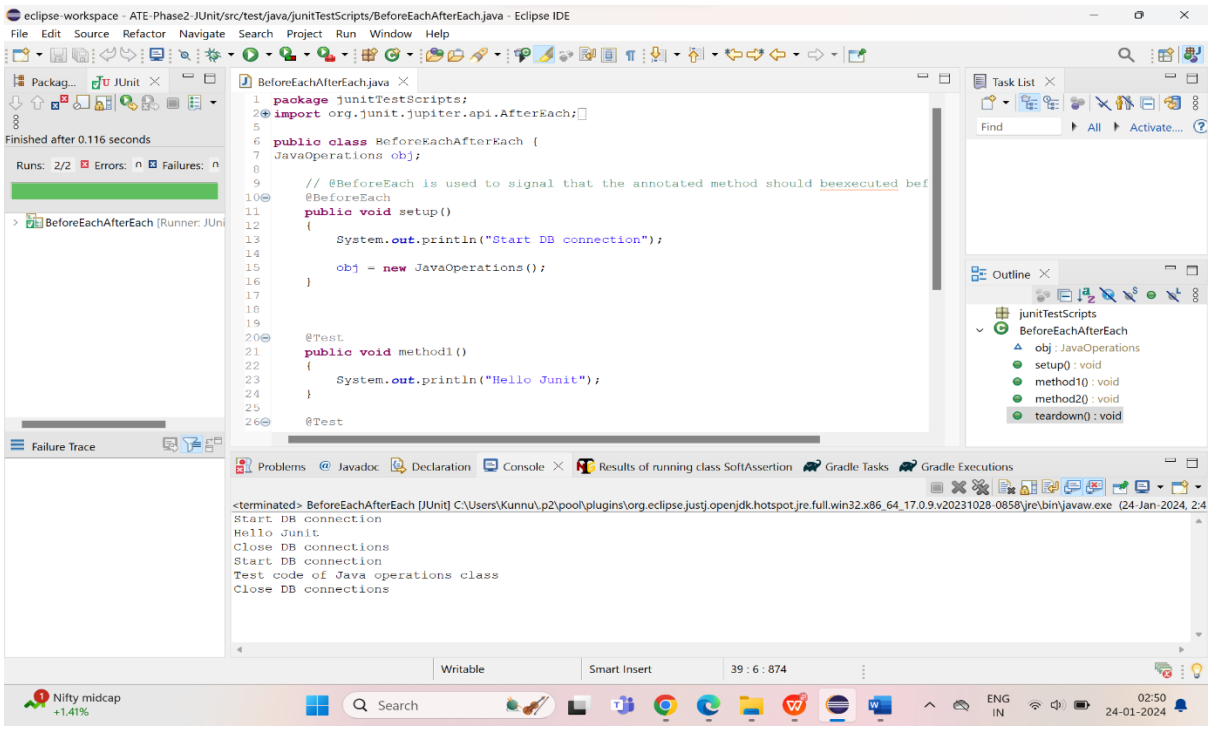
Life Cycle Methods:

```
package junitTestScripts;

import org.junit.jupiter.api.AfterEach;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;


public class BeforeEachAfterEach {

JavaOperations obj;


	// @BeforeEach is used to signal that the annotated method should beexecuted before each @Test

	@BeforeEach

	public void setup()

	{

		System.out.println("Start DB connection");


		obj = new JavaOperations();

	}




	@Test

	public void method1()

	{

		System.out.println("Hello Junit");

	}


	@Test

	public void method2()

	{

		System.out.println("Test code of Java operations class");
```

```
}

// @AfterEach : exeucte the method after each test method

// @AfterEach is used to signal that the annotated method should be executed after each @Test

@AfterEach

public void teardown()

{

System.out.println("Close DB connections");

}

}
```



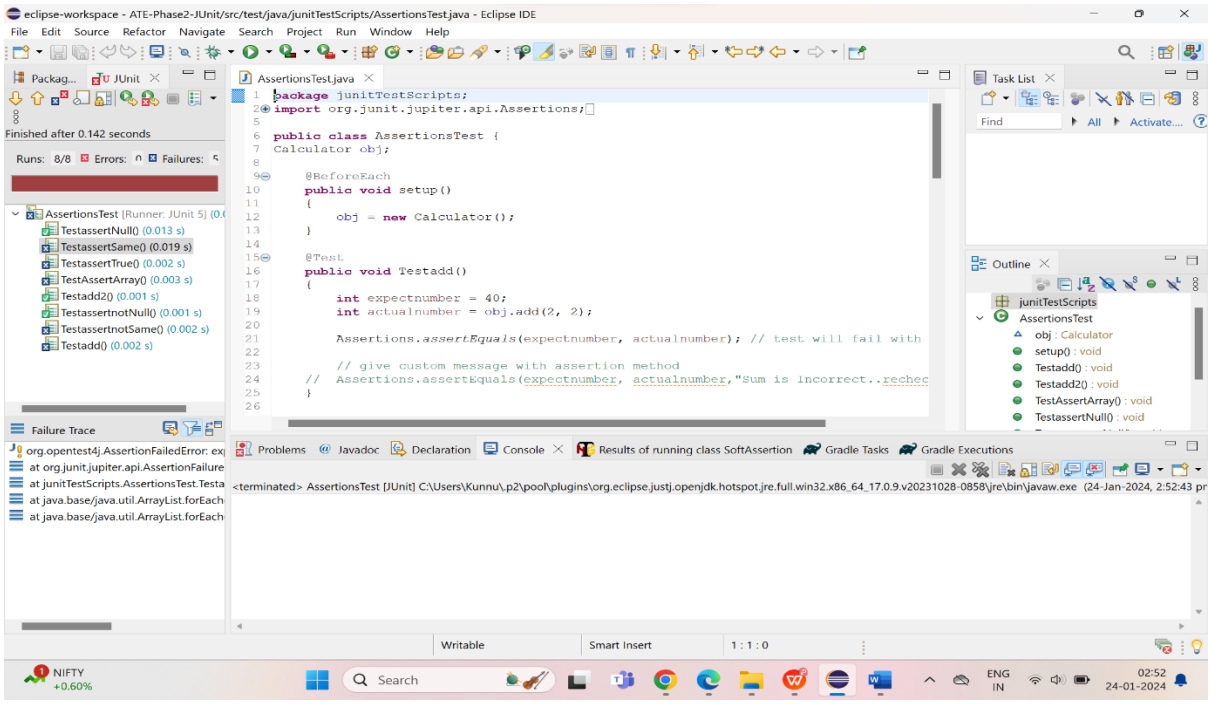
2.Assertions:

```
package junitTestScripts;

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.BeforeEach;

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

```
public class AssertionsTest {
Calculator obj;

	@BeforeEach
	public void setup()
	{
		obj = new Calculator();
	}

	@Test
	public void Testadd()
	{
		int expectnumber = 40;
		int actualnumber = obj.add(2, 2);

		Assertions.assertEquals(expectnumber, actualnumber); // test will fail with
assertion

		// give custom message with assertion method
	//	Assertions.assertEquals(expectnumber, actualnumber,"Sum is Incorrect..recheck
the code");
	}

	@Test
	public void Testadd2()
	{
		int expectnumber = 40;
		int actualnumber = obj.add(2, 2);
		Assertions.assertNotEquals(expectnumber, actualnumber); // test will pass
	}
	@Test
	public void TestAssertArray()
```

```
{

	Assertions.assertArrayEquals(new int[] {1,2,3}, new int[] {1,2,3});

	Assertions.assertArrayEquals(new int[] {1,2,3}, new int[] {1,2,3,4,5});

}

@Test
public void TestassertNull()
{
	String str1 = null;

	Assertions.assertNull(str1); // string is null, so test will pass

}

@Test
public void TestassertnotNull()
{
	String str1 = "selenium";

