Junit – Unit Testing Framework for JAVA

Testing ensures the code quality. Before delivering the code to the client, it is important to test the code against all the requirements.

Testing ensures the code is taking care of all the requirements specified in the requirement documents.

Types of Testing

1. Unit Testing (Testing individual components by the coder)
2. Integration Testing (Testing the performance of code along with other files) [e2e – End to End Test]
3. Load Testing (Performance Testing)
4. Black/White/Grey Box & Sand Box Testing

TDD – Test Driven Development – It’s a method of Developing Applications/Projects

1. Write the Test Case First
2. The test case fails because no application code
3. Then write the code to make the test pass
4. Repeat it.

Unit Testing refers to Testing individual component in isolation.

1. One Class – One Test file to test that class
2. Test file will consist of one or more test cases

Junit – Official Site - <https://junit.org/junit5/>

Maven is a project management tool

This will help to build, test, package, deploy and also manage the project dependencies.

Eclipse – plugins (Eclipse EE – Maven plugin also integrated with it)

Junit is a Unit Testing Framework for JAVA

NUnit is a Unit Testing Framework for .Net

Types of Java Project –

* Core JAVA Project (Stand-alone Projects – project which gets executed in JVM directly with the help of command line)
* Adv Java Project (Web Based [Servlet/JSP based] that can be executed in browser)
* JPA (Java Persistence API projects to interact with DATABASES)

All these types of Java projects can be created normally or using some tools (maven, graddle)

Maven & Graddle supports JVM based Languages

Java is JVM based Language.

