While folks are joining

- Get you laptops ready and login to www.crio.do
- Confirm that you are enrolled into QCalc - QCalc
- Open QCalc ME and start your workspace.
- Open Terminal and type

  cd ~/workspace
- Clone the repo in ~/workspace directory
  - git clone git@gitlab.crio.do:bdt-sprint-codes/java-ii/java-ii-session-activities.git
- Open session-5 folder.
- Setup Video for Reference

While folks are joining

- Get you laptops ready and login to www.crio.do
- Confirm that you are enrolled into QCalc - QCalc
- Open QCalc ME and start your workspace.
- Open Terminal and type

  cd ~/workspace
- Setup Video for Reference

Crio Sprint: JAVA-112

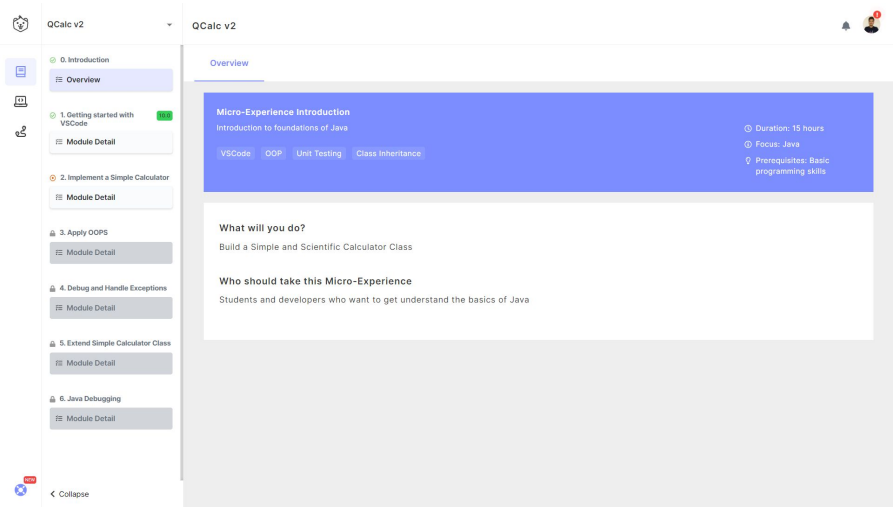
Session 7 - QCalc and JUnit

Today’s Session Agenda

- Introduction to QCalc Micro-Experience
- QCalc: Module 1 Introduction (In-Session)
- Unit Testing using JUnit
- QCalc: Module 2 Introduction

Introduction to Micro-Experiences

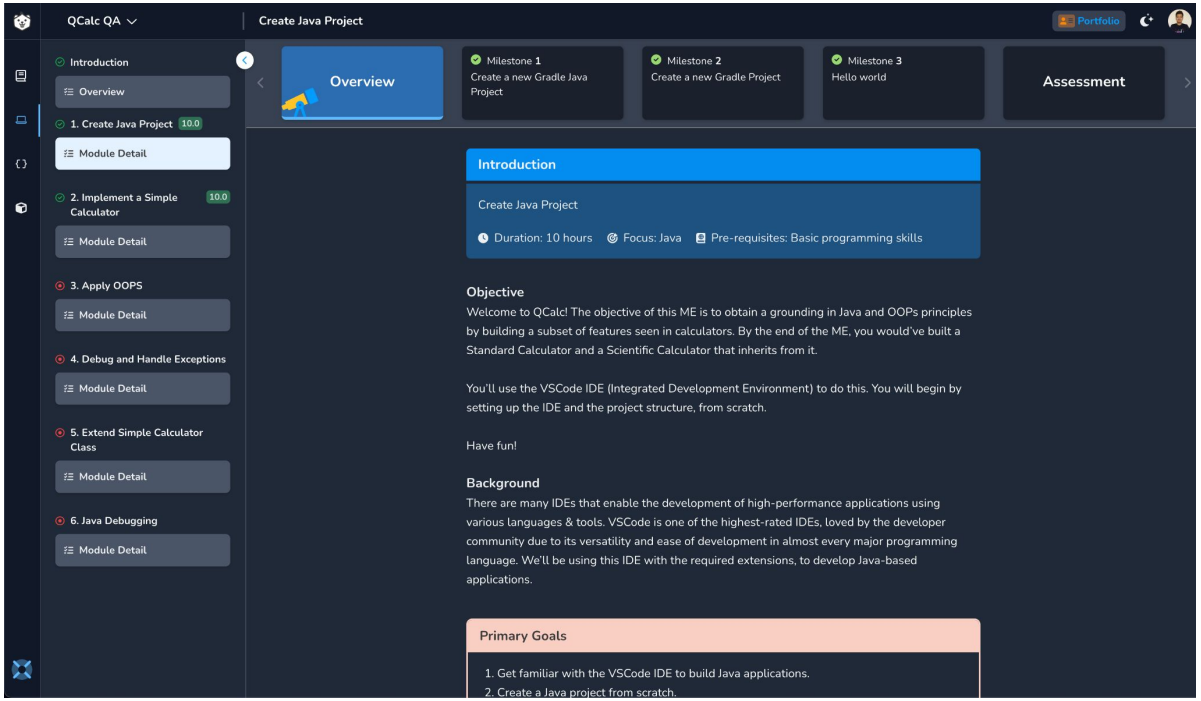
- Micro Experiences also referred to as MEs are guided projects where you will be building a product with similarities to real life products (eg: QEats - Swiggy like, QKart - Flipkart like)
- Through MEs, you will get to put the concepts and techniques learned from Live sessions and Bytes into practice by implementing different functionalities of the project, debugging issues, improving the product etc
- Each of these MEs are further divided into Modules. Each of these modules can take around 4-6 hours to complete on average
- A module will have a set of objectives to complete. Some starter code will be provided to help you with completing the MEs