	Assertions.assertNotNull(str1); // string is not null, so test will pass



}

@Test
public void TestassertSame()
{
	String str1 = "selenium";
```

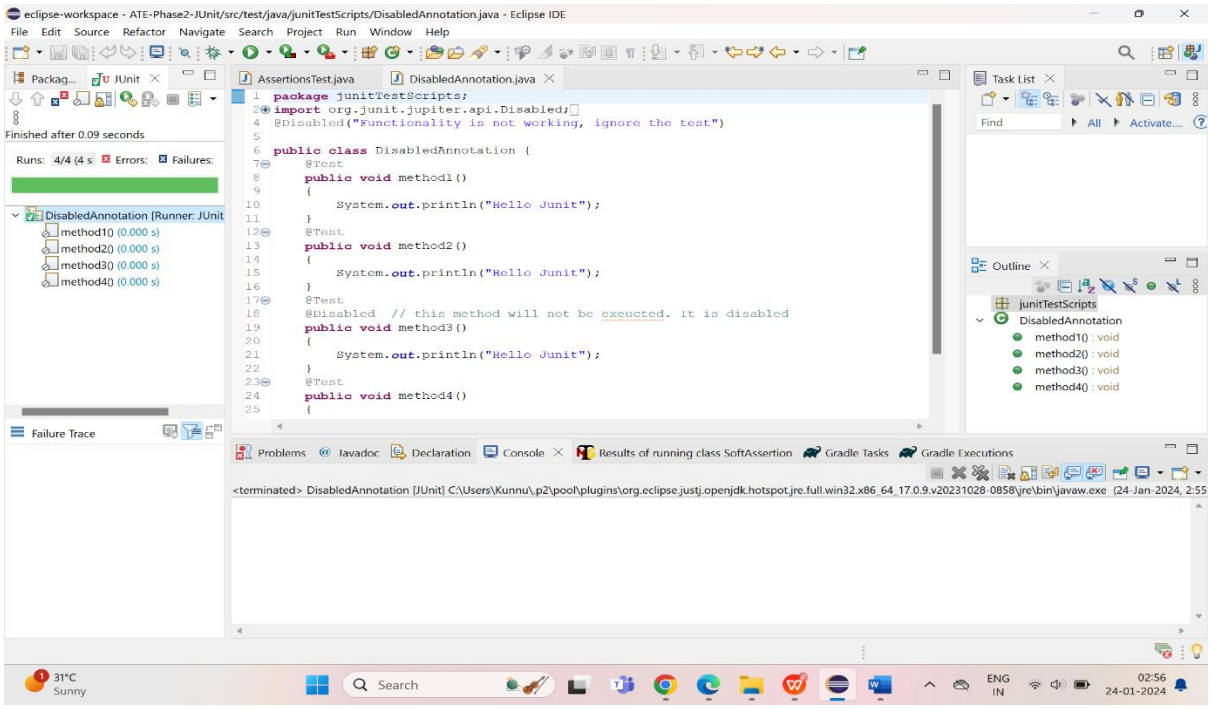
```
            String str2 = "tool";

            Assertions.assertSame(str1, str2);

        }

        @Test
        public void TestassertnotSame()
        {
            String str1 = "selenium";

            String str2 = "selenium";

            Assertions.assertNotSame(str1, str2);

        }
        @Test
        public void TestassertTrue()
        {
            int a =20;
            int b = 30;

            Assertions.assertTrue(a<b); // condition is true.. test will pass

            Assertions.assertFalse(a<b); // test will fail  as condition is true
            // for the above lines to pass, the condition should return false value
        }

}
```



3.Disabled Tests:

```
package junitTestScripts;

import org.junit.jupiter.api.Disabled;

import org.junit.jupiter.api.Test;

@Disabled("Functionality is not working, ignore the test")


public class DisabledAnnotation {

	@Test

	public void method1()

	{

		System.out.println("Hello Junit");

	}

	@Test

	public void method2()

	{

		System.out.println("Hello Junit");

	}

	@Test

	@Disabled  // this method will not be exeucted. It is disabled
```

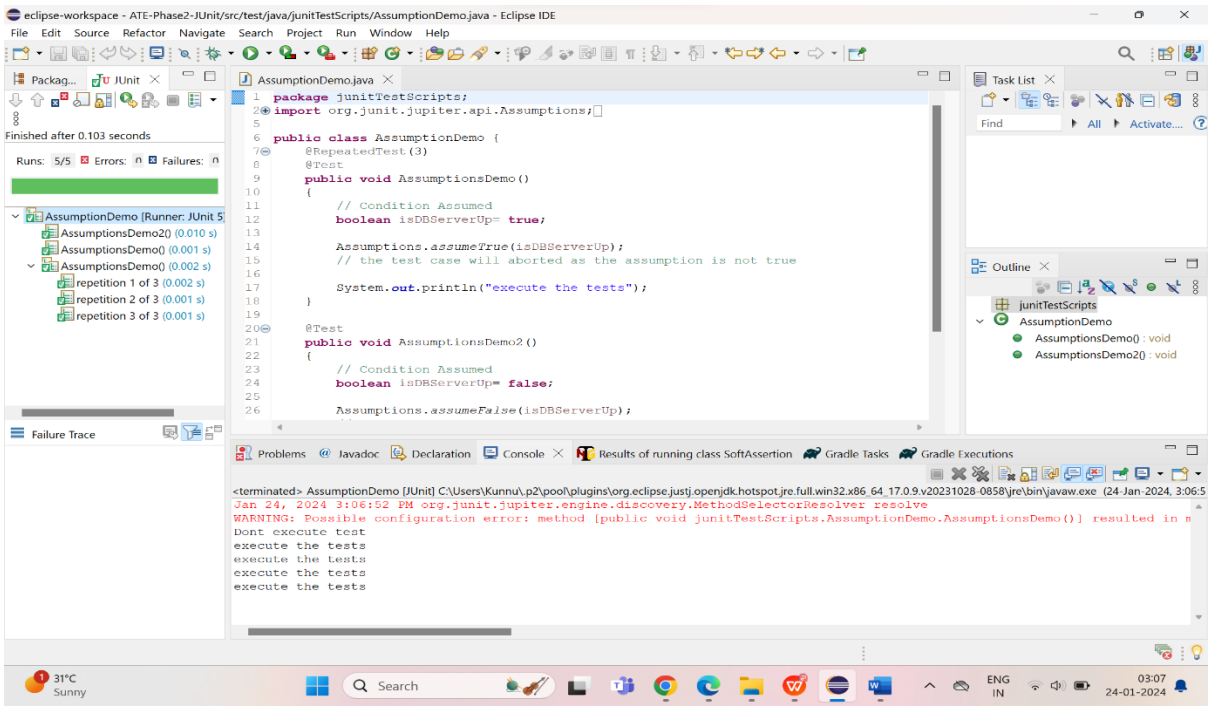
```
	public void method3()
	{
		System.out.println("Hello Junit");
	}
	@Test
	public void method4()
	{
		System.out.println("Hello Junit");
	}
}
```



4.Assumptions:

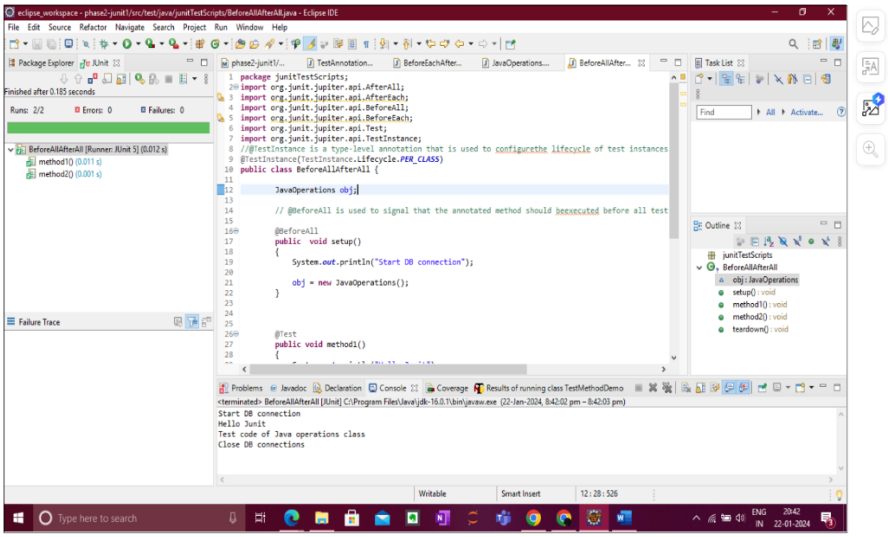
```
package junitTestScripts;

import org.junit.jupiter.api.Assumptions;

import org.junit.jupiter.api.RepeatedTest;

import org.junit.jupiter.api.Test;


public class AssumptionDemo {
	@RepeatedTest(3)
	@Test
```

```
public void AssumptionsDemo()
{
        // Condition Assumed
        boolean isDBServerUp= true;
        Assumptions.assumeTrue(isDBServerUp);
        // the test case will aborted as the assumption is not true
        System.out.println("execute the tests");
}