Kotlin is also JVM based Language

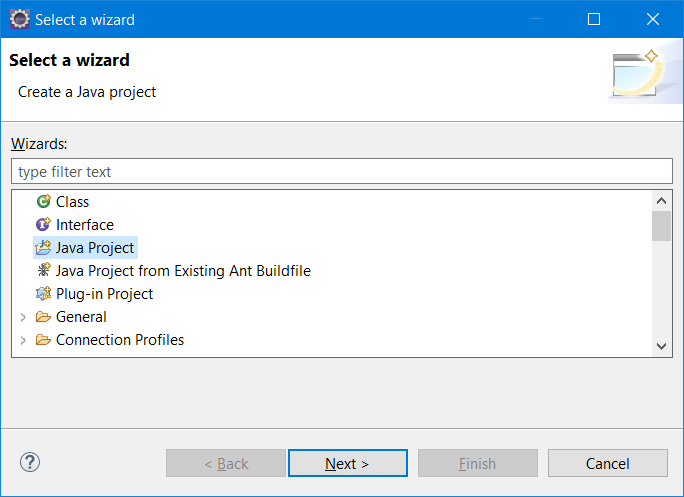
Groovy is also JVM based Language

Maven & Graddle – Project Management Tool

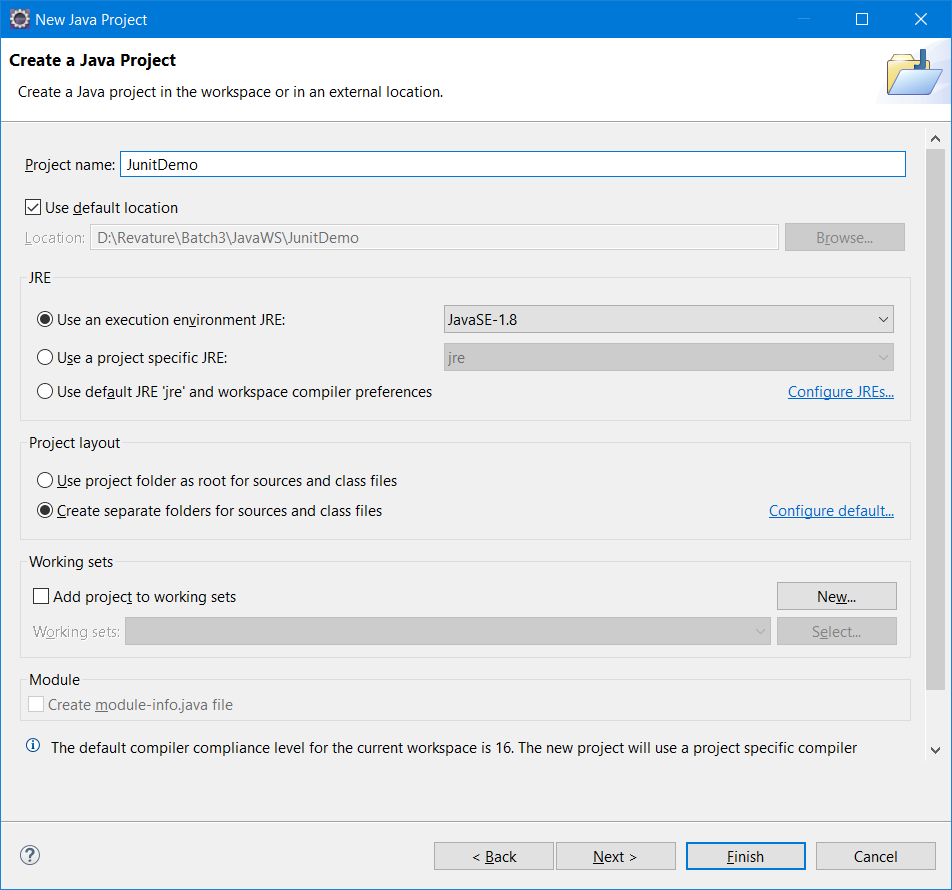
1. Compile/Build the code (converting .java to .class)
2. Test the Code (It will run the Junit test cases)
3. Package (convert in to a single package .jar/.war/.ear)
4. Deploy (load the compressed code to server/jvm)
5. Manage all the project dependent files. (dependency management)

To Create a Junit Test

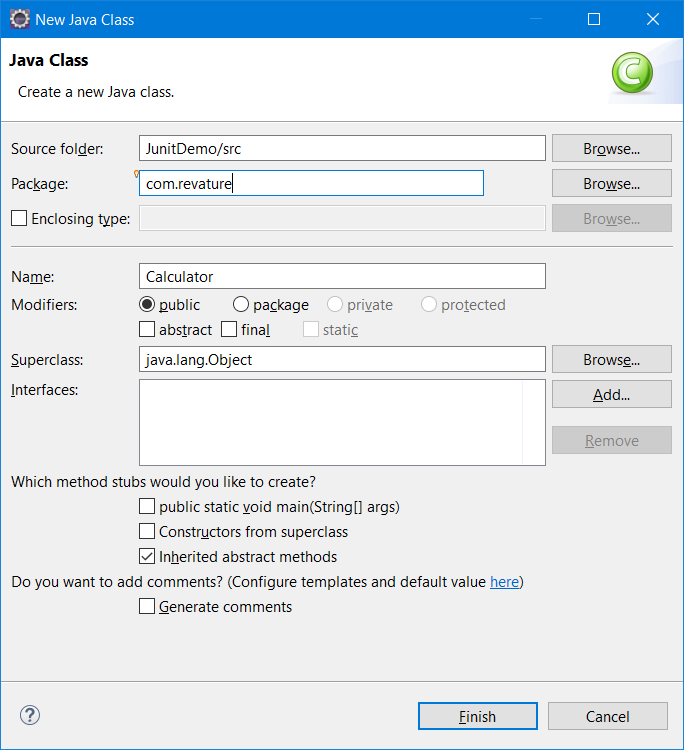
Step 1: Create a Java Project (file 🡪 New 🡪 Project)



Step 2: enter the name of the project as “Junit Demo” and click on “Finish” button



Step 3: Add a new Class by right clicking the project and selecting 🡪 New 🡪 Class)



