DEMO NOTES FOR CREATING JAVA PROJECT IN VSCode

How to Install JDK

1. Browse to the AdoptOpenJDK website.
2. Choose OpenJDK 16 (latest) as the version.
3. Choose OpenJ9 as the JVM.
4. Click the Latest release download button to download the package.

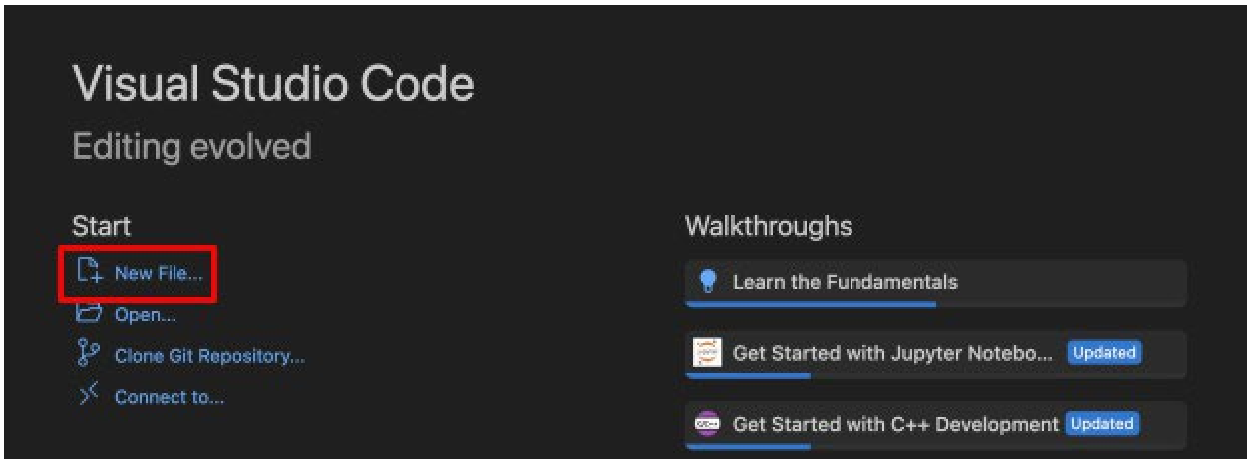
How to Install and setup VSCode and Select Workspace

1. Browse to https://code.visualstudio.com/download
2. Select Download based on your device type
3. Follow the prompted steps and continue
4. Once installed, open VS Code and install the **Extension Pack for Java** from the Extensions Marketplace. This pack includes the essential tools for Java development in VS Code.

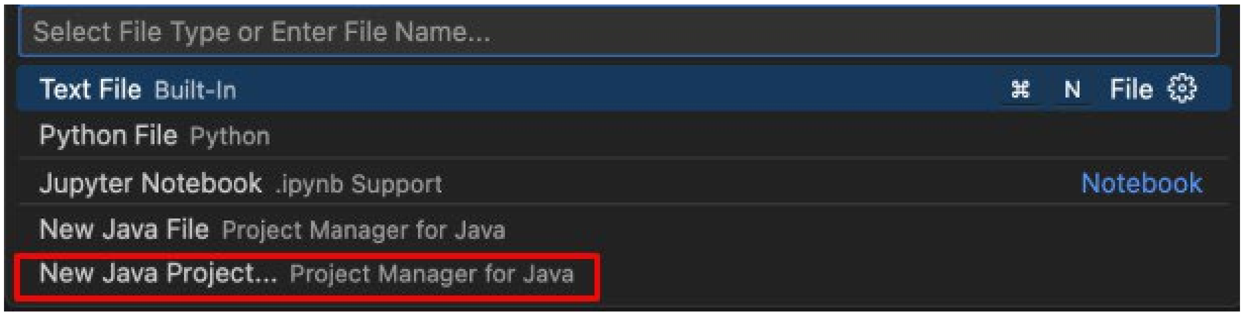
Reference: https://code.visualstudio.com/docs/java/java-tutorial

You can also download the coding pack and the extension pack instead from the above link.