@Test
public void AssumptionsDemo2()
{
        // Condition Assumed
        boolean isDBServerUp= false;
        Assumptions.assumeFalse(isDBServerUp);
        // the test case will aborted as the assumption is not true
        System.out.println("Dont execute test");
}
}
```



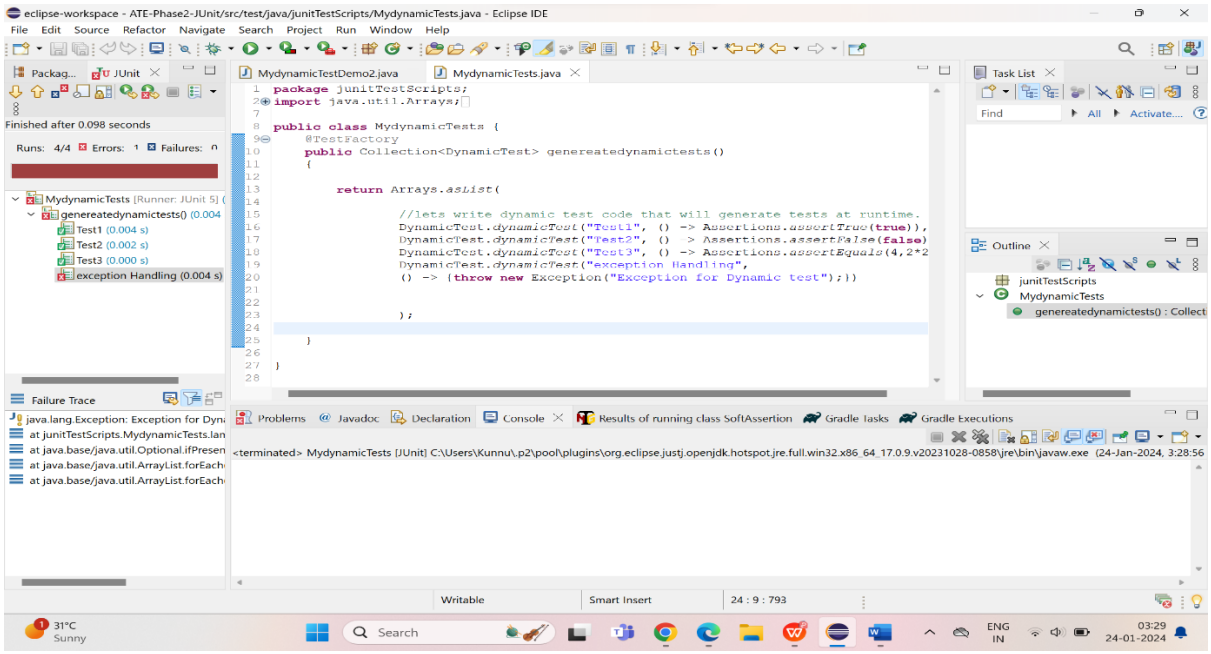
5.Test Interfaces:

```
package junitTestScripts;
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestInstance;
//@TestInstance is a type-level annotation that is used to configurethe lifecycle of test instances for
the annotatedtest class or test interface.
@TestInstance(TestInstance.Lifecycle.PER_CLASS)
public class BeforeAllAfterAll {
JavaOperations obj;
// @BeforeAll is used to signal that the annotated method should beexecuted before
all tests in the current test class
@BeforeAll
public void setup()
{
System.out.println("Start DB connection");
obj = new JavaOperations();
}
@Test
public void method1()
{
System.out.println("Hello Junit");
}
@Test
public void method2()
{
System.out.println("Test code of Java operations class");
}//
@AfterAll is used to signal that the annotated method should be executed after all
tests in the current test class.
@AfterAll
public void teardown()
{
System.out.println("Close DB connections");
}
}
```



6.Dynamic Tests:

```
package junitTestScripts;

import java.util.Arrays;

import java.util.Collection;

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.api.DynamicTest;

import org.junit.jupiter.api.TestFactory;


public class MydynamicTests {

	@TestFactory

	public Collection<DynamicTest> genereatedynamictests()

	{


		return Arrays.asList(


					//lets write dynamic test code that will generate tests at runtime.

					DynamicTest.dynamicTest("Test1", () ->
Assertions.assertTrue(true)),

					DynamicTest.dynamicTest("Test2", () ->
Assertions.assertFalse(false)),

					DynamicTest.dynamicTest("Test3", () ->
Assertions.assertEquals(4,2*2)),

					DynamicTest.dynamicTest("exception Handling",

					() -> {throw new Exception("Exception for Dynamic test");
}

					);


	}


}
```

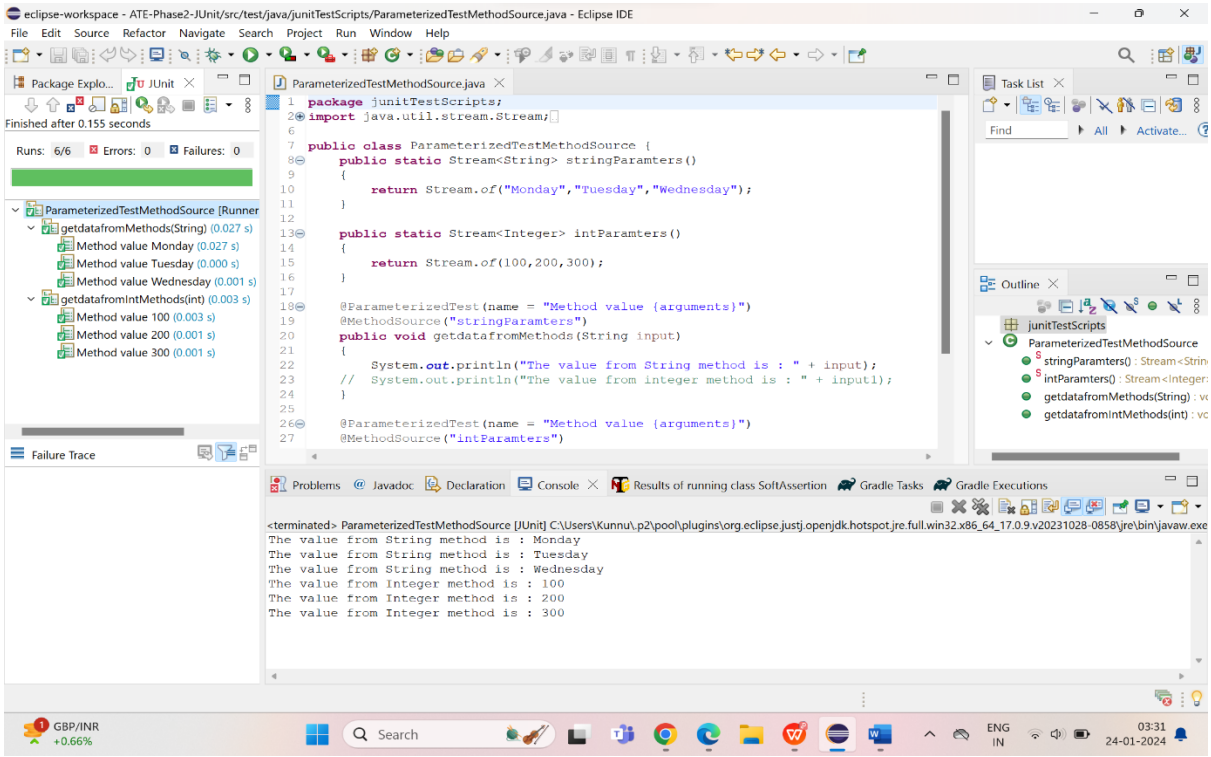


7.Parameterized Test:

```
package junitTestScripts;

import java.util.stream.Stream;


import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.MethodSource;


public class ParameterizedTestMethodSource {

	public static Stream<String> stringParamters()

