focus on the important cases first



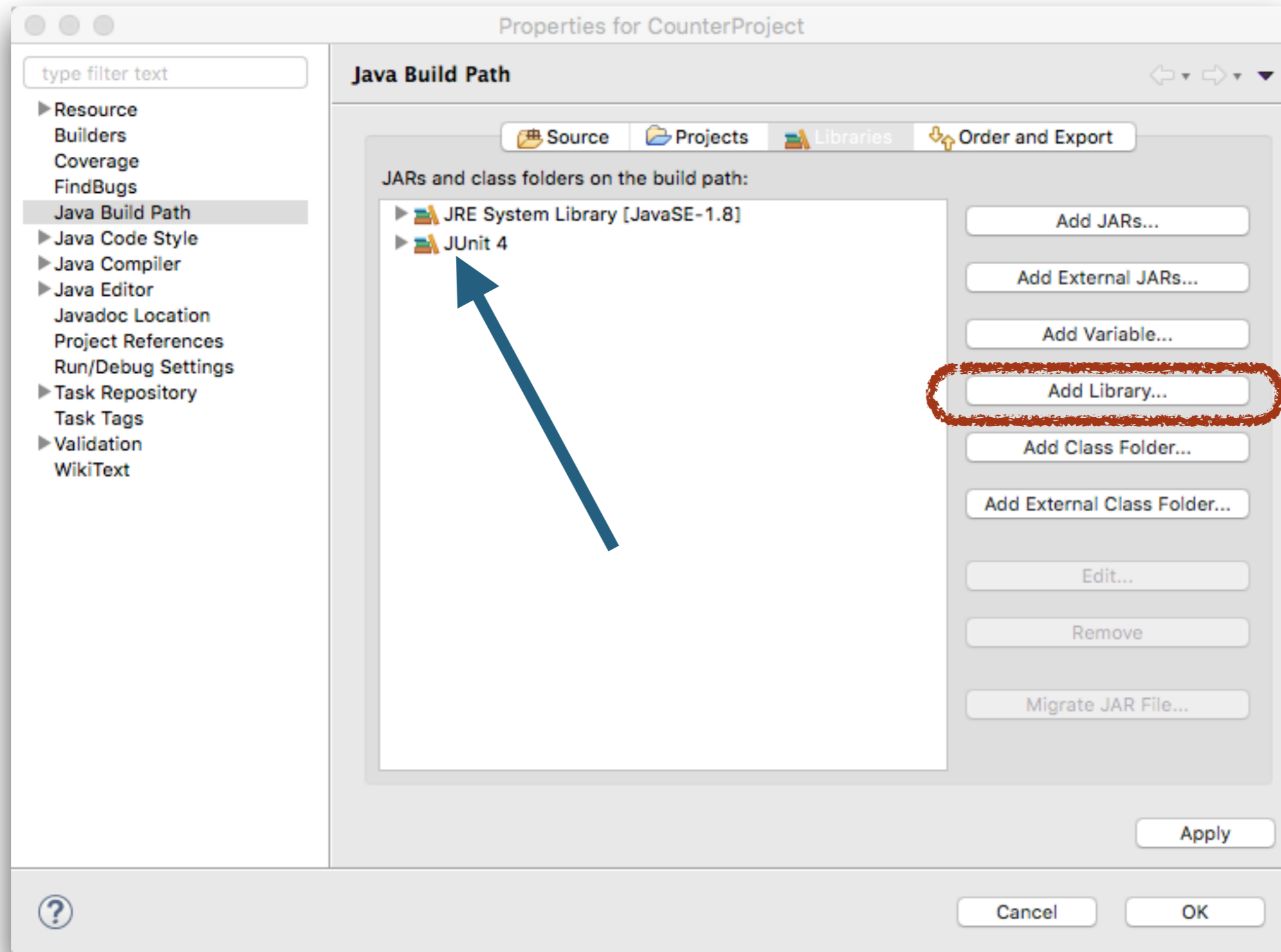
JUNIT TESTING



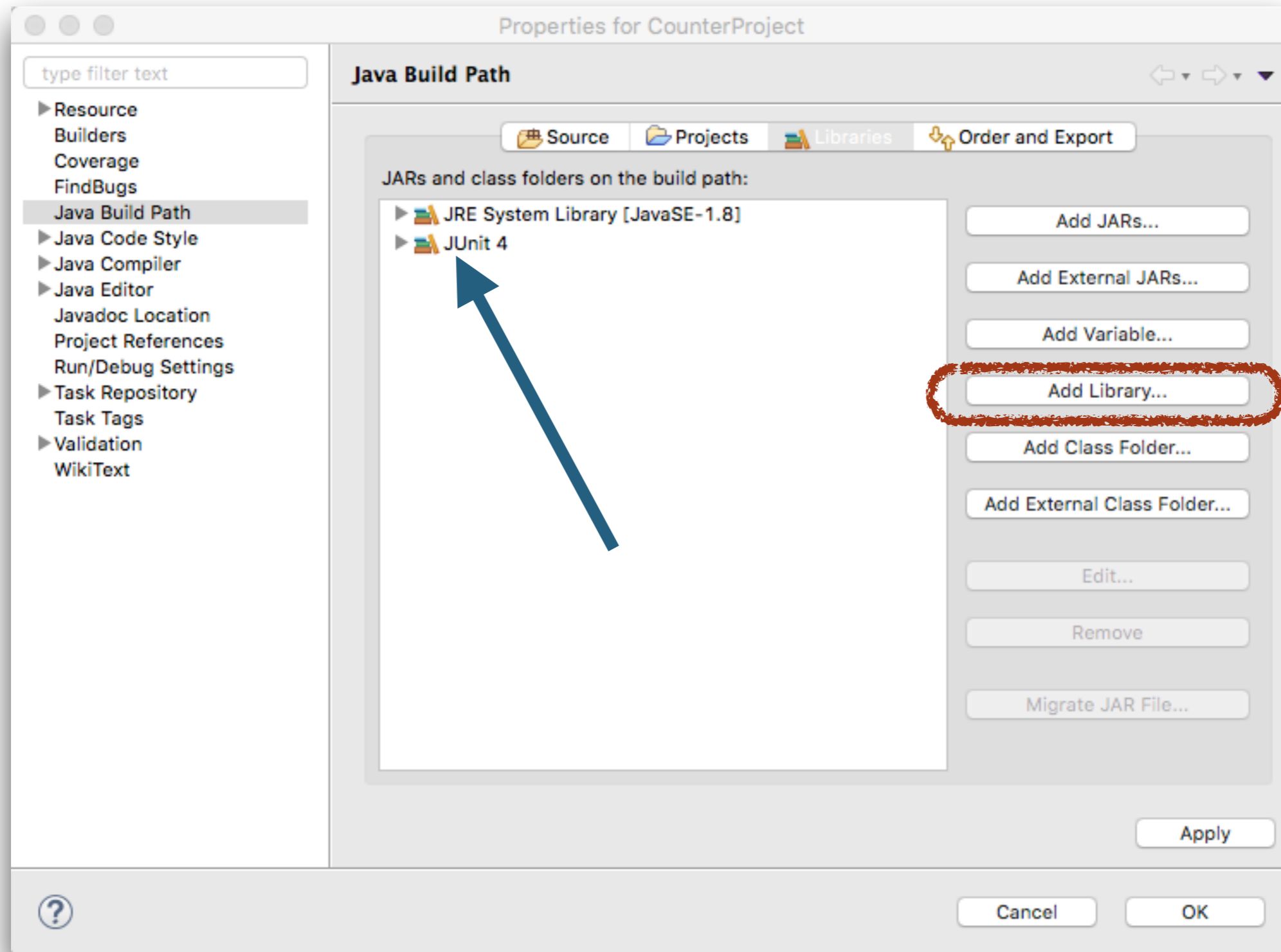
JUNIT TESTING



JUNIT TESTING



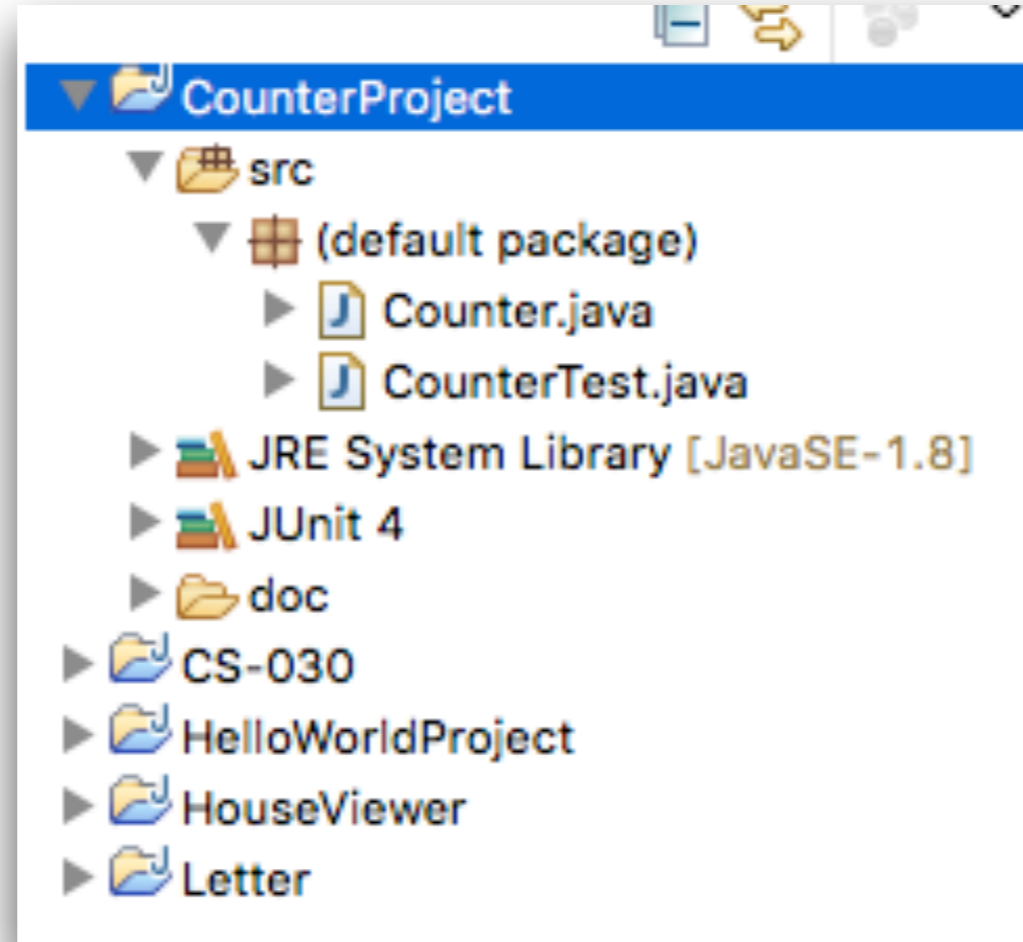
JUNIT TESTING



JUNIT TESTING

A TEST CASE IS ANOTHER CLASS

- It only exists to validate a different class
- Usually there is a one to one mapping between test classes and production classes



JUNIT TESTING

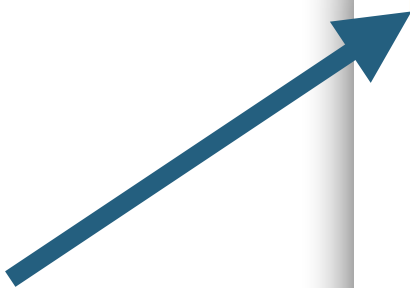
A TEST IS INDICATED BY AN ANNOTATION

```
@Test
public void testClick() {
    Counter testCounter = new Counter();
    testCounter.click();
    assertTrue(testCounter.getValue() == 1);
}
```



JUNIT TESTING

A TEST IS INDICATED BY AN ANNOTATION



```
@Test
public void testClick() {
    Counter testCounter = new Counter();
    testCounter.click();
    assertTrue(testCounter.getValue() == 1);
}
```



JUNIT TESTING

THE "TEST" IS DONE BY SPECIAL METHODS

Methods & Description
void assertEquals(boolean expected, boolean actual) Check that two primitives/Objects are equal
void assertFalse(boolean condition) Check that a condition is false
void assertNotNull(Object object) Check that an object isn't null.
void assertNull(Object object) Check that an object is null
void assertTrue(boolean condition) Check that a condition is true.
void fail() Fails a test with no message.



