While folks are joining

Get you laptops ready and login to www.crio.do

Confirm that you are enrolled into QCalc - QCalc

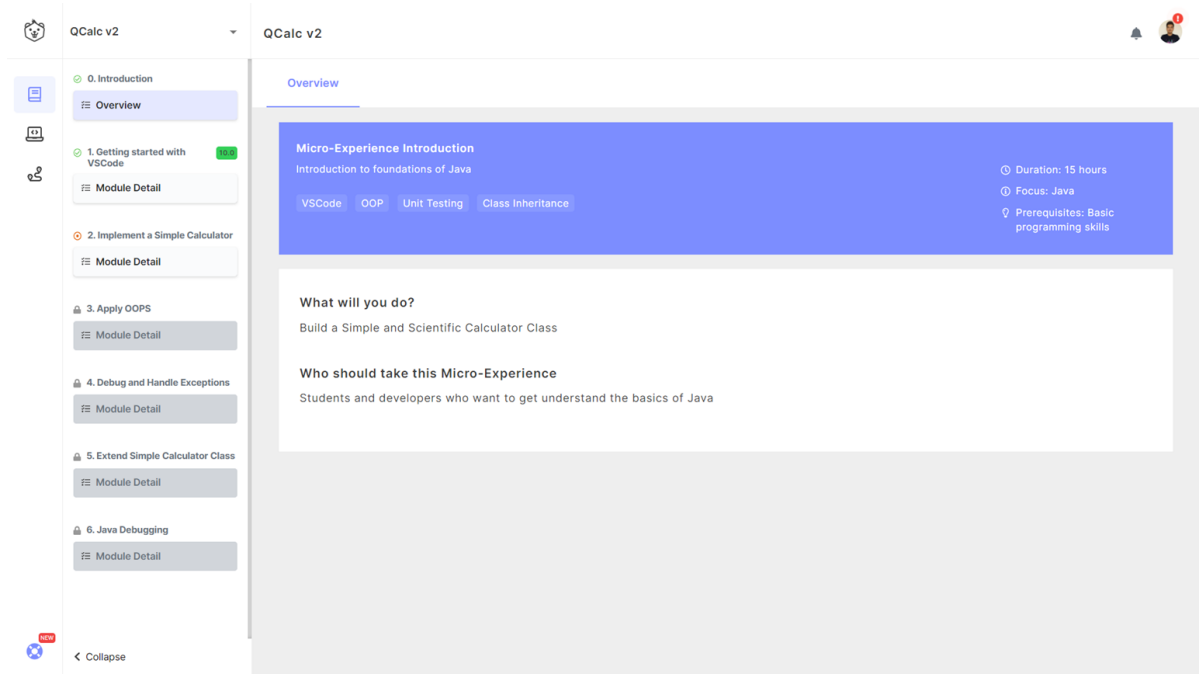
Crio Sprint : JAVA-2

Session 7

Today’s Session Agenda

- Introduction to QCalc Micro-Experience
- Module Introduction
  - Module 1 (in session)
  - Module 2
  - Module 3
- VSCode Basics
- Unit Testing Basics

Introduction to QCalc Micro-Experience



- Calculator project
- VSCode IDE
- Create a Project
- Methods for calculator operations
- Unit Tests
- Method Overloading
- Exception Handling
- Inheritance - Scientific Calculator
- Debugger

Start up your VMs!

QCalc - Module 1: Getting Started with VSCode

- VSCode is a powerful, lightweight code editor with support for multiple languages (Java, Python, C++, etc) and support for multiple platforms (Windows, Mac, Web browser).

- In this module:
  - Get started with VSCode
  - Install Extensions
  - Generate a new Java Project
  - Run the application and print “Hello World!”

Project creation error? Visit -> Project creation error

Complete Module 1 of QCalc

- Let’s do this together

VSCode Tips and Tricks

- Clone a repo which has multiple files
- Explore how to use VSCode shortcuts, navigate files, search strings and more