	{

		return Stream.of("Monday","Tuesday","Wednesday");

	}


	public static Stream<Integer> intParamters()

	{

		return Stream.of(100,200,300);

	}


	@ParameterizedTest(name = "Method value {arguments}")

	@MethodSource("stringParamters")
```

```
public void getdatafromMethods(String input)
{
        System.out.println("The value from String method is : " + input);
//      System.out.println("The value from integer method is : " + input1);
}

@ParameterizedTest(name = "Method value {arguments}")
@MethodSource("intParamters")
public void getdatafromIntMethods(int input)
{
        System.out.println("The value from Integer method is : " + input);

}
```

}



8.Argument Source:

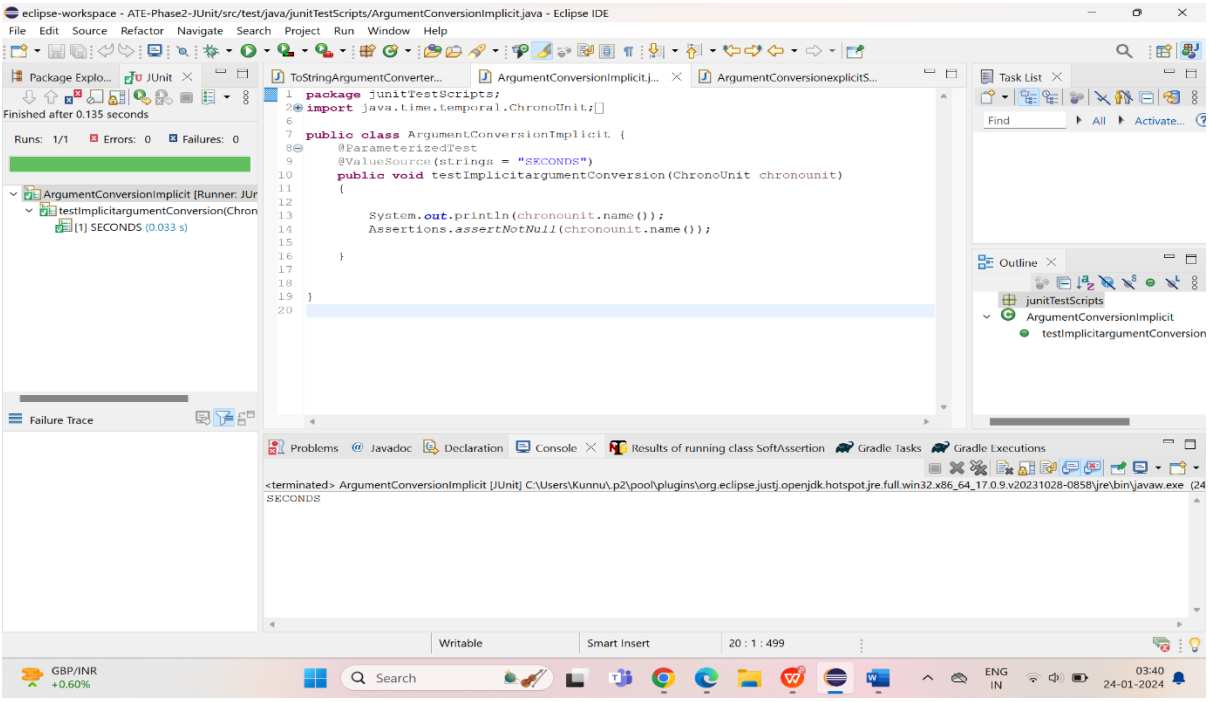
```
package junitTestScripts;

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

```
import org.junit.jupiter.params.converter.ArgumentConversionException;
import org.junit.jupiter.params.converter.SimpleArgumentConverter;

public class ToStringArgumentConverter extends SimpleArgumentConverter {
	@Override
	protected Object convert(Object source, Class<?> targetType) throws
ArgumentConversionException {

		Assertions.assertEquals(String.class, targetType, "Can only convert to String");

		return String.valueOf(source);
	}

}
```

Implicit Argument:

```
package junitTestScripts;
import java.time.temporal.ChronoUnit;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;

public class ArgumentConversionImplicit {
	@ParameterizedTest
	@ValueSource(strings = "SECONDS")
	public void testImplicitargumentConversion(ChronoUnit chronounit)
	{

		System.out.println(chronounit.name());
		Assertions.assertNotNull(chronounit.name());
```

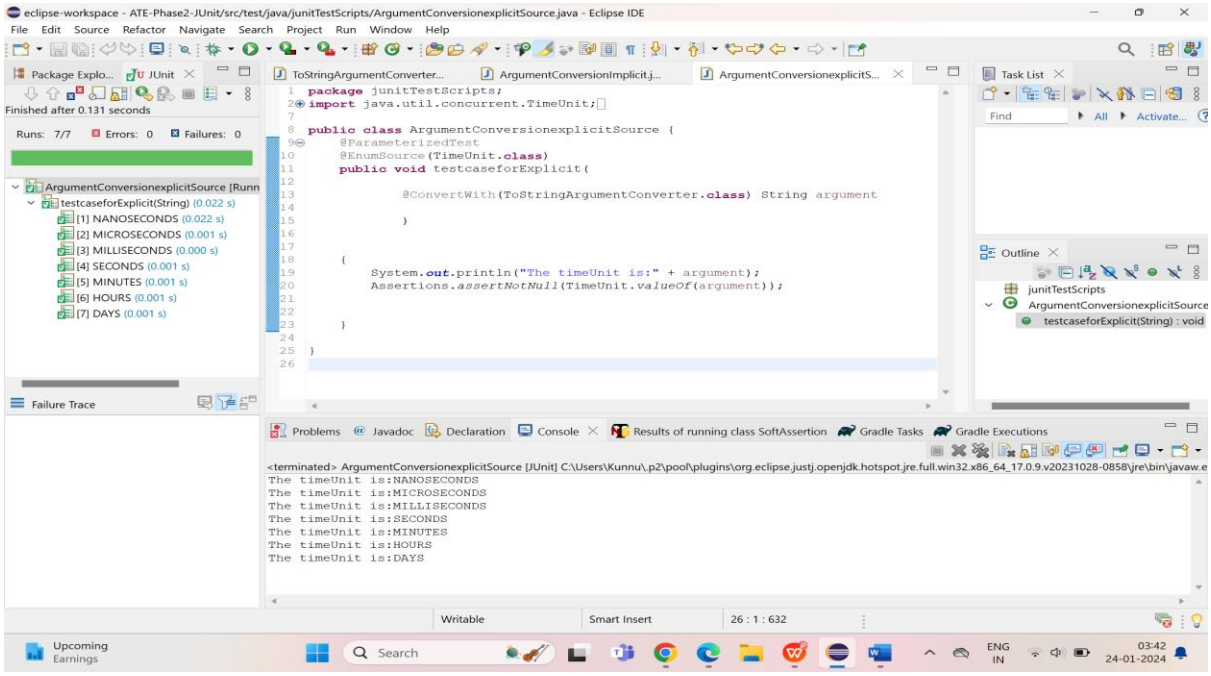
```
        }
}
```



10.Argument Conversion:

```
package junitTestScripts;

import java.util.concurrent.TimeUnit;

import org.junit.jupiter.api.Assertions;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.converter.ConvertWith;

import org.junit.jupiter.params.provider.EnumSource;


public class ArgumentConversionexplicitSource {

        @ParameterizedTest

        @EnumSource(TimeUnit.class)

        public void testcaseforExplicit(


                        @ConvertWith(ToStringArgumentConverter.class) String argument


                        )
```

```
    {
        System.out.println("The timeUnit is:" + argument);
        Assertions.assertNotNull(TimeUnit.valueOf(argument));


    }
}
```