Note:

- The time required given here is a rough estimate and can vary based on each ME and module. Our CSM team will be sharing with you the average of each module
- Make sure you understood topics covered in all Live concept sessions and completed all the session take-home assignments to grab the required prerequisite knowledge for the MEs. This will ensure a smoother ME experience.

Introduction to QCalc Micro-Experience



In this ME (covered across 3 sessions):

- Calculator project
  - Create a Project
  - Methods for calculator operations
  - Method Overloading
  - Inheritance - Scientific Calculator
  - Unit Tests
- VSCode IDE
- JUnit and related annotations
- Exception Handling
- IDE Debugger - Breakpoints, Watches, Step In, Step Out, Stacktrace

Start up your VMs!

Introduction to QCalc Micro-Experience

In this ME (covered across 3 sessions):

- Calculator project
  - Create a Project
  - Methods for calculator operations
  - Method Overloading
  - Inheritance - Scientific Calculator
  - Unit Tests
- VSCode IDE
- JUnit and related annotations
- Exception Handling
- IDE Debugger - Breakpoints, Watches, Step In, Step Out, Stacktrace

Start up your VMs!

Let’s think about a question…

- What happens if we manually create a Java project?
  - It will become time consuming to manage and compile so many files
  - What if we want to work with external dependencies?
    - We will have to manually add each of the dependencies we require
- Instead, we can use a build tool to create a project
- Build tools speed up development time by solving some of the most crucial issues like the ones above
- We will be learning about Build Tools in depth in the upcoming sprints
- For now, all you need to know is that build tools speed up project creation and development time
- Famous build tools in Java: Maven, Gradle, Ant
- We will be using Gradle to create and work with projects at Crio

Analogy to understand Build Tools

- Build Tools are Like an Operating System for Your Computer
- Without an Operating System (No Build Tools):
  - Your computer lacks an organized system to manage resources and run programs
  - You have to manually handle memory, file management, and device communication
  - It’s easy to encounter errors, conflicts, and inefficiencies without proper organization
- With an Operating System (Using Build Tools):
  - The operating system provides a structured environment for managing resources and executing tasks
  - It automates processes like memory allocation, file handling, and device drivers
  - It ensures smooth operation and efficient utilization of hardware resources
- Like using an OS, build tools help you to focus on the most important task at hand
- Gradle is just one of the many build tools, like different Operating Systems
- Build tools will be covered in more detail later

QCalc - Module 1: Create Java Project

- In this module:
  - Generate a new **Gradle** Java Project
  - Run the application and print “Hello World!”