**package** com.revature;

**public** **class** Calculator {

**public** **int** add(**int** a, **int** b) {

**return** (a + b);

}

**public** **int** sub(**int** a, **int** b) {

**return** (a - b);

}

**public** **int** mul(**int** a, **int** b) {

**return** (a \* b);

}

**public** **int** div(**int** a, **int** b) {

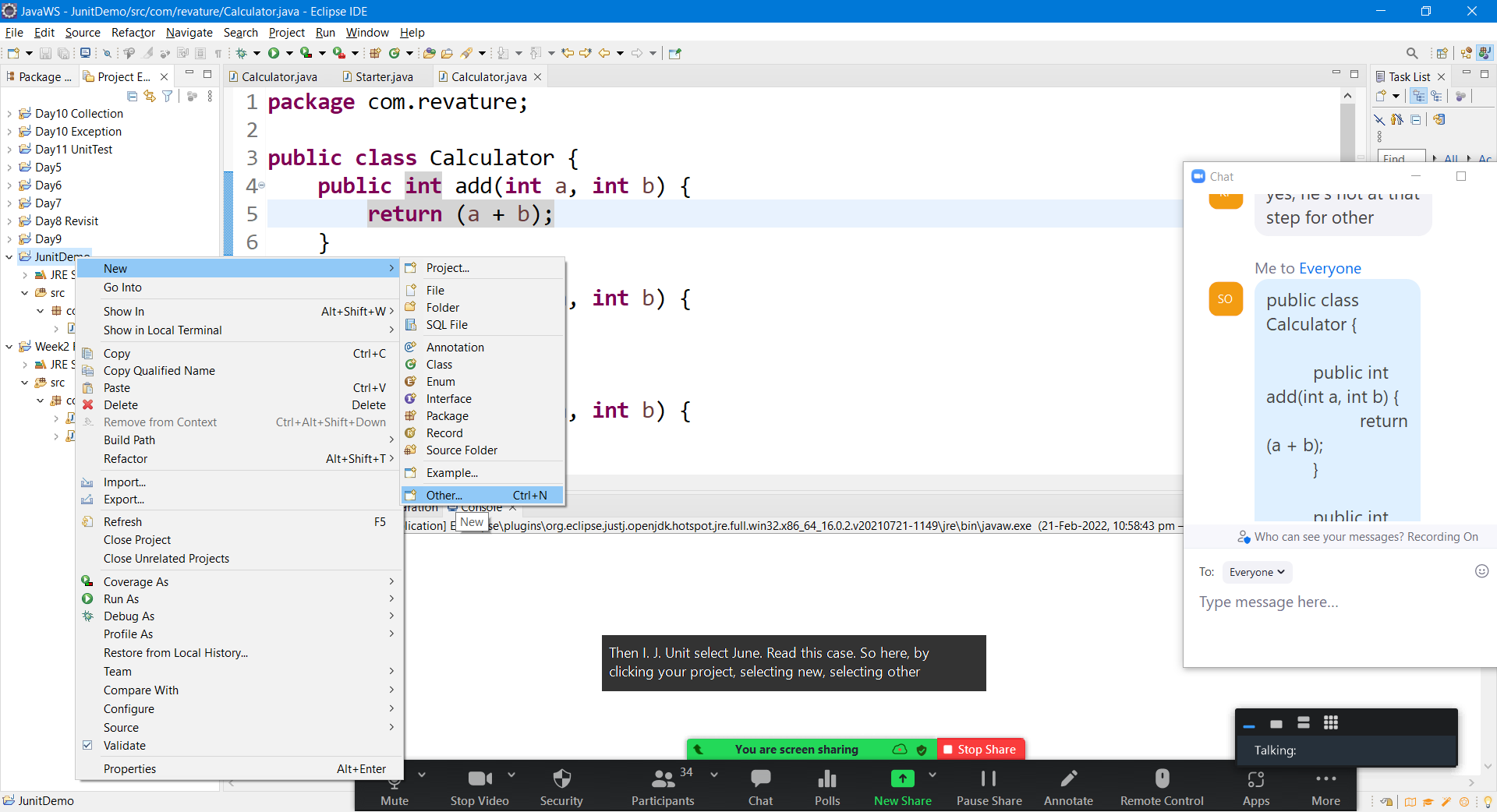
**return** (a / b);