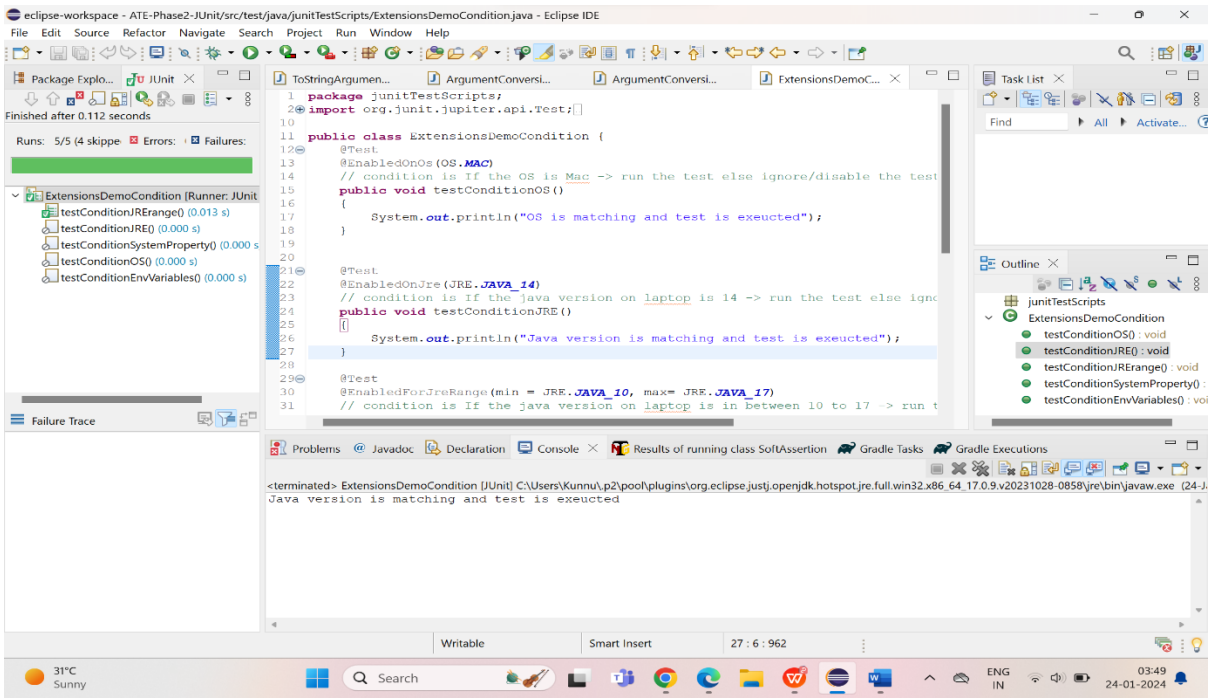


11.Extension Points:

```
package junitTestScripts;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.condition.EnabledForJreRange;

import org.junit.jupiter.api.condition.EnabledIfEnvironmentVariable;

import org.junit.jupiter.api.condition.EnabledIfSystemProperty;

import org.junit.jupiter.api.condition.EnabledOnJre;

import org.junit.jupiter.api.condition.EnabledOnOs;

import org.junit.jupiter.api.condition.JRE;

import org.junit.jupiter.api.condition.OS;
```

```
public class ExtensionsDemoCondition {
	@Test
	@EnabledOnOs(OS.MAC)
	// condition is If the OS is Mac -> run the test else ignore/disable the test
	public void testConditionOS()
	{
		System.out.println("OS is matching and test is exeucted");
	}


	@Test
	@EnabledOnJre(JRE.JAVA_14)
	// condition is If the java version on laptop is 14 -> run the test else ignore/disable the test
	public void testConditionJRE()
	{
		System.out.println("Java version is matching and test is exeucted");
	}

	@Test
	@EnabledForJreRange(min = JRE.JAVA_10, max= JRE.JAVA_17)
	// condition is If the java version on laptop is in between 10 to 17 -> run the test else ignore/disable the test
	public void testConditionJRErange()
	{
		System.out.println("Java version is matching and test is exeucted");
	}


@Test
@EnabledIfSystemProperty(named="java.vm.vendor", matches="Oracle.*")
// condition is If the java version is provided by oracle-> run the test else ignore/disable the test
```

```
public void testConditionSystemProperty()
{
        System.out.println("Java version is installed by oracle and test is exeucted");
}

@Test

 @EnabledIfEnvironmentVariable(named = "myENVOS", matches="Linux") //
 public void testConditionEnvVariables() {
 System.out.println(" Envirnoment varibale is linux and test is exeucted");
  }
}
```



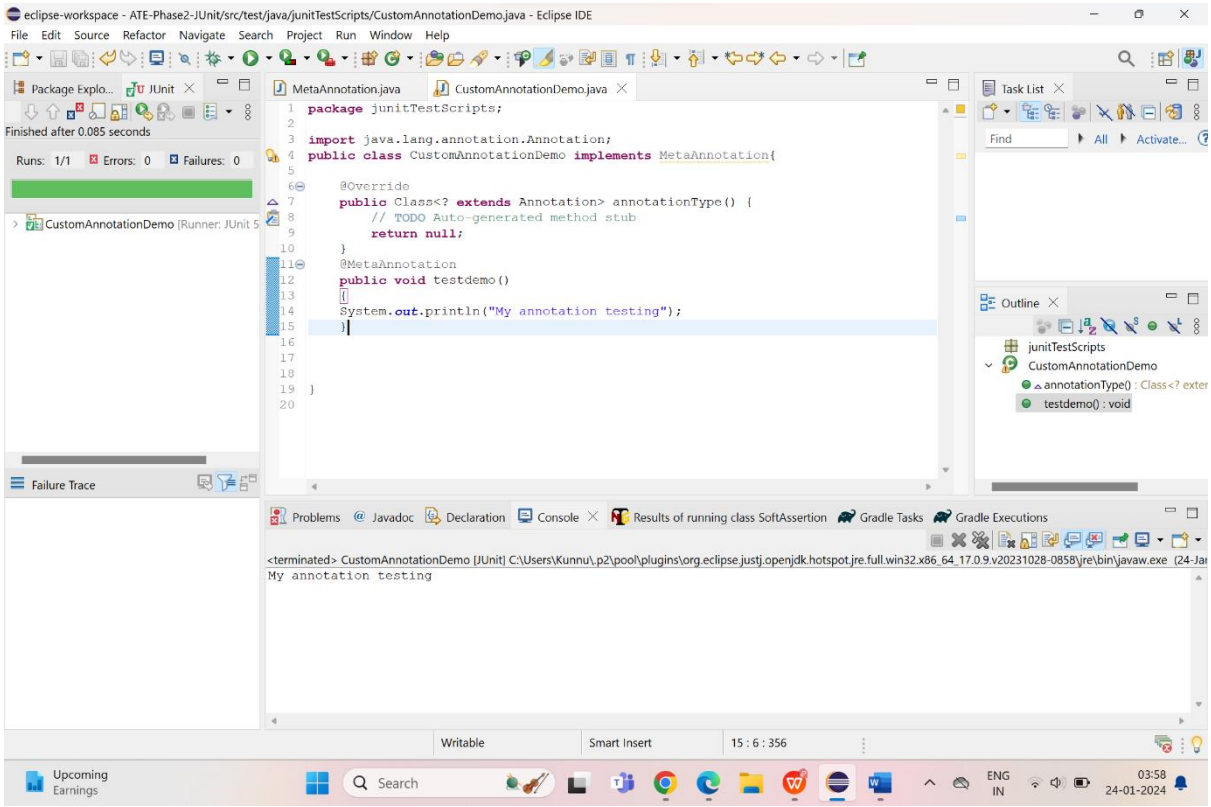
12.Meta Annotations:

```
package junitTestScripts;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;

        @Target({ElementType.TYPE, ElementType.METHOD})
```

```
@Retention(RetentionPolicy.RUNTIME)

@Tag("dev")
@Test
public @interface MetaAnnotation {
}
```