Creating a Project



1. Click **New File**
2. Search Java Project in the search bar and click **New Java Project**


Select File Type or Enter File Name...
Text File Built-In ⌘ N File
Python File Python
Jupyter Notebook .ipynb Support Notebook
New Java File Project Manager for Java
New Java Project... Project Manager for Java

3. Select **No Build Tools** and Name the project **myFirstProject**
4. Open the Command Palette (Ctrl + Shift + P or Cmd + Shift + P)
5. Search for Java: Configure Java Runtime and select it. Ensure that the JDK downloaded before is the one selected
6. In the Src file, create a new folder labeled **cs3300**
7. Within the folder, create a **PrintCheck.java** class
8. Within the class, create the main method and write an example print statement.
9. In the top right corner, you can click the triangle button to run the current file.
    a. You can also click the run button seen in the class directly.

```java
package CS3300;

public class PrintCheck {
    public static void main(String[] args) throws Exception {
        System.out.println("Hello World!");
    }
}
```

    b. The output will appear in the terminal at the bottom of the application.

Perspectives/Views

1. The Java Project View on the left pane will show you your project hierarchy. This allows you to view how the classes are organized within your project.
2. Source Code Editor in the middle. This is where most of your time will be spent within the IDE and where the majority of your work will take place.

Configurations

1. Add greeting.java

```java
package CS3300;

public class greeting {
    public static void main(String[] args) {
        if (args.length > 0) {
            System.out.println("Hello, " + args[0] + "!");
        } else {
            System.out.println("Hello, World!");
        }
    }
}
```

This program prints a personalized greeting if a name is provided as a command line argument, otherwise, it defaults to "Hello, World!". So, lets do a **Customized Launch Configuration**.

2. Step-by-Step Setup with `launch.json` in VSCode

- Go to the Run and Debug view in VSCode (using the sidebar icon or `Ctrl+Shift+D`).
- Click on "create a launch.json file" if it's not already created. Select "Java" when prompted for the environment.
- You'll see a `launch.json` file open with some default configurations. Modify or add a new configuration that includes command line arguments. I added args: Stacy

```
        {
            "type": "java",
            "name": "greeting",
            "request": "launch",
            "mainClass": "CS3300.greeting",
            "projectName": "myFirstProject_899b3e29",
            "args": "Stacy"
        }
```

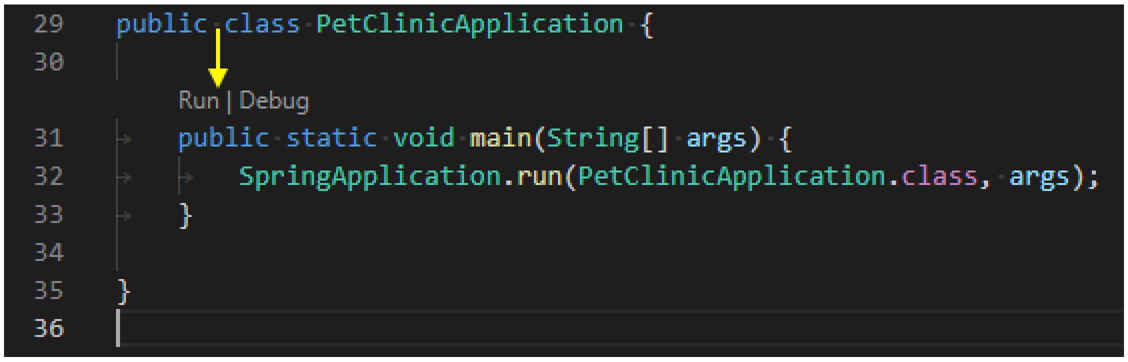
3. Run the Configuration**:**
   - Select the new configuration ("Launch Greeting with Arguments") from the dropdown at the top of the Run and Debug sidebar.
   - Press the green play button or `F5` to start debugging.
   - The debugger will start, and you should see "Hello, Stacy!" printed in the Debug Console.
4. Demonstrate Debug Features**:**
   - Add a breakpoint inside the `if` statement by clicking in the margin next to the line number in `Greeting.java`.
   - Run the debug configuration again. This time, the debugger will pause at the breakpoint, allowing you to inspect variables, step through code, and observe the behavior of the program.
5. Benefits of This Setup

- **Parameter Testing**: Easily test how your application handles different inputs without manually entering them each time.
- **Reproducibility**: Other developers can pull your code and have the exact same setup ready for debugging.
- **Efficiency**: Streamlines debugging sessions by pre-configuring complex command line arguments.

