CS489:
Applied Software Development

Author: Prof. O. Kalu
MIU Computer Science - November 2023

Lesson 11:

Testing

Part 1:

Introduction to Testing

Wholeness

- Software Testing is the process of evaluating and verifying that a software product or application does what it is supposed to do.
- The goal of testing is to prevent or find and remove bugs, reduce development cost and enhance the performance of the application.
- Science of Consciousness: Action leads to Achievement. *Achievement leads to Fulfillment.*

Introduction to Automated Testing

- To understand automated testing, let's consider the manual testing process which goes as follows:
  - Write code
  - Compile (Build) to executable form
  - Execute the code manually (in some cases, enter input data in a form etc.)

Introduction to Automated Testing

- Check the result (such as output on the screen or values of variables or database data or output log files etc.)
- If it does not work, if the result is incorrect, we repeat the above process

- In Automated Testing, these above steps are perform using code

Automated versus Manual Testing

- Some aspects of application software require manual testing. Such as:
  - Finding strange edge cases
  - Judging about the aesthetics and visual design
  - Judging about the overall user experience

Automated versus Manual Testing

- However, Automated testing offers the following advantages:
  - Tests can be run many times, over and over
  - Tests are quicker to run
  - Tests can be run anytime
  - Excellent for checking mechanical & logical assertions