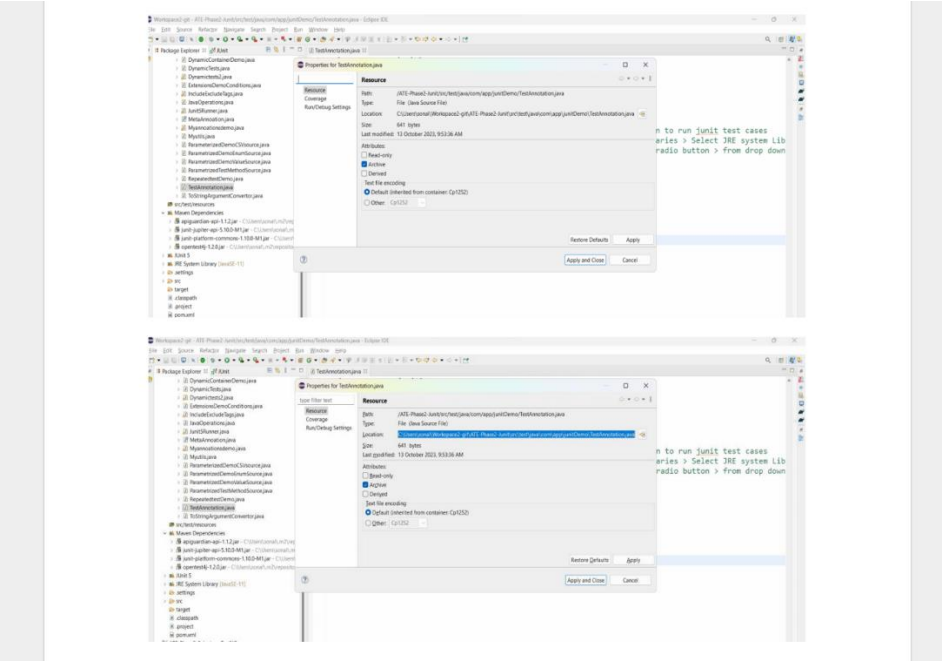
Custom Annotation Demo:

```
package junitTestScripts;

import java.lang.annotation.Annotation;
public class CustomAnnotationDemo implements MetaAnnotation{

	@Override
	public Class<? extends Annotation> annotationType() {
		// TODO Auto-generated method stub
		return null;
	}
	@MetaAnnotation
	public void testdemo()
	{
	System.out.println("My annotation testing");
	}

}
```



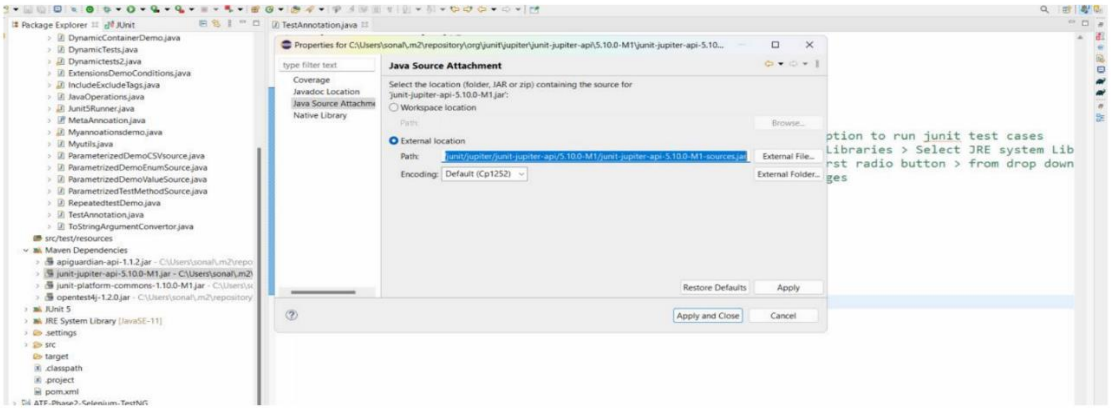
13.Running Tests from Console:

Right click on the java class file→Go to Properties → copy the path of the file location



Open the command prompt in your laptop and copy the path but remove the filename at the end
Like this cd C:\Users\sonal\Workspace2-git\ATE-Phase2-Junit\src\test\java\com\app\junitDemo

We also need the path of Junit Jar file on your project. We need to get that path from eclipse project
Right click on mavane depency folder > right click on junit jar and go to properties
Go to java source attachment and copt the path as show below
For me it is:
C:/Users/sonal/.m2/repository/org/junit/jupiter/junit-jupiter-api/5.10.0-M1/junit-jupiter-api-5.10.0-M1-
sources.jar



Execute below command on your command line now:
javac -cp C:/Users/sonal/.m2/repository/org/junit/jupiter/junit-jupiter-api/5.10.0-M1/junit-jupiter-api-
5.10.0-M1-sources.jar;. TestAnnotation.java

```
C:\Users\sonal\Workspace2-git\ATE-Phase2-Junit\src\test\java\com\app\junitDemo>javac -cp C:/Users/sonal/.m2/repository/org/junit/jupiter/junit-jupiter-api/5
.10.0-M1/junit-jupiter-api-5.10.0-M1-sources.jar;. TestAnnotation.java
```

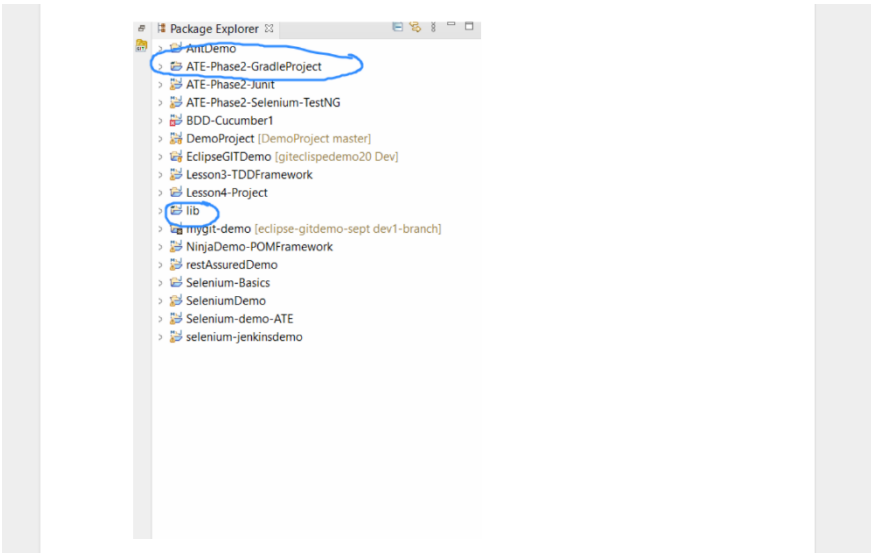
You will get error, ignore them

java -cp C:/Users/sonal/.m2/repository/org/junit/jupiter/junit-jupiter-api/5.10.0-M1/junit-jupiter-api-5.10.0-M1-sources.jar TestAnnotation.java

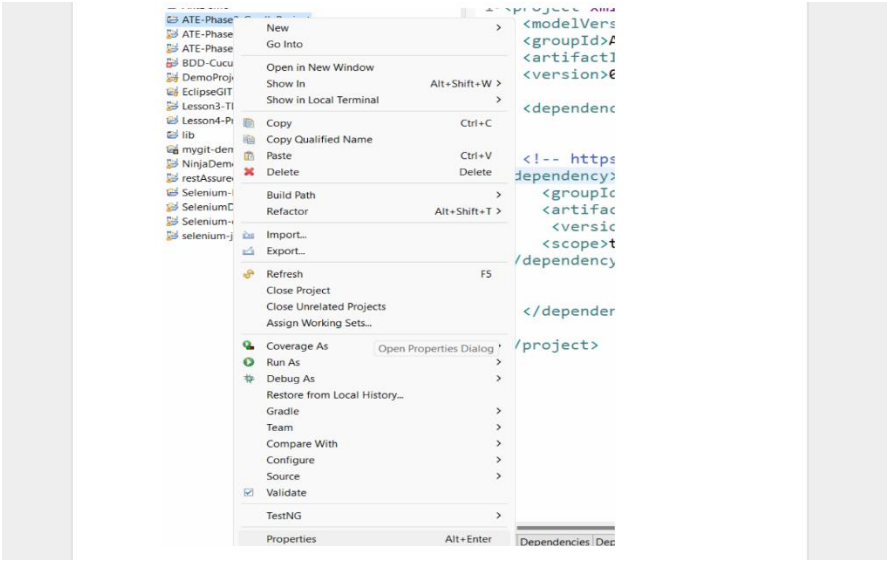
```
C:\Users\sonal\Workspace2-git\ATE-Phase2-Junit\src\test\java\com\app\junitDemo>java -cp C:/Users/sonal/.m2/repository/org/junit/jupiter/junit-jupiter-api/5.
10.0-M1/junit-jupiter-api-5.10.0-M1-sources.jar TestAnnotation.java
TestAnnotation.java:3: error: package org.junit.jupiter.api does not exist
import org.junit.jupiter.api.Test;
                           ^
TestAnnotation.java:13: error: cannot find symbol
       @Test  // execute the below test
        ^
  symbol:   class Test
  location: class TestAnnotation
2 errors
error: compilation failed

C:\Users\sonal\Workspace2-git\ATE-Phase2-Junit\src\test\java\com\app\junitDemo>
```