Debugging

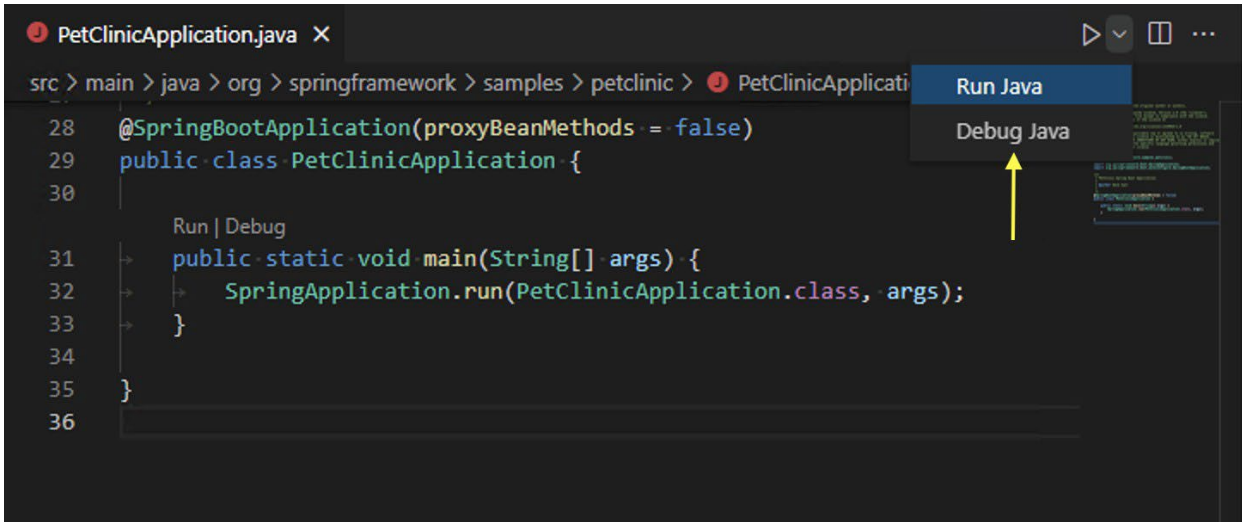
Reference: https://code.visualstudio.com/docs/java/java-debugging

1. Install the Debugger for Java Extension
2. Click the Debug option in the main method



```
public class PetClinicApplication {

    public static void main(String[] args) {
        SpringApplication.run(PetClinicApplication.class, args);
    }

}
```

You can also select the button in the top right corner


PetClinicApplication.java
src > main > java > org > springframework > samples > petclinic > PetClinicApplicati
Run Java
Debug Java
Run | Debug

```
@SpringBootApplication(proxyBeanMethods = false)
public class PetClinicApplication {

    public static void main(String[] args) {
        SpringApplication.run(PetClinicApplication.class, args);
    }

}
```

3. To debug your Java application, you can place breakpoints by clicking in the left margin next to the line numbers.
   a. **Breakpoint - Conditional breakpoint**: With the help of expression evaluation, the debugger also supports conditional breakpoint. You can set your breakpoint to break when the expression evaluates to true.
   b. **Breakpoint - Data breakpoint**: You can have the debugger break when a variable changes its value. Note that the data breakpoint can only be set inside a debug session. This means you need to launch your application and break on a regular breakpoint first. You can then pick a field in the VARIABLES view and set a data breakpoint.
   c. **Breakpoint - Logpoints**: Logpoints is also supported by Java Debugger. Logpoints allow you to send output to Debug Console without editing code. They're different from breakpoints because they don't stop the execution flow of your application.

Step 1: Setting Breakpoints

1. **Set a Regular Breakpoint**:
   - Click in the left margin next to the line `System.out.println("Hello, " + args[0] + "!");` in `greeting.java`. This will pause execution whenever this line is about to execute.
2. **Set a Conditional Breakpoint**:
   - Right-click on the breakpoint you just set and choose "Edit Breakpoint...".
   - Add a condition like `args.length == 0`. This means the breakpoint will only trigger if no arguments are provided to the program, which won't be the case here since "Stacy" is provided, but it's good for testing.

Step 2: Adding a Logpoint

1. **Set a Logpoint**:
   - Right-click in the left margin on the line inside the `else` block: `System.out.println("Hello, World!");`.
   - Choose "Add Logpoint...".
   - Type `"No arguments provided, defaulting to World"`. This message will appear in the debug console when this line of code is executed without stopping the program.