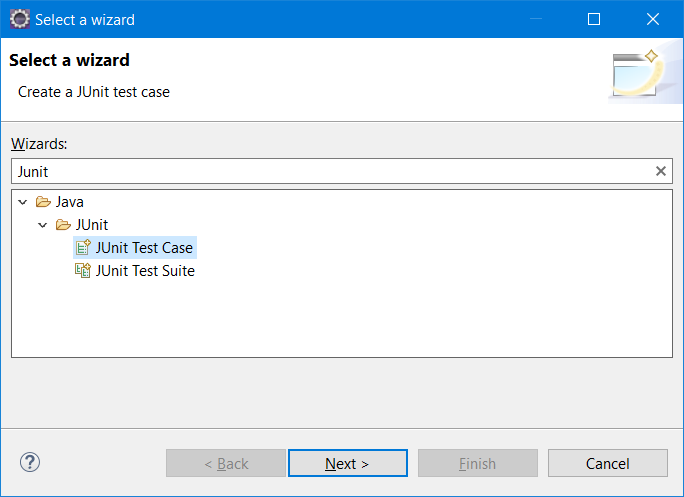
}

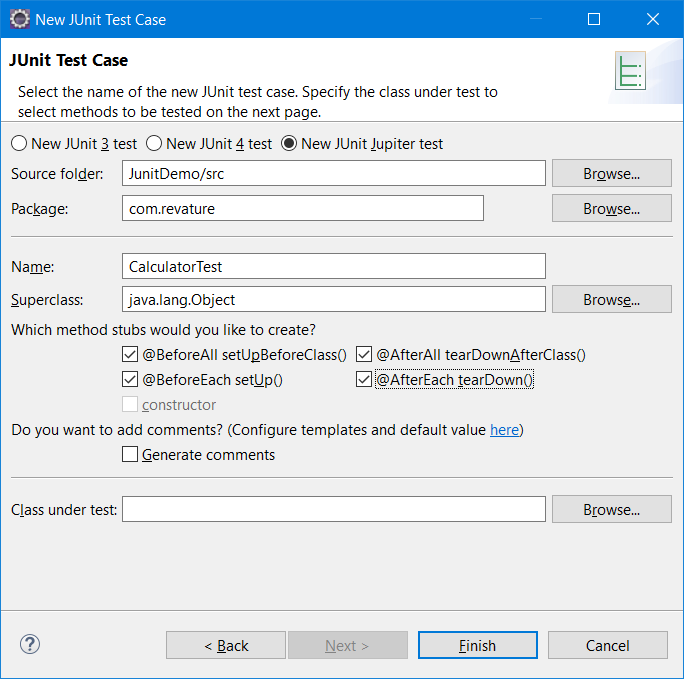
}

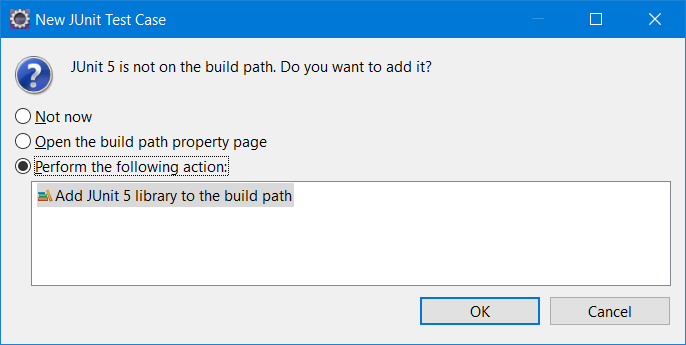
Step 4: Writing a Unit Test case file called CalculatorTest.java (Unit Test code will be added in this class file)