JUNIT TESTING

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

import org.junit.Test;

public class CounterTest {

    @Test
    public void testGetValue() {
        Counter testCounter = new Counter();
        testCounter.initializeCounter(0);
        assertEquals(0, testCounter.getValue());
    }

    @Test
    public void testClick() {
        Counter testCounter = new Counter();
        testCounter.click();
        assertEquals(1, testCounter.getValue());
    }
}
```

```
@Test
public void testReset() {
    Counter testCounter = new Counter();
    testCounter.click();
    testCounter.click();
    testCounter.click();
    testCounter.click();
    testCounter.click();
    testCounter.reset();
    assertTrue(testCounter.getValue() == 0);
}

@Test
public void testInitializeCounter() {
    Counter testCounter = new Counter();
    testCounter.click();
    testCounter.initializeCounter(0);
    assertTrue(testCounter.getValue() == 0);
}
```

```
}
```



JUNIT TESTING

REPEAT CODE

- BeforeClass
- AfterClass
- Before
- After

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

public class CounterTest2 {

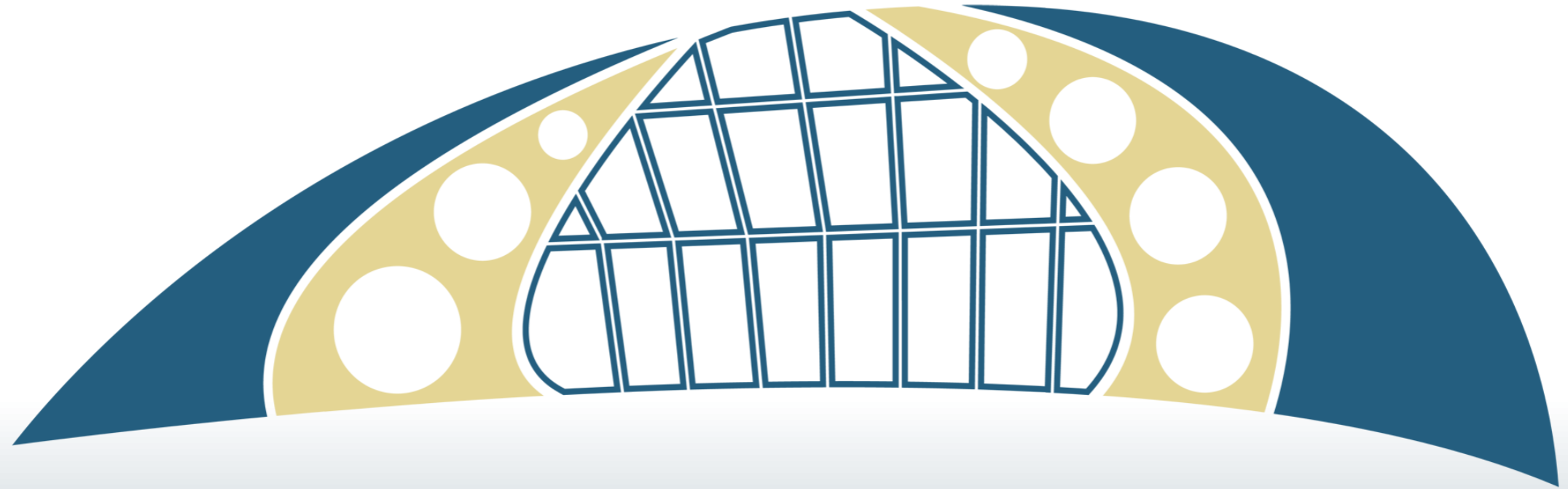
    @BeforeClass
    public static void setUpBeforeClass() throws Exception {
    }

    @AfterClass
    public static void tearDownAfterClass() throws Exception {
    }

    @Before
    public void setUp() throws Exception {
    }

    @After
    public void tearDown() throws Exception {
    }

    @Test
    public void test() {
        fail("Not yet implemented");
    }
}
```



WESTMONT **INSPIRED**
— COMPUTING LAB —