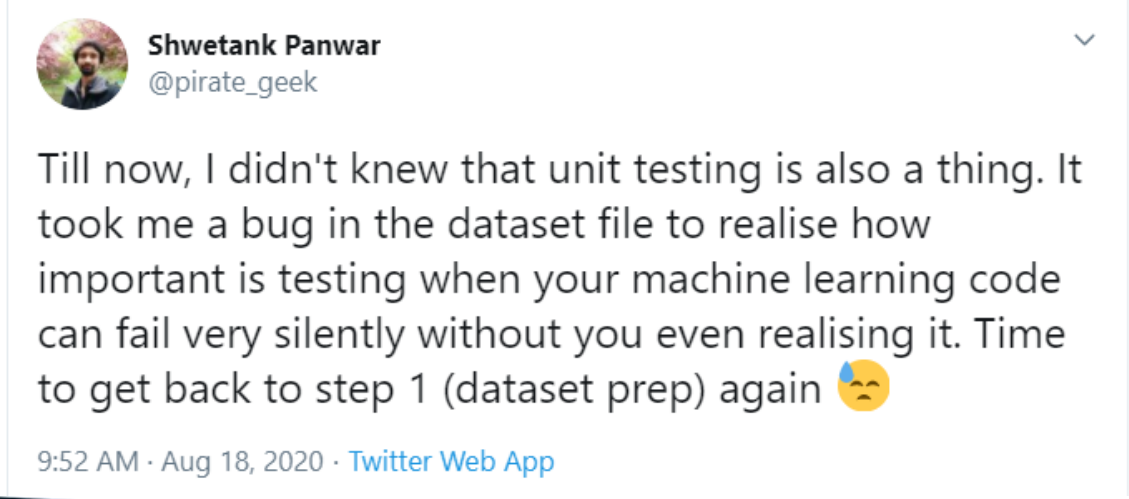
Don’t worry if you don’t understand the full project. You will be creating such bigger projects from scratch going forward.

QCalc - Module 2: Implement a Simple Calculator

- In this module:
  - Implement methods in the Calculator Class
  - Write Unit Tests for the methods
  - Execute and test

What is Unit Testing?

- A procedure to **validate individual units of Source Code**
- Validating each individual piece **reduces errors** when integrating the pieces together later
- **Junit** is a unit testing **framework for Java**
- Allows you to write unit tests in Java using a simple interface
- **Automated testing** enables running and re-running tests very easily and quickly.



Random Testing Memes

```
Roses are Red,
Violets are Blue

Unexpected '{'
on line 32.
```

Create a New Demo Java Project

- Create a new “junitdemo” Project using the Spring Initializer Extension (cd ~/workspace)
- Create a new file `Rectangle.java` under src > com > crio > junitdemo folder.
- Create a new file `RectangleTest.java` under test > com > crio > junitdemo folder.
- In this file, we will be writing and executing our unit tests.

Junit Annotations Basics

@Test

- This annotation denotes that a method is a test method.
- Note this annotation does not take any attributes.

```
import org.junit.jupiter.api.Test;
import static
org.junit.jupiter.api.Assertions.assertEquals;

class RectangleTest {

   @Test
   void helloJUnit5() {
      assertEquals(10, 5+5);
   }
}
```

Junit Annotations Basics

@DisplayName

- Test classes and test methods can declare custom display names that will be displayed by test runners and test reports.

```
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInfo;

@DisplayName("DisplayName Demo")
class JUnit5Test {
   @Test
   @DisplayName("Custom test name")
   void testWithDisplayName() {
   }

   @Test
   @DisplayName("Print test name")
   void printDisplayName(TestInfo testInfo) {
      System.out.println(testInfo.getDisplayName());
   }
}
```

Junit Annotations Basics

@BeforeEach

The @BeforeEach annotation denotes that the annotated method should be executed before each test method.

```
import org.junit.jupiter.api.*;

class JUnit5Test {
   @BeforeEach
   void init() {
      System.out.println("Executing this before each testcase");
   }

   @Test
   void firstTest() {
      System.out.println(1);
   }

   @Test
   void secondTest() {
      System.out.println(2);
   }
}
```

Junit Annotation

Other Annotations

- @ParameterizedTest
- @RepeatedTest
- @AfterEach
- @BeforeAll
- @AfterAll
- @Tag
- @Disabled

Get to know about them in detail here [JUnit 5 Annotations With Examples (devqa.io)]

JUnit will be covered in detail in next Sprint.