Right -click the project 🡪 New 🡪 Other 🡪 type Junit 🡪 Junit Test Case









Automatically the Test code will get created

**package** com.revature;

**import** **static** org.junit.jupiter.api.Assertions.\*;

**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;

**class** CalculatorTest {

@BeforeAll

**static** **void** setUpBeforeClass() **throws** Exception {

}

@AfterAll

**static** **void** tearDownAfterClass() **throws** Exception {

}

@BeforeEach

**void** setUp() **throws** Exception {

}

@AfterEach

**void** tearDown() **throws** Exception {

}

//Test Case

@Test

**void** test() {

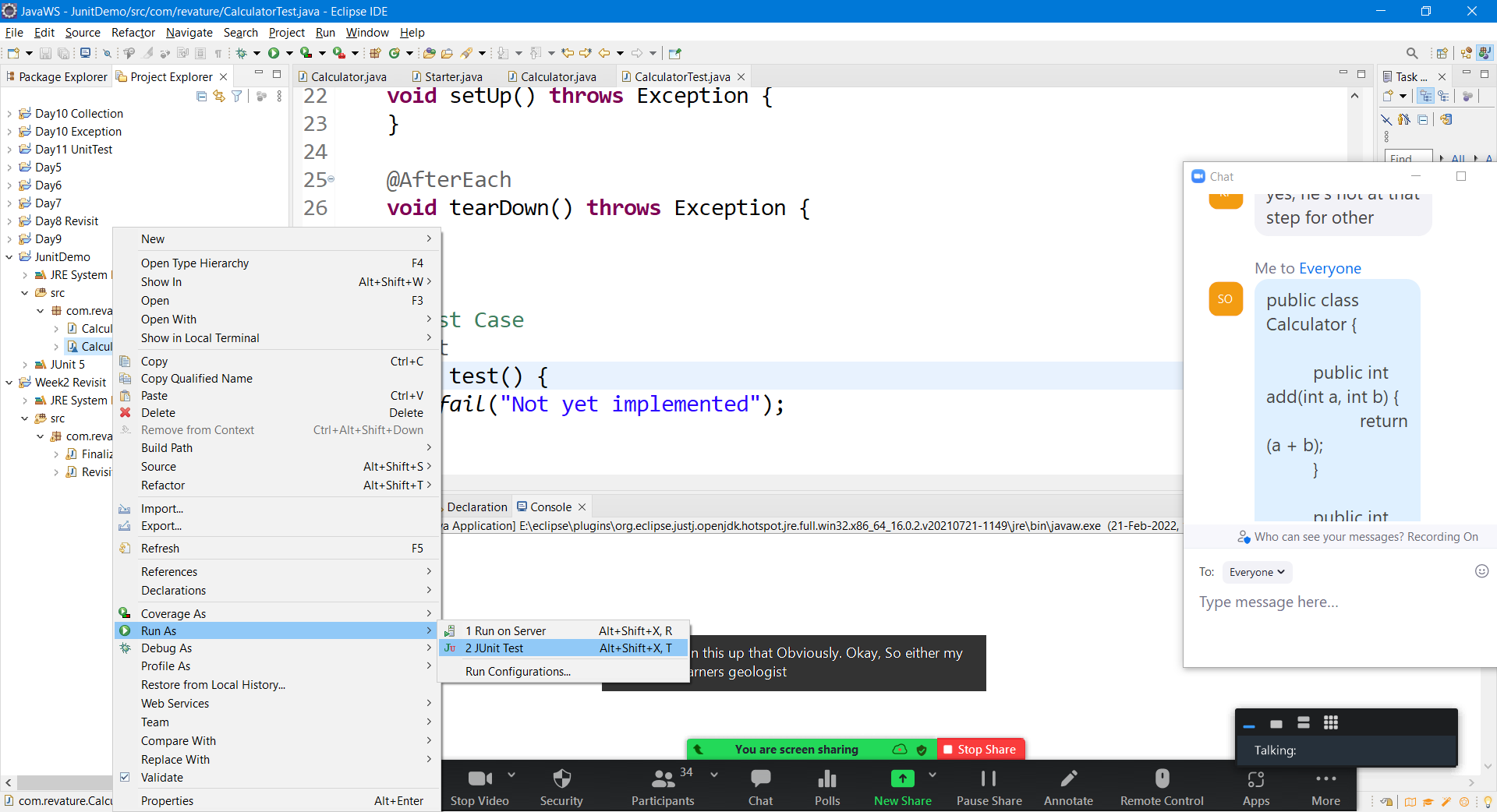
*fail*("Not yet implemented");