14.Running Tests From Gradle:

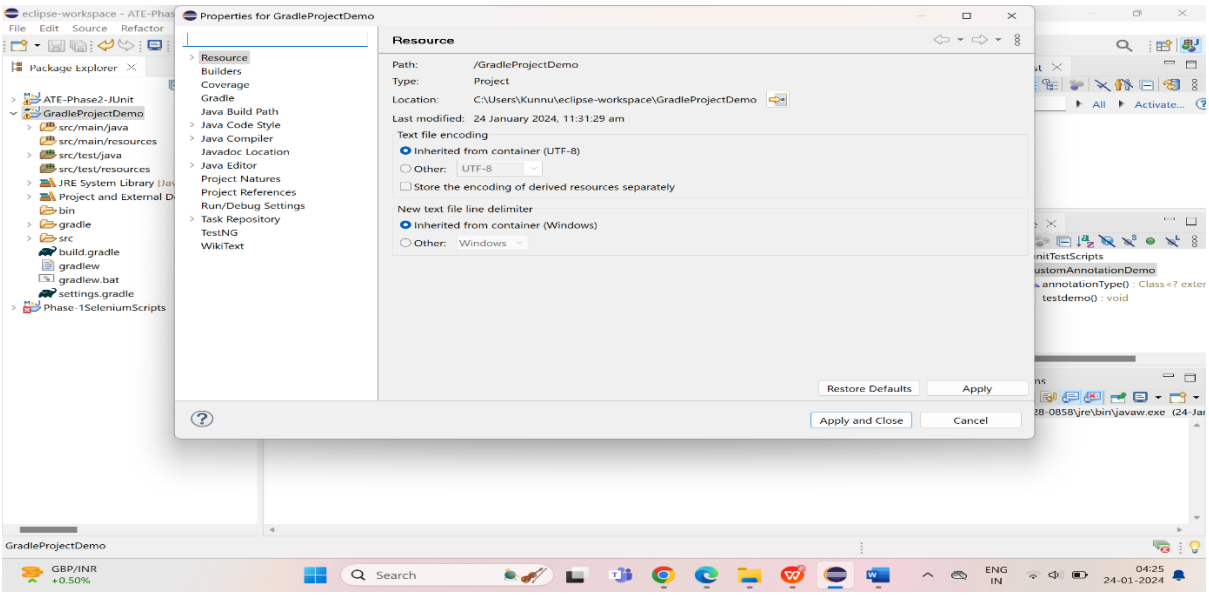
Create a Gradle project
Press Next button
Press Next button and give a name to the project and click on Finish
After Finish, it will create 2 projects



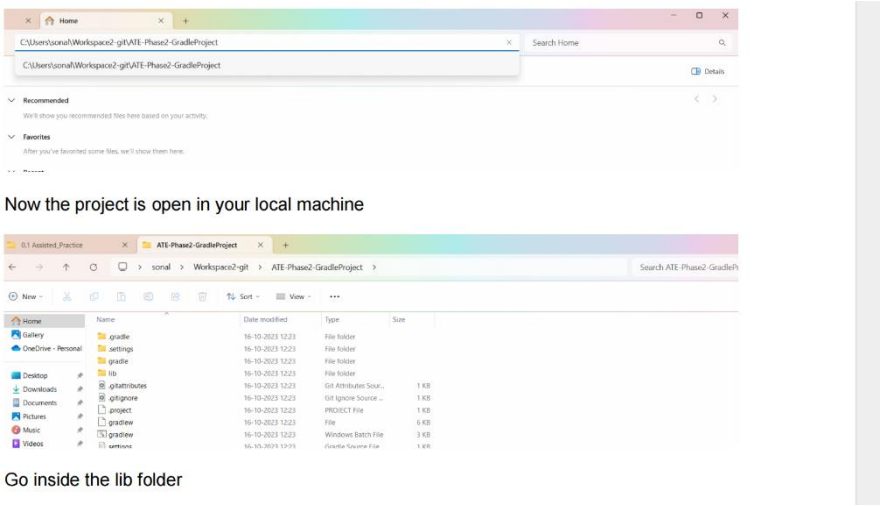
We have to fix the issue of lib folder should be inside the main gradle project
So go to properties of your gradle project and get the path of the project created on your local Machine



Copy the location of project on your local machine:

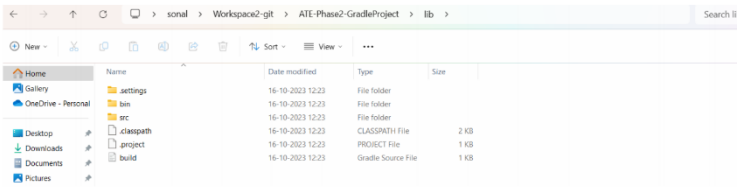


Open the folder in your local machine by pasting the path of workspace:

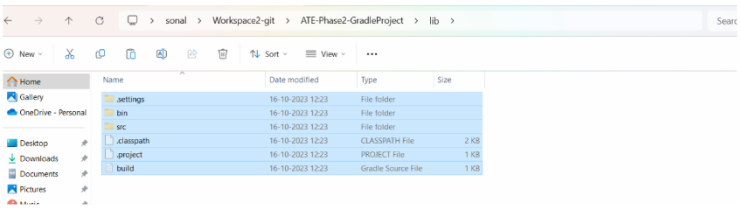


Now the project is open in your local machine

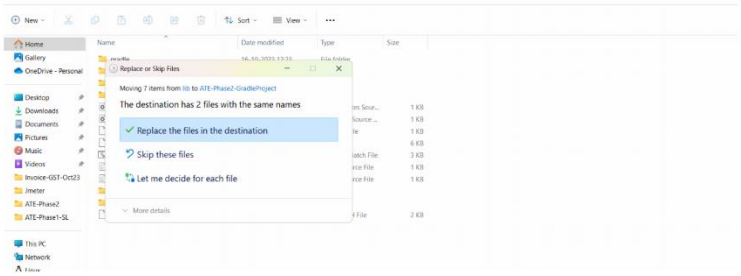
Go inside the lib folder



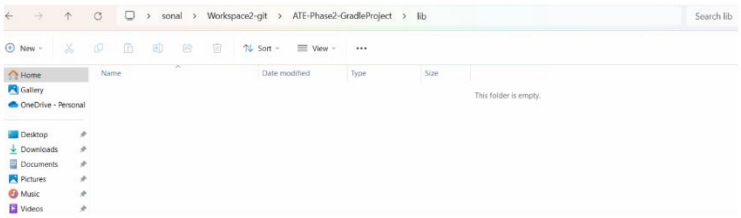
Select All the items in the lib folder and press CTL X (cut them)



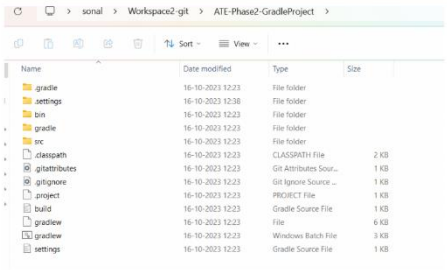
Now go back to your gradle project directory and paste the contents of lib folder.
It may ask for replace the files in the destination.
Select the option of Replace files in the destination



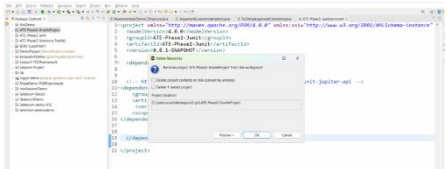
After this if you will go to lib folder, it will be empty:



Delete the Lib folder from gradle folder directory

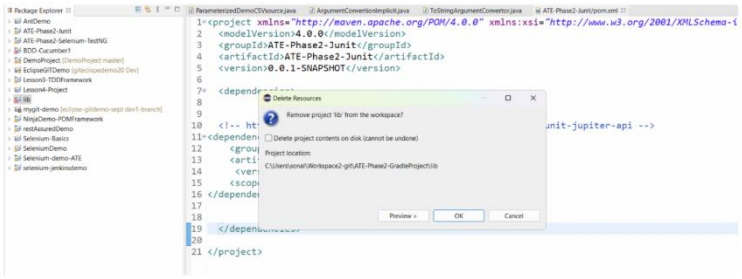


Now go to eclipse and delete the gradle project and lib project



Press OK

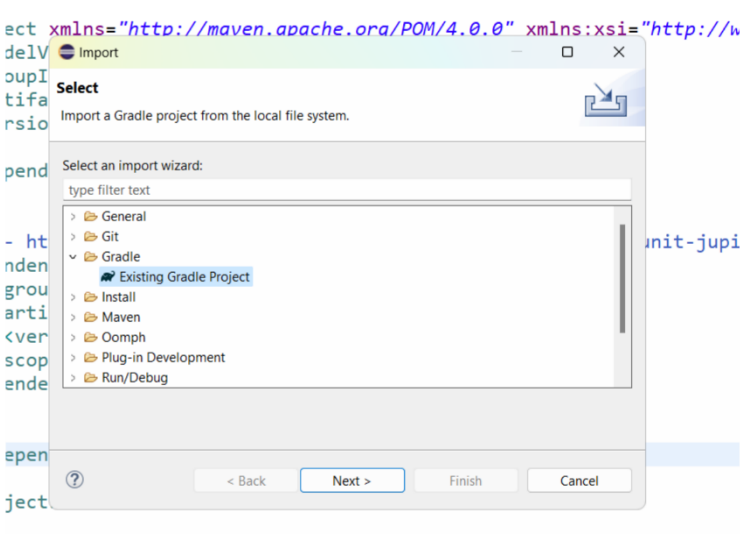
Delete the lib folder



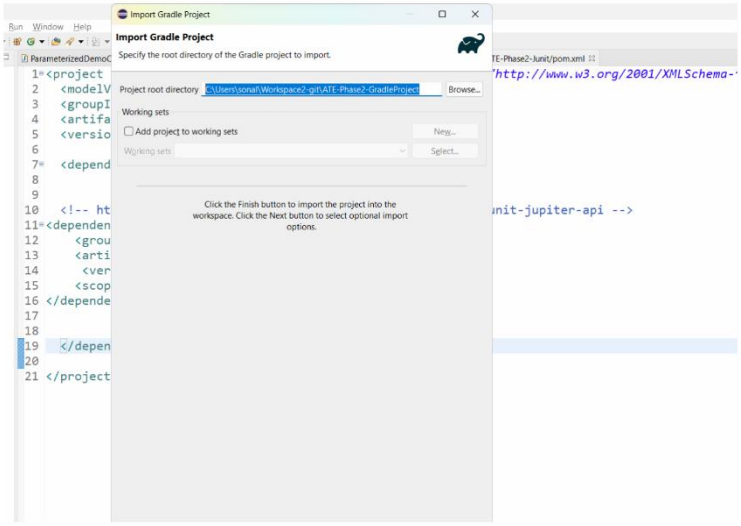
The gradle project are not there in eclipse but they are present in our local system
We need to import them to eclipse

Click on File in eclipse→ click on Import

Select Gradle→ importing Exisitng gradle project



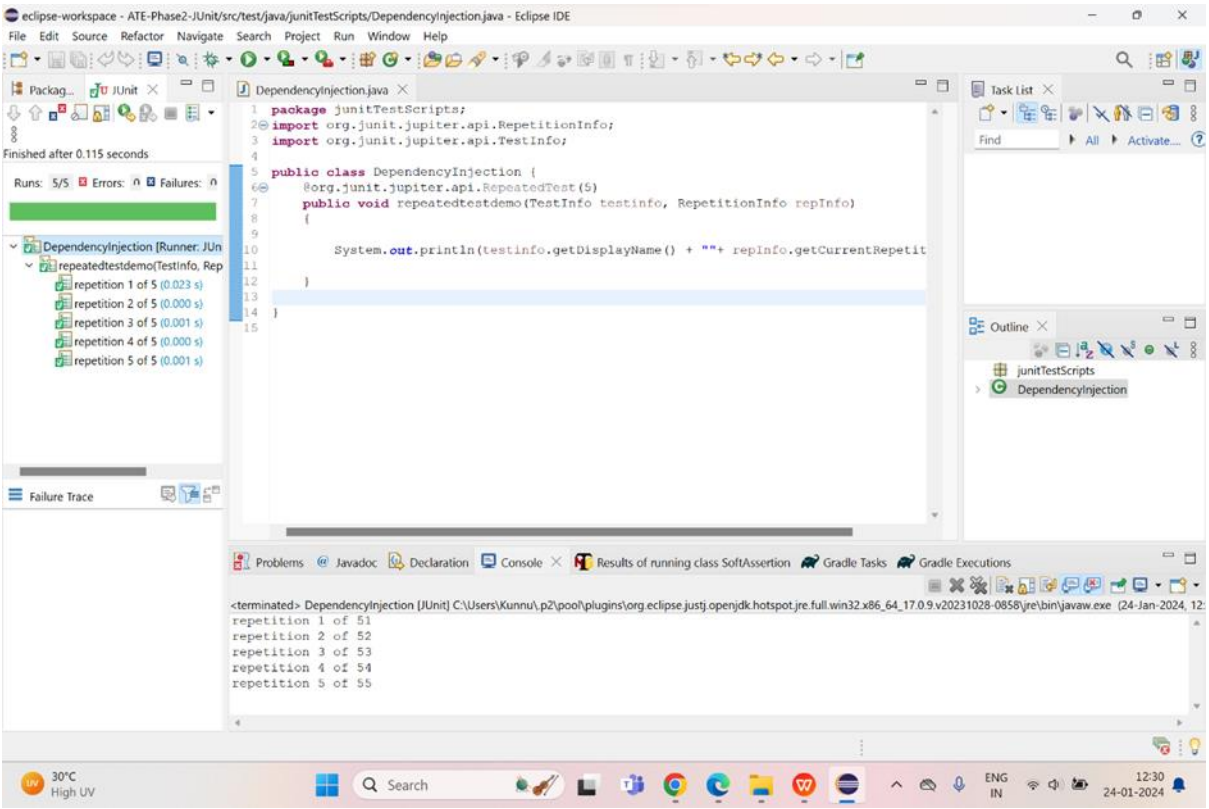
Press next
Click on browse button and go to the workspace path >> go to your project folder
You will have the path of your project populated automatically



Press Next and Again press next and click on finish

15.Running Tests From Maven:

```
package junitTestScripts;

import org.junit.jupiter.api.RepetitionInfo;

import org.junit.jupiter.api.TestInfo;


public class DependencyInjection {

	@org.junit.jupiter.api.RepeatedTest(5)

	public void repeatedtestdemo(TestInfo testinfo, RepetitionInfo repInfo)

	{

		System.out.println(testinfo.getDisplayName() + ""+ repInfo.getCurrentRepetition() );


	}

}
```



16.Include/Exclude Tags:

```
package junitTestScripts;

import org.junit.jupiter.api.Tag;

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

```
import org.junit.platform.suite.api.ExcludeTags;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
public class IncludeExcludeTags {
	@Test
	@Tag("feature1")
	public void feature1Test1()
	{
		System.out.println("test 1 for feature1");
	}

	@Test
	@Tag("feature1")
	public void feature1Test2()
	{
		System.out.println("test 2 for feature1");
	}

	@Test
	@Tag("feature1")
	public void feature1Test3()
	{
		System.out.println("test 3 for feature1");
	}

	@Test
	@Tag("feature1")
	public void feature1Test4()
	{
```

```
		System.out.println("test 4 for feature1");
	}

	@Test
	@Tag("feature2")
	public void feature2Test1()
	{
		System.out.println("test 1 for feature2");
	}
	@Test
	@Tag("feature2")
	public void feature2Test2()
	{
		System.out.println("test 2 for feature2");
	}
	@Test
	@Tag("feature2")
	public void feature2Test3()
	{
		System.out.println("test 3 for feature2");
	}
}
```