Junit Assertions

There are variety of Assertions provided by Junit 5 framework. Most commonly used are:

- *assertEquals*
- *assertTrue* and *assertFalse*
- *assertNull* and *assertNotNull*
- *assertThrows (used with exceptions)*

Check out this link for more advanced Assertions [Assertions in JUnit 4 and JUnit 5 | Baeldung]

Let’s test Rectangle Class

```
public class Rectangle {
    private final double width, height; //sides
    public Rectangle() {
        this(1,1);
    }
    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }
    public double calculateArea() {
        return width * height;
    }
    public boolean isSquare(){
        if(width == height){  return true; }
        return false;
    }
}
```

assertEquals()

- In **assertEquals**() method, we check that the **two objects are equals or not.**

```
@Test
public void testCalculateArea() {
   Rectangle r = new Rectangle(4,8);
   assertEquals(32.0,r.calculateArea());
}
```

assertTrue() and assertFalse()

- **assertTrue()** method asserts that a condition is **True.**

- **assertFalse()** method asserts that a condition is **False.**

```
@Test
public void testIsSquare() {
   Rectangle r = new Rectangle(2,2);
   assertTrue(r.isSquare());
}
@Test
public void testIsSquare() {
   Rectangle r = new Rectangle(2,3);
   assertFalse(r.isSquare());
}
```

assertNull() and assertNotNull()

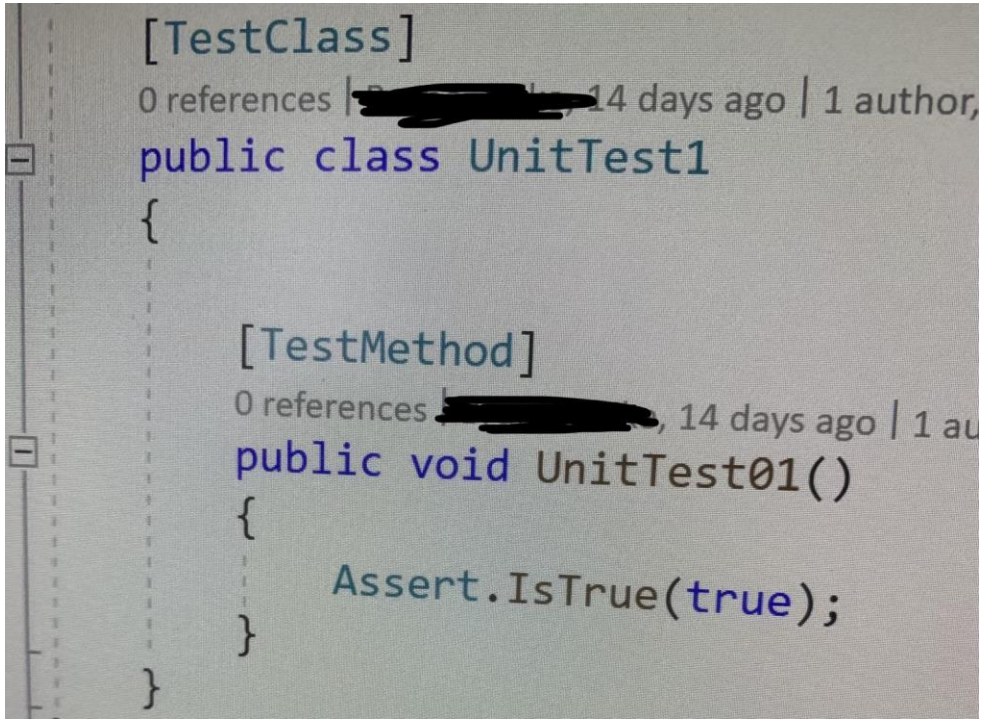
- **assertNull()** method asserts that an object is null.

```
@Test
public void whenAssertingNotNull_thenTrue() {
    Rectangle r = new Rectangle();
    assertNotNull(r);
}
```

- **assertNotNull()** method asserts that an object isn't null.

```
@Test
public void whenAssertingNull_thenTrue() {
    Rectangle r = null;
    assertNull(r);
}
```

Don’t do this


[TestClass]
0 references | [redacted] 14 days ago | 1 author,
public class UnitTest1
{
[TestMethod]
0 references [redacted], 14 days ago | 1 au
public void UnitTest01()
{
Assert.IsTrue(true);
}
}

Activity: Think and Test

- Training on Thinking & Testing : True or False | Codewars
- Training on Thinking & Testing: A and B? | Codewars

QCalc - Module 3: Apply OOPS

- In this module:
  - Implement additional functionality using Method Overloading
  - Write Unit Tests for the new methods as well
  - Execute and Test

Take home exercises for the session

- You will have to complete the below modules of QCalc Micro-Experience**:**
  - Getting Started with VSCode
  - Implement a Simple Calculator
  - Apply OOPS
- Try to finish this before the next session.

Feedback

Thank you for joining in today.

We’d love to hear your thoughts and feedback - Feedback for JAVA-2 Session

Thank you