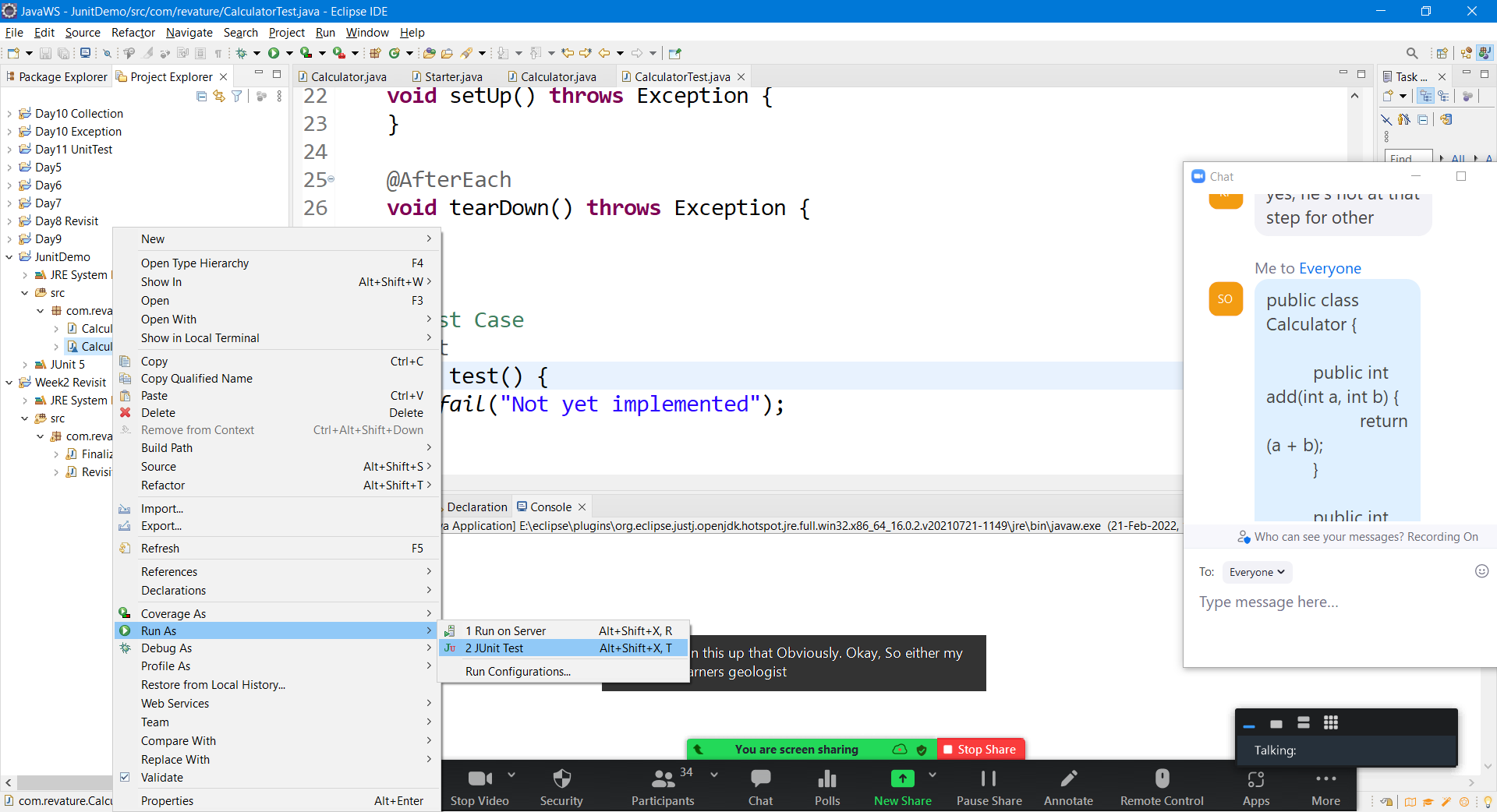
}

}

Running the Test Case

Right Click the project or Right click the unit test code class 🡪 Run As 🡪 Junit test





Getting user input from console.

With the help of Scanner class, reading input from user is the easiest way, simplest method.

**package** com.revature;

**import** java.util.Scanner;

**public** **class** UserInput {

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

**int** a = 0;

**float** b = 0;

String g = "";

**double** d = 0;

Scanner input = **new** Scanner(System.***in***);

// the default input stream in java is console & keyboard

System.***out***.print("Enter Number for a = ");

a = input.nextInt();

System.***out***.println(" Entered Number is a=" + a);

System.***out***.print("Enter Float for b = ");

b = input.nextFloat();

System.***out***.println(" Entered Float Number is b=" + b);

System.***out***.print("Enter any character = ");

g = input.next();

System.***out***.println(" Entered Character is b=" + g);

System.***out***.print("Enter Double value for d = ");

d = input.nextDouble();

System.***out***.println(" Entered Double value for d = " + d);

input.close();

}

}

Output :

Enter Number for a = 564

Entered Number is a=564

Enter Float for b = 87.98

Entered Float Number is b=87.98

Enter any character = j

Entered Character is b=j

Enter Double value for d = 9.0978678

Entered Double value for d = 9.0978678

Assert Methods in Junit 5

assertTrue(Expression that evaluate to true)

assertFalse(Expression that evaluate to false)

assertEquals(Expected, Actual)

assertNotEquals(Expected, Actual)

assertAll()