Step 3: Observing the Debugger

- As you run the debugger with these settings, observe the behavior:
    - The debugger should hit the regular breakpoint and pause execution.
    - You can inspect variables like `args` in the left-hand "Variables" pane.
    - Check the "Debug Console" for any output from logpoints or the program itself.

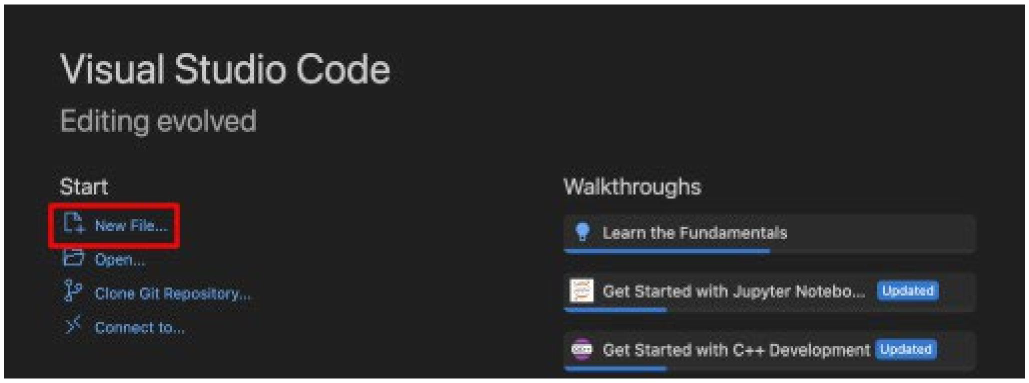
Step 4: Using Data Breakpoints

- **Add a Data Breakpoint** (if supported by the Java extension and JDK):
    - While the debugger is paused, hover over a variable you want to monitor, such as `args`.
    - Right-click and choose "Add Data Breakpoint" if available. This will pause execution if the value of the variable changes.
    - Note: Data breakpoints are highly specific and might not be supported for all types of data in Java.

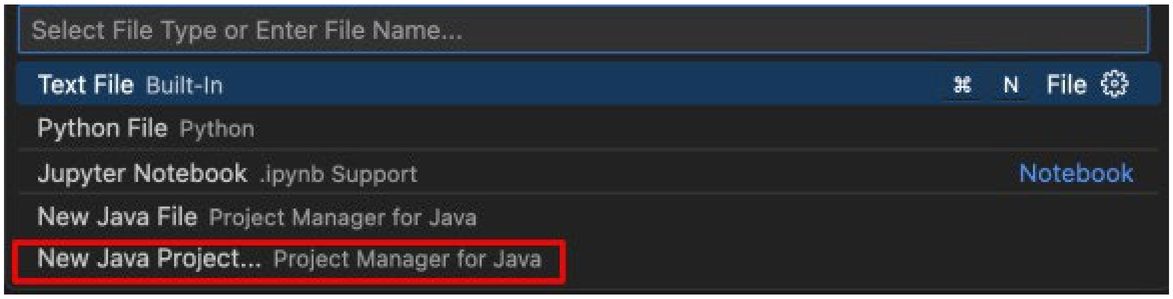
DEMO NOTES FOR CREATING A MAVEN PROJECT IN VSCode

VS Code does not come with built in support for Maven (unlike IntelliJ IDEA). So you need to download and install Maven.
Creating maven project is simple in VSCode as it is an out of the box feature