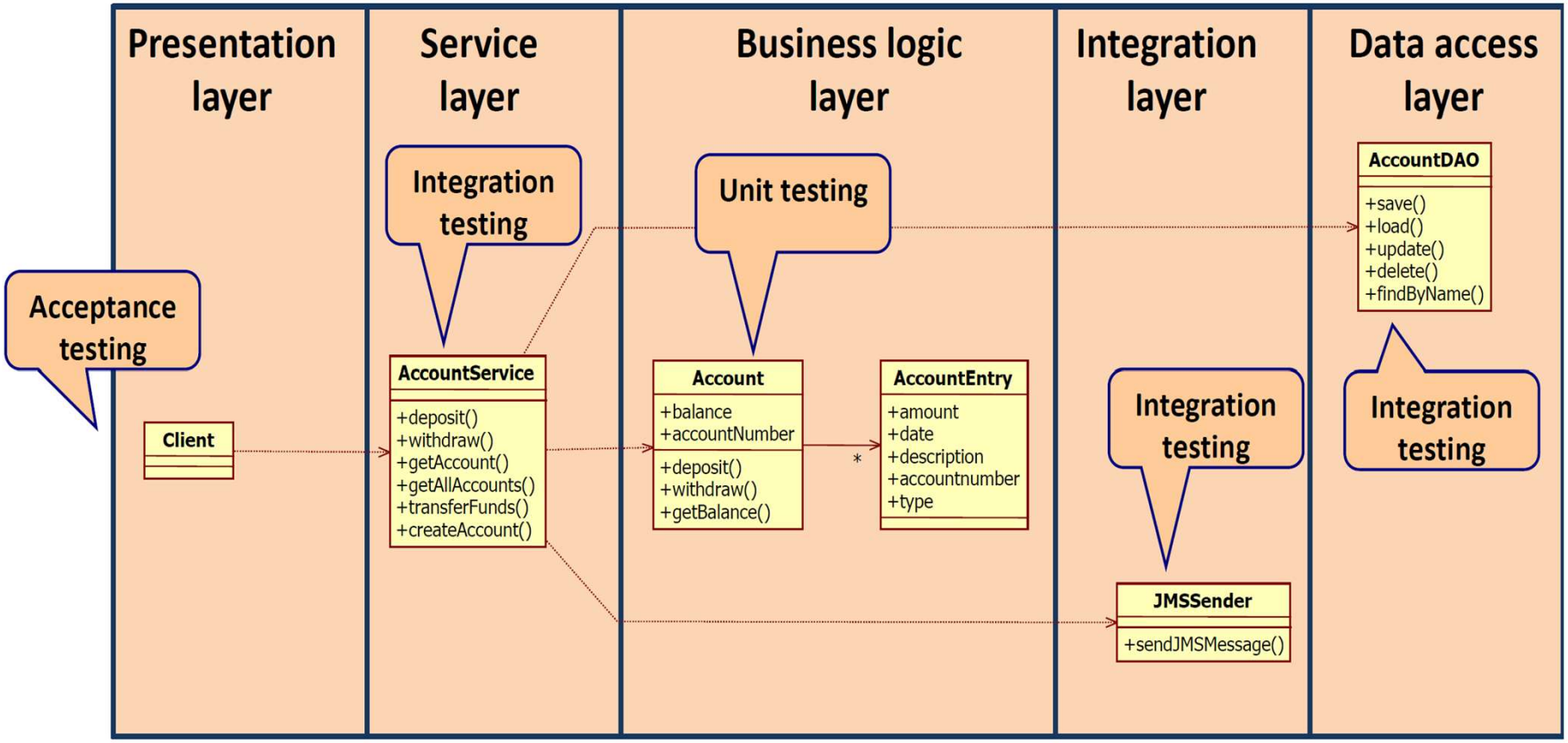
Automated versus Manual Testing

Manual test	Automated test
Boring	Challenging
Not reusable	Reusable
Complex manual setup and teardown	Automated setup and teardown
High risk to forget something	Zero risk to forget something
No safety net	Safety net
Low training value	Acts as documentation
Slow and time consuming	Fast

Unit versus Integration Testing

- Unit testing
  - We stay within the boudaries
    - Component boundary
    - Layer boundary
  - Deals mostly with testing business logic
- Integration testing
  - We go outside our boundary
    - Call the database
    - Send a message
    - Read a file from disk
  - Often time consuming operations

Different kinds of Tests



Unit Testing with JUnit Framework

- What is Unit Testing?
  - A unit test is a test that test one single class.
    - A test case test one single method
    - A test class test one single class
    - A test suite is a collection of test classes
  - Unit tests make use of a testing framework
  - A unit test
    1. Create an object
    2. Call a method
    3. Check if the result is correct

Example of Unit Testing

```
package count;

public class Counter {
    private int counterValue=0;

    public int increment(){
        return ++counterValue;
    }
    public int decrement(){
        return --counterValue;
    }
    public int getCounterValue() {
        return counterValue;
    }
}
```

Example of Unit Testing

```java
public class Counter {
    private int counterValue=0;

    public int increment(){
       return ++counterValue;
    }
    public int decrement(){
       return --counterValue;
    }
    public int getCounterValue() {
       return counterValue;
    }
}
```

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

public class CounterTest {
    private Counter counter;