Complete Module 1 of QCalc

- Let’s do this together

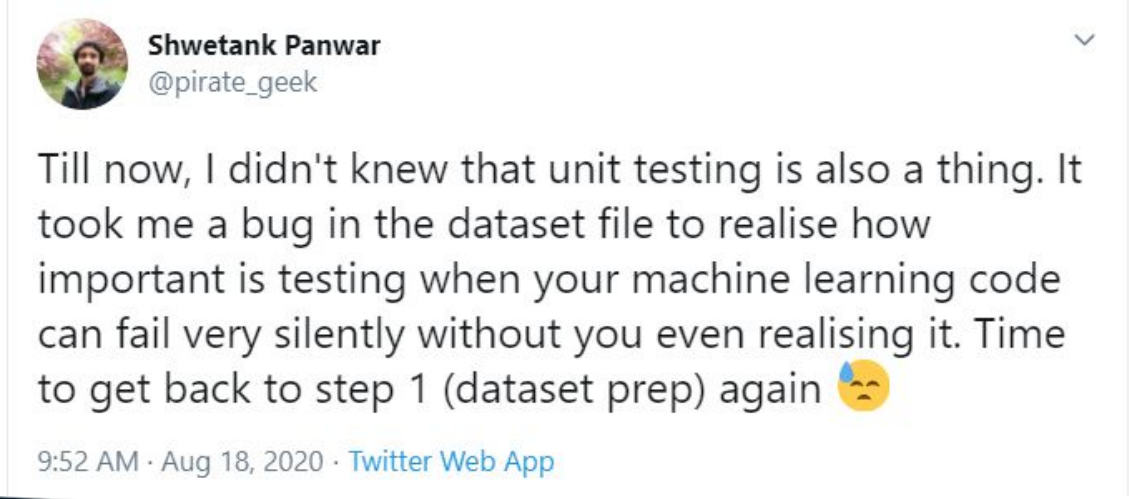
VSCode Tips and Tricks (watch this video after the session)

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

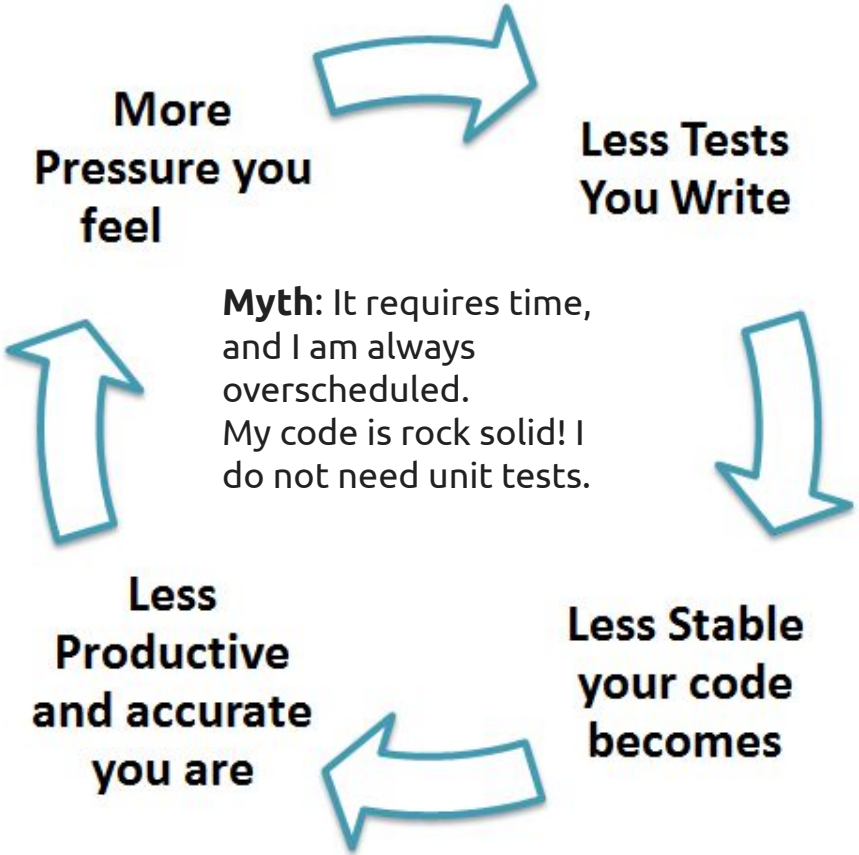
Don’t worry if you don’t understand the full project. You will be creating such bigger projects from scratch going forward.