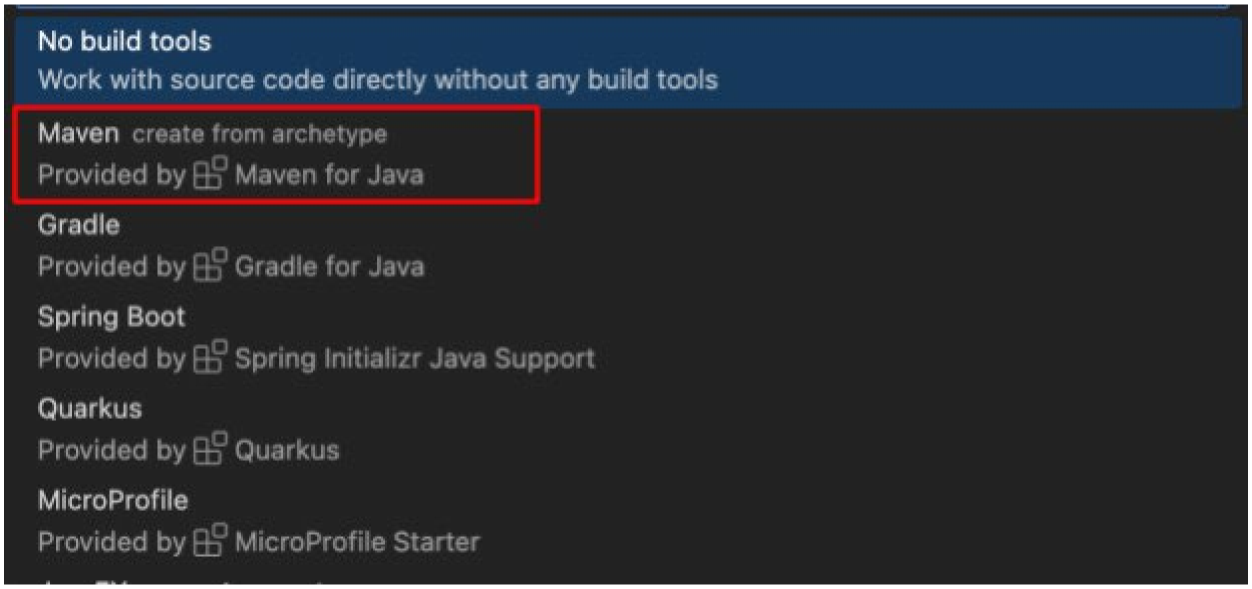
1. Similar to the steps seen in the creating a java project section, click the new file button



2. Next, select the New Java Project option

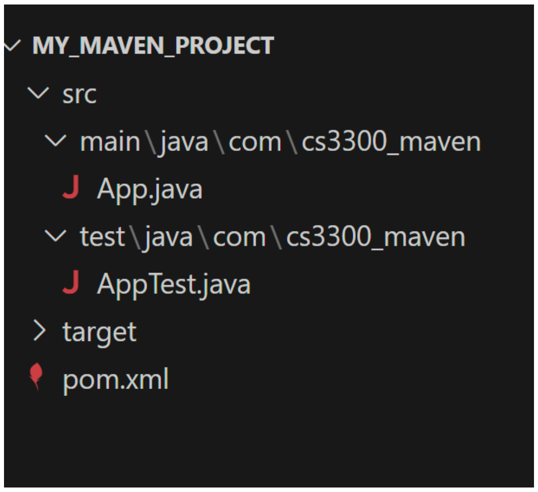

Select File Type or Enter File Name...
Text File Built-In ⌘ N File
Python File Python
Jupyter Notebook .ipynb Support Notebook
New Java File Project Manager for Java
New Java Project... Project Manager for Java

3. Now, select the Maven option


No build tools
Work with source code directly without any build tools
Maven create from archetype
Provided by Maven for Java
Gradle
Provided by Gradle for Java
Spring Boot
Provided by Spring Initializr Java Support
Quarkus
Provided by Quarkus
MicroProfile
Provided by MicroProfile Starter

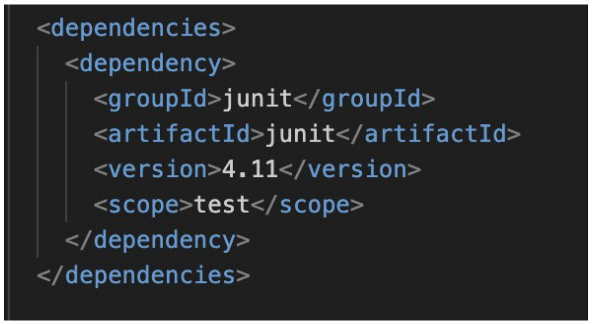
4. Select the quick start option, and then, select the version wanted.
5. Choose groupid and artifactid
    - Groupid: uniquely identifies your project across all projects. Kind of like package. *com.cs3300_maven*
    - artifactId is the name of the project. Let's say *my_maven_project*

6. Select enter or continue with all default options
7. Within the application, you will see a main and test folder. Explore both of these to get an idea of the maven project.