     @Before
     public void setUp() throws Exception {
       counter = new Counter();
     }
     @Test
     public void testIncrement(){
        assertEquals("Counter.increment does not work correctly",1,counter.increment());
        assertEquals("Counter.increment does not work correctly",2,counter.increment());
     }
     @Test
     public void testDecrement(){
        assertEquals("Counter.decrement does not work correctly",-1,counter.decrement());
        assertEquals("Counter.decrement does not work correctly",-2,counter.decrement());
     }
}
```

Initialization

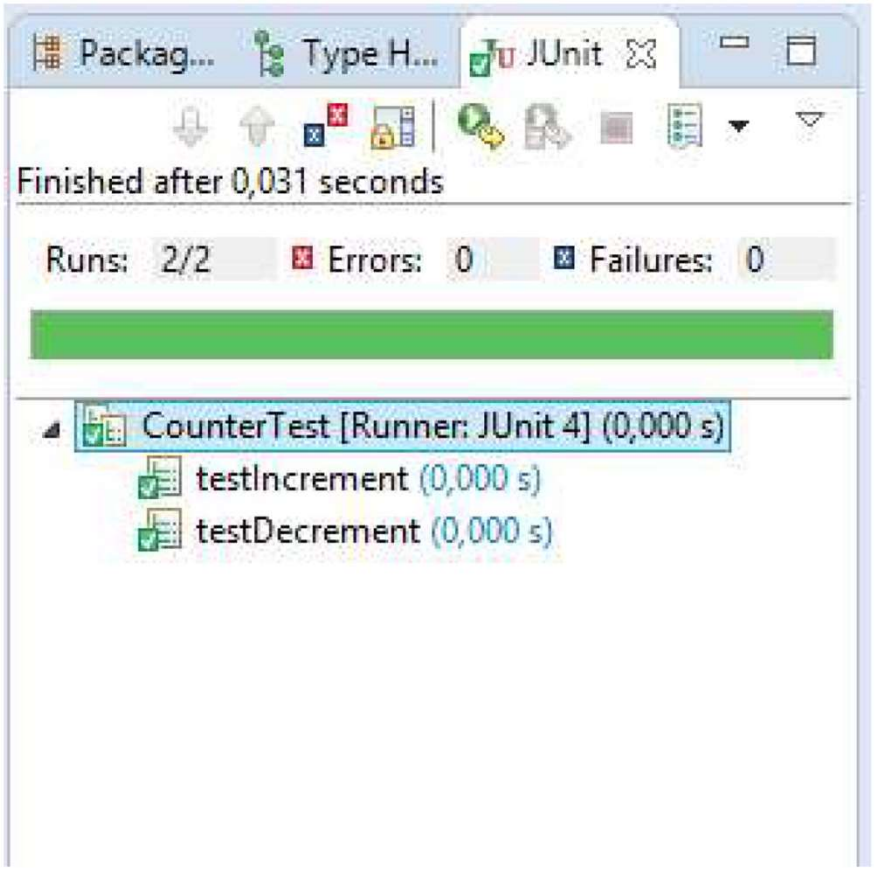
Test method

Test method

17

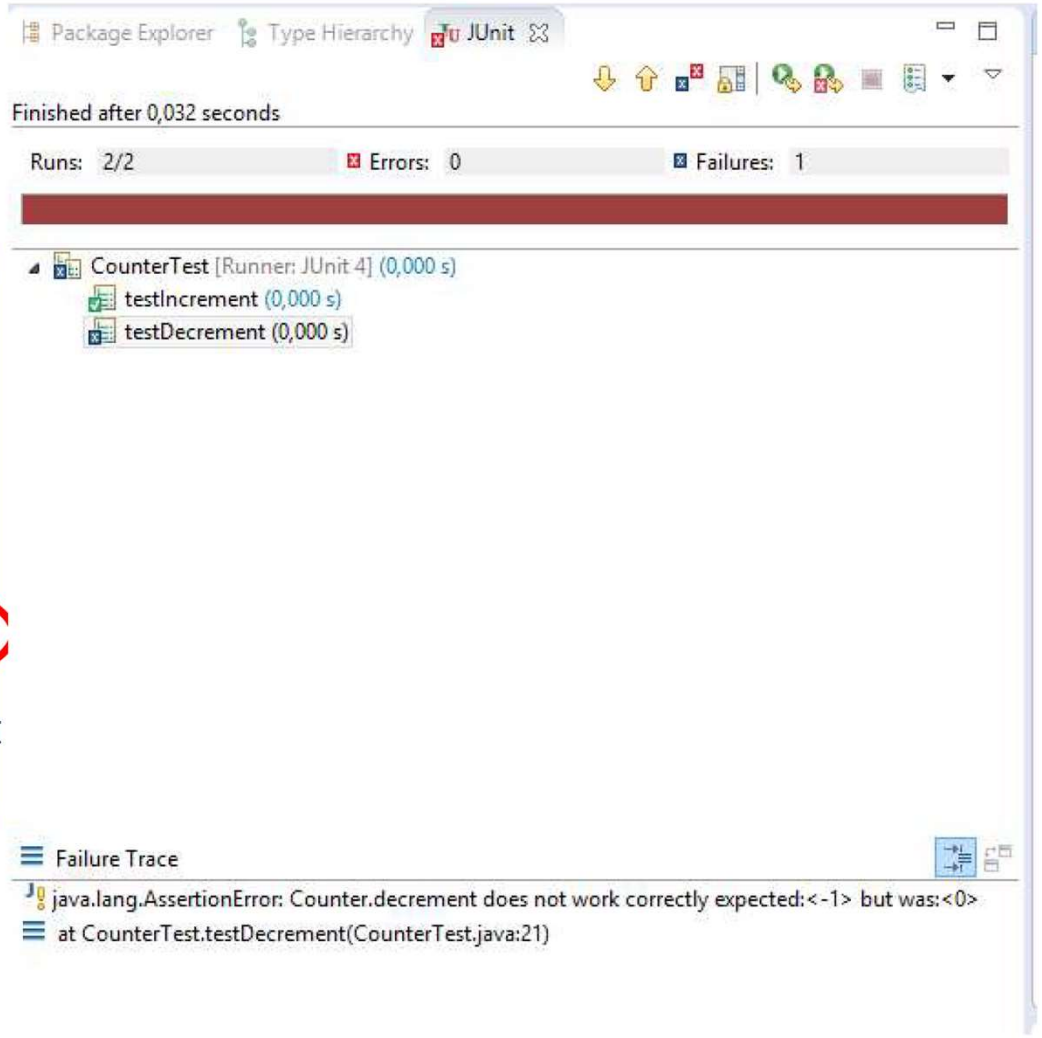
Running the Tests