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.



Why Unit Testing?



Truth is Unit testing increase the speed of development.

Create a New Demo Java Project

- Create a new “junitdemo” Project using Gradle (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.

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) {
   if(width <=0 || height <=0){
      throw new ArithmeticException("Width or Height Cannot be Negative or Zero");
   }
      this.width = width;
      this.height = height;
   }
   public double calculateArea() {
      return width * height;
   }
   public boolean isSquare(){
      if(width == height){  return true; }
      return false;
   }
}
```

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 Annotations

Other Annotations

- @AfterEach
- @BeforeAll
- @AfterAll
- @RepeatedTest
- @Disabled

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

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]

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.**

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

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

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

assertNull() and assertNotNull()

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

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

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

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

assertThrows()

- **assertThrows()** method asserts a method which **throws an exception.**
- If **no exception is thrown** from the executable block then assertThrows() will **FAIL.**
- If an **exception of a different type** is thrown, **assertThrows()** will **FAIL.**

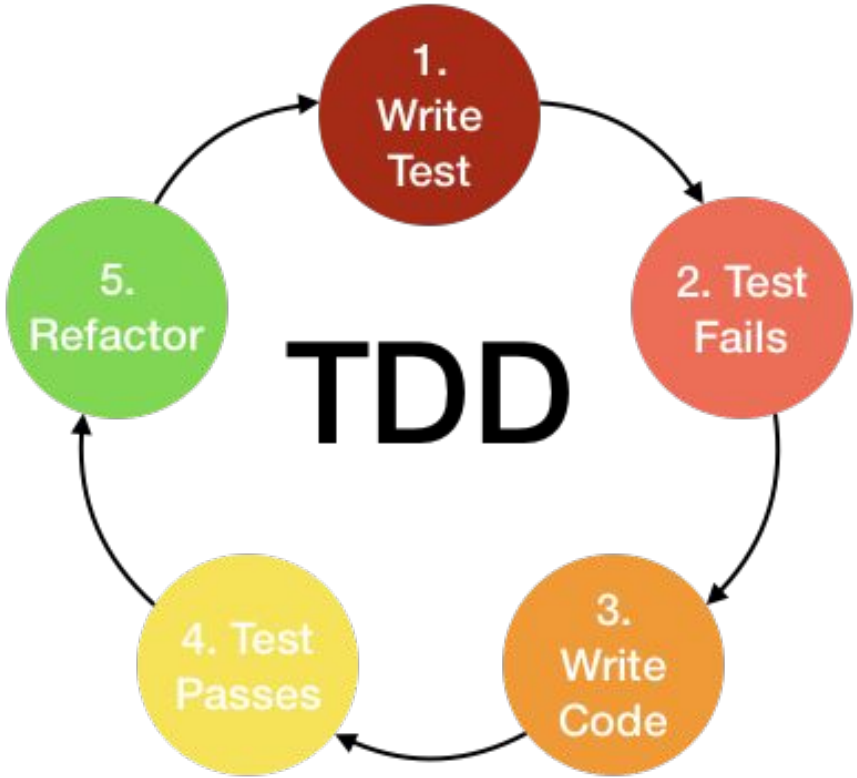
```
import org.junit.jupiter.api.function.Executable;