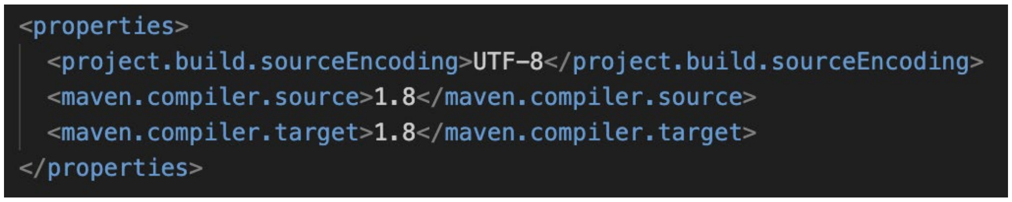
```
MY_MAVEN_PROJECT
  src
    main\java\com\cs3300_maven
      App.java
    test\java\com\cs3300_maven
      AppTest.java
  target
  pom.xml
```

8. You will also see a pom.xml file. Here, you can declare any dependencies needed. For this example, you can take note of the junit dependency included.



```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

9. Run the App.java file to test out the build. If you see an error related to the java version, change the version in the pom.xml file to the supported version seen in the error message.



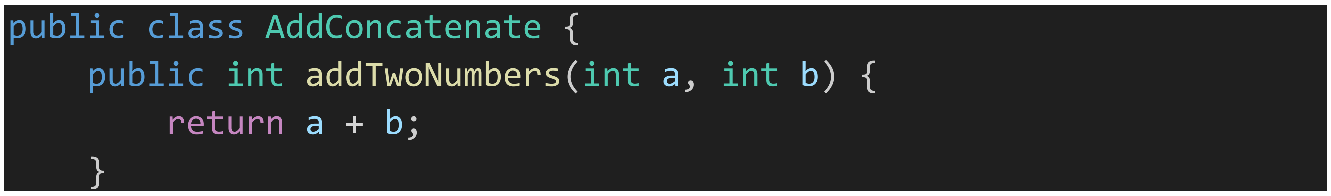
```
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
</properties>
```

Additionally, in the search bar at the top of the application, you can first hit > and then search for Java: Configure Java Runtime to edit your settings.

10. You may also need to *mvn clean install* first

<u>DEMO NOTES FOR CREATING JUNIT TESTS IN VSCode using Maven</u>

1. Create a class name AddConcatenate.java, and within this, create a method where you add two integers and return the integer result and add 2 strings.



```
public class AddConcatenate {
    public int addTwoNumbers(int a, int b) {
        return a + b;
    }
    public String concatTwoStrings(String a, String b) {
        return a + b;
    }
}
```

2. Create a test file named AddConcatTest.java

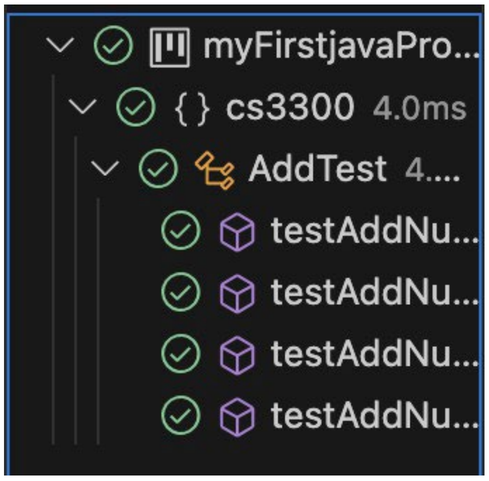
   Within this file, write a test using the @Test flag above the method.

```
package CS3300;

public class AddConcatTest {
    @Test
    public void testAddTwoNumbers() {
        //arrange
        int x = 23;
        int y = 35;
        String xx = "Georgia";
        String yy = "Tech";
        AddConcatenate ac = new AddConcatenate();

        //act
        int result1 = ac.addTwoNumbers(x, y);
        String result2 = ac.concatTwoStrings(xx, yy);

        //Assert
        assertEquals("23 + 35 must be 58", 58, result1);
        assertEquals("Georgia+Tech should be 
GeorgiaTech","GeorgiaTech", result2);
    }
}
```

3. Check that dependency in pom.xml is correct and is junit 4. Already installed.

4. Finish writing tests

5. Within the test tab on the left hand side bar, you can run the tests. You will see an output as seen in the image below if all tests pass. If not, you will see which test is failing and why.



myFirstjavaPro...
cs3300 4.0ms
AddTest 4....
testAddNu...
testAddNu...
testAddNu...
testAddNu...

6. Within the App.java, create a new method that adds two numbers together, and within the AppTest.java, create a JUnit test for the new method.
7. Run your tests