```
package count;

public class Counter {
    private int counterValue=0;

    public int increment(){
        return ++counterValue;
    }
    public int decrement(){
        return --counterValue;
    }
    public int getCounterValue() {
        return counterValue;
    }
 }
```



Running the Tests - Failing

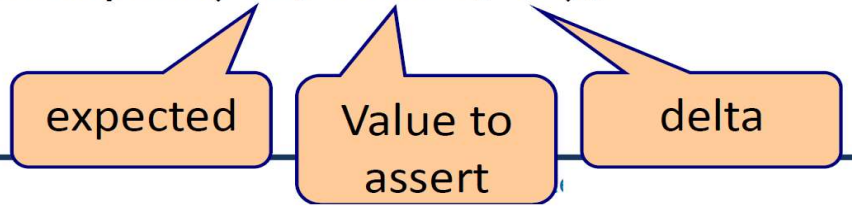
```
package count;

public class Counter {
    private int counterValue=0;

    public int increment(){
       return ++counterValue;
    }
    public int decrement(){
       return counterValue;
    }
    public int getCounterValue() {
       return counterValue;
    }
 }
```



JUnit Test-case

```
public class Calculator
{
    public double add( double number1, double number2 )
    {
        return number1 + number2;
    }
}
```

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

public class CalculatorTest
{
    @Test
    public void add()
    {
        Calculator calculator = new Calculator();
        double result = calculator.add( 10, 50 );
        assertEquals( 60, result, 0 );
    }
}
```



JUnit assert (…) methods

- static void assertTrue(boolean ***test***)
- static void assertTrue(String **message**, boolean ***test***)

- static void assertFalse(boolean ***test***)
- static void assertFalse(String **message**, boolean ***test***)

- assertEquals(Object ***expected***, Object ***actual***)
- assertEquals(String **message**, ***expected***, ***actual***)

- assertSame (Object ***expected***, Object ***actual***)
- assertSame(String **message**, Object ***expected***, Object ***actual***)

- assertNotSame(Object ***expected***, Object ***actual***)
- assertNotSame(String **message**, Object ***expected***, Object ***actual***)

- assertNull(Object ***object***)
- assertNull(String **message**, Object ***object***)

- assertNotNull(Object ***object***)
- assertNotNull(String **message**, Object ***object***)

- fail()
- fail(String **message**)

@Before and @After annotations

```
public class CounterTest {
    private Counter counter;

    @Before
    public void setUp() throws Exception {
       counter = new Counter();
    }

    @After
    public void tearDown() throws Exception {
       counter=null;
    }
    @Test
    public void testConstructor(){
        assertEquals("Counter constructor does not set counter to
                     0",0,counter.getCounterValue());
     }
}
```

This method is called before every testmethod

This method is called after every testmethod

22

@BeforeClass and @AfterClass annotations

```
public class CounterTest {
    private static Counter counter;

    @BeforeClass
    public static void setUpOnce() throws Exception {
       counter = new Counter();
    }

    @AfterClass
    public static void tearDownOnce() throws Exception {
       counter=null;
    }
    @Test
    public void testConstructor(){
        assertEquals("Counter constructor does not set counter to
                      0",0,counter.getCounterValue());
     }
}
```