@Test
void testRectangleException(){
//First argument - specifies the expected exception.
        //Here it expects that code block will throw
        ArithmeticException
//Second argument - is used to pass an executable code
block
assertThrows(ArithmeticException.class,new Executable(){
    @Override
    public void execute() throws Throwable{
      new Rectangle(0,-3);
    }
  });
}
```

Activity #1 - Filtering even numbers

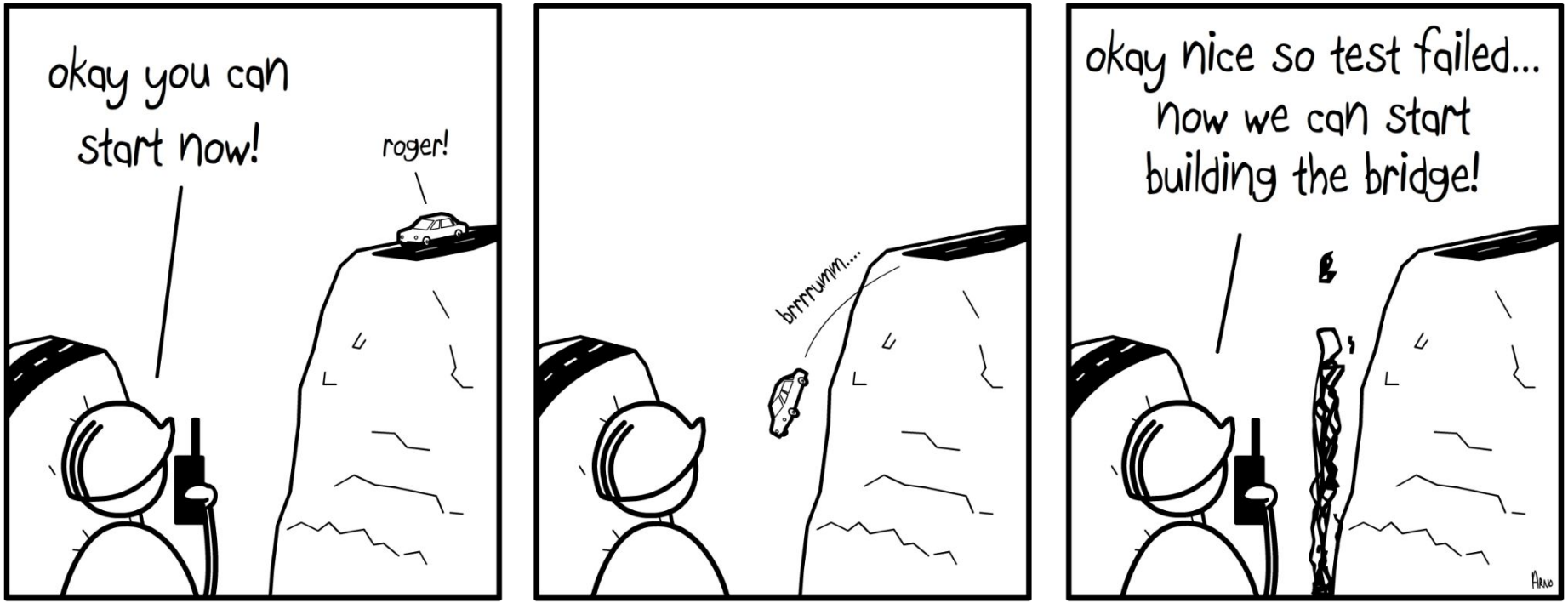
- In this activity, we will understand how we can make use of Unit Tests to correct our code implementation.
- Clone this repository: https://gitlab.crio.do/public_content/bdt/session-activities/junit
- Open the folder: FilterEvenNumbers
- Understand the bug and debug it

5 minute break

Test Driven Development



Test Driven Development

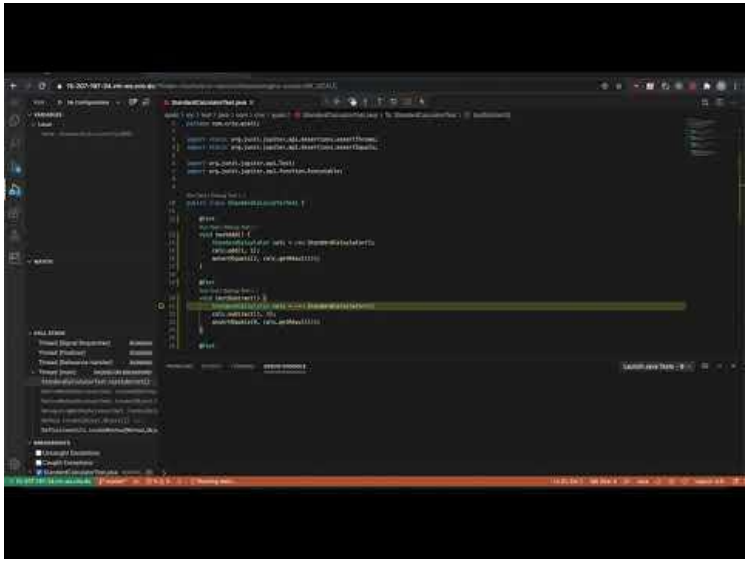


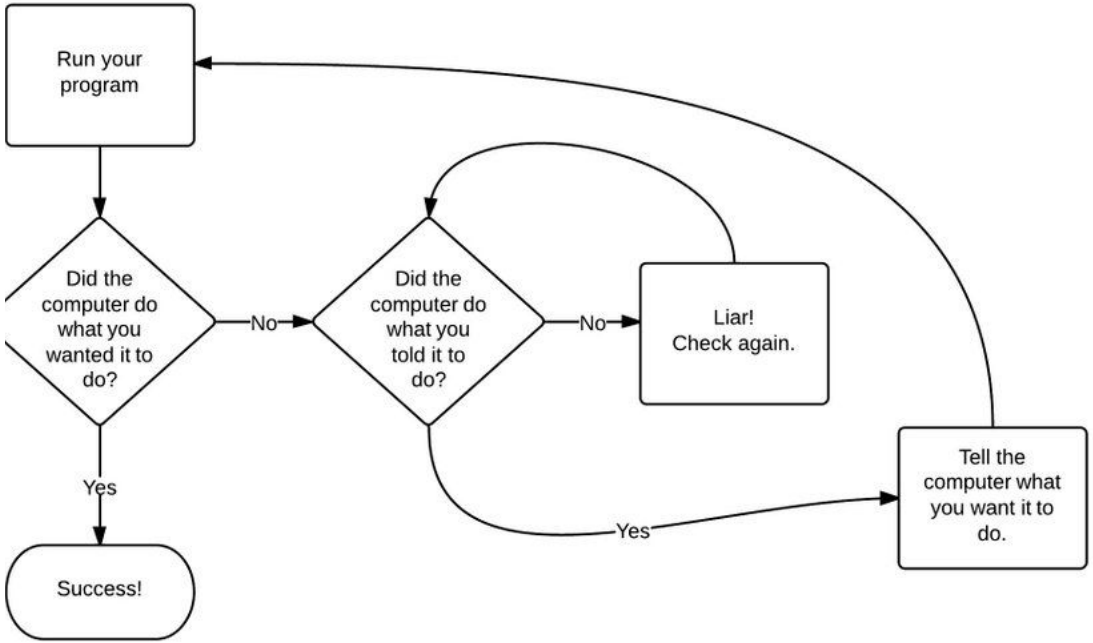
Activity #2 - Array Plus Array

- In this activity, we will apply TDD approach to complete the implementation provided the unit tests.
- From the previously cloned repository, open the folder: SumOfArrays
- Implement the totalSum method using TDD

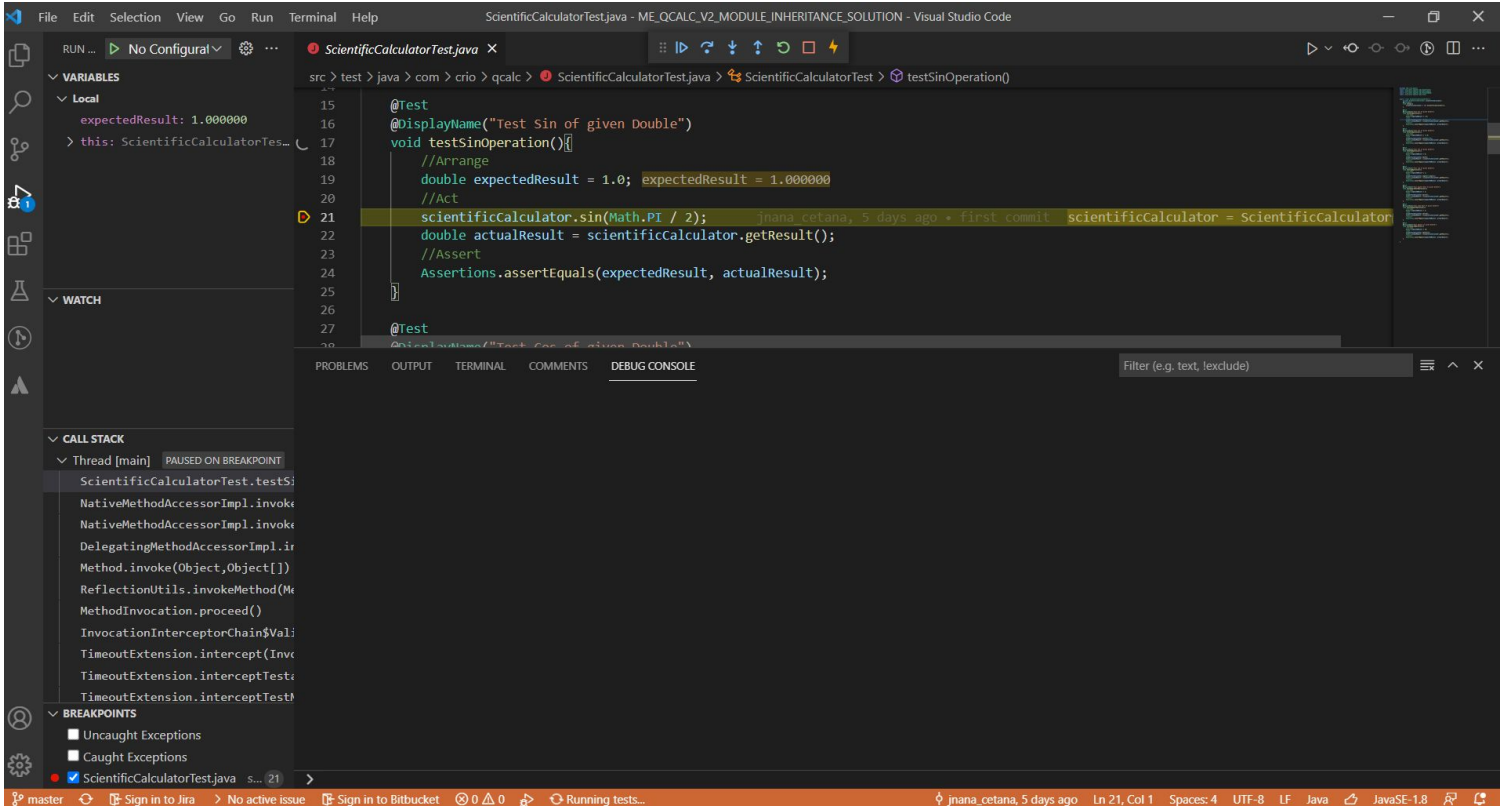
VSCode Debugger (Watch After Session)

- Learn how to debug your code with the help of Unit Tests using VSCode Debugger





VSCode Debugger Interface



Debugging Toolbar

Continue (F5) - This command resumes the program and continues to run it normally, without pausing unless it hits another breakpoint.

Step Over (F11) - The Step Over command takes a single step. It executes the currently highlighted line, and then pauses again.

Step Into (F10) - Step Into goes *into* that function and pauses on the first line inside.

Step Out (Shift + F11) - Step Out command executes all the code in the current function, and then pauses at the next statement (if there is one).

Stop (Shift + F5) - Stop Debugging

Restart (Ctrl + Shift + F5) - Restart Debugging

QCalc - Module 2: Implement a Simple Calculator

- In this module:
  - Implement methods in the Calculator Class
    - Addition
    - Subtraction
    - Multiplication
    - Division
  - Apply Encapsulation by making Use of Getter and Setters.
  - Write Unit Tests for the above mentioned operations.
  - Execute the unit tests written for the above method to verify correctness of the implementation.
  - Submit the code for assessment.

Questions

1. What is JVM (Java Virtual Machine) and what role does it play in Java programming?
2. What is the purpose of the ClassPath in Java? Explain its types.
3. What is JRE (Java Runtime Environment)? How does it differ from JDK?
4. What is Gradle and how does it relate to Java development?
5. What is the purpose of the Java classloader and how does it work?

Session Revision Quiz

Quiz Link

Solve this quiz to access your understanding of session’s topics clearly

Take home exercises for the session

- Watch VSCode Tips & Tricks Video
- You will have to complete the below modules of QCalc Micro-Experience:
  - Module 1: Generate a Java project
  - Module 2: Implement a Simple Calculator

Further Reading

- Tutorial on how Java classpath works | Javarevisited (medium.com)
- Runtime Classpath vs Compile-Time Classpath - DZone Java
- Java JAR tutorial and guide to create them | Javarevisited (medium.com)
- How to Set Classpath for Java on Windows and Linux? Steps and Example (javarevisited.blogspot.com)

References

- Run JUnit Test Cases From the Command Line | Baeldung

Thank you