This method is called once, before the testmethods are called

This method is called once, after the testmethods are called

Timeout tests

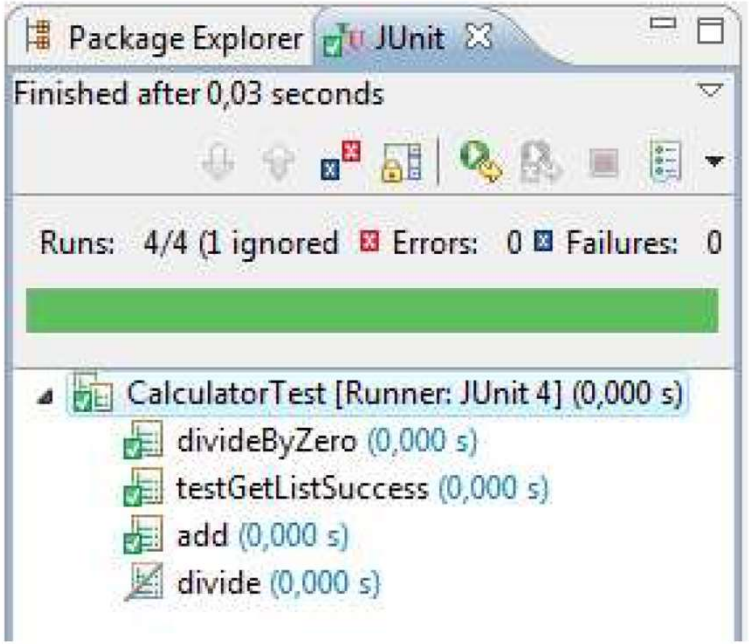
```
@Test(timeout=2000)
public void longOperation() {

}
```

Fail if the test method takes longer than 2000 milliseconds

Skipping a test

```
@Test
@Ignore
public void divide(){
    assertEquals( 5, calculator.divide( 10, 2 ), 0 );
}
```

Skip this test



Test Suite

- A Test Suite is used for organizing and putting a collection of Test classes into a single pack, that can be executed at once
- It provides a facility for organizing related set of tests

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(value=Suite.class)
@SuiteClasses(value={CalculatorTest.class, ParameterizedTest.class})
public class CalculatorTestSuite {

}
```

Suite of 2 test classes

JUnit Test Example - Calculator

```
public class Calculator {
    private double value;

    public Calculator() {
      value =0.0;
    }
    public void add(double number) {
      value = value + number;
    }
    public void subtract (double number) {
      value = value - number;
    }
    public void multiply(double number) {
      value = value * number;
    }
    public void divide (double number) throws DivideByZeroException{
      if (number == 0){
        throw new DivideByZeroException();
      }
      value = value / number;
    }
    public double getValue() {
      return value;
    }
}
```

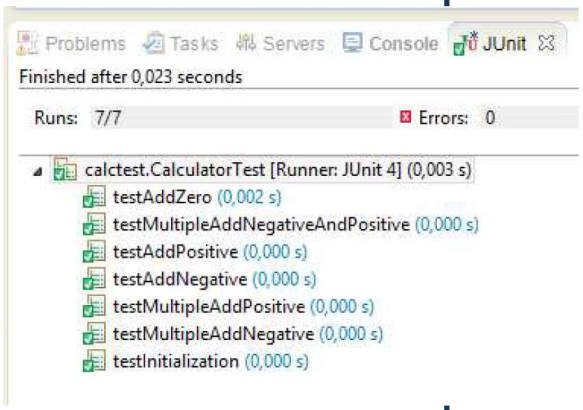
JUnit Test Example - CalculatorTest

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

import calculation.Calculator;

public class CalculatorTest {

  private Calculator calculator;

  @Before
  public void setup(){
    calculator = new Calculator();
  }

  @Test
  public void testInitialization() {
    assertEquals(0.0, calculator.getValue(),0.0000001);
  }

  @Test
  public void testAddZero() {
    calculator.add(0.0);
    assertEquals(0.0, calculator.getValue(),0.0000001);
  }
```

JUnit Test Example - CalculatorTest

```
@Test
public void testAddPositive() {
  calculator.add(23.255);
  assertEquals(23.255, calculator.getValue(),0.0000001);
}
@Test
public void testAddNegative() {
  calculator.add(-23.255);
  assertEquals(-23.255, calculator.getValue(),0.0000001);
}
@Test
public void testMultipleAddPositive() {
  calculator.add(23.255);
  calculator.add(10.255);
  assertEquals(33.510, calculator.getValue(),0.0000001);
}
@Test
public void testMultipleAddNegative() {
  calculator.add(-23.255);
  calculator.add(-10.255);
  assertEquals(-33.510, calculator.getValue(),0.0000001);
}
@Test
public void testMultipleAddNegativeAndPositive() {
  calculator.add(-23.255);
  calculator.add(10.250);
  assertEquals(-13.005, calculator.getValue(),0.0000001);
}
```

Only test methods for add()



End of Part 1

CS489:
Applied Software Development