assertArrayEquals()

| **Annotation** | **Description** |
| --- | --- |
| @Test | Denotes that a method is a test method. Unlike JUnit 4’s @Test annotation, this annotation does not declare any attributes, since test extensions in JUnit Jupiter operate based on their own dedicated annotations. Such methods are inherited unless they are overridden. |
| @ParameterizedTest | Denotes that a method is a [parameterized test](https://junit.org/junit5/docs/current/user-guide/#writing-tests-parameterized-tests). Such methods are inherited unless they are overridden. |
| @RepeatedTest | Denotes that a method is a test template for a [repeated test](https://junit.org/junit5/docs/current/user-guide/#writing-tests-repeated-tests). Such methods are inherited unless they are overridden. |
| @TestFactory | Denotes that a method is a test factory for [dynamic tests](https://junit.org/junit5/docs/current/user-guide/#writing-tests-dynamic-tests). Such methods are inherited unless they are overridden. |
| @TestTemplate | Denotes that a method is a [template for test cases](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-templates) designed to be invoked multiple times depending on the number of invocation contexts returned by the registered [providers](https://junit.org/junit5/docs/current/user-guide/#extensions-test-templates). Such methods are inherited unless they are overridden. |
| @TestClassOrder | Used to configure the [test class execution order](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-execution-order-classes) for @Nested test classes in the annotated test class. Such annotations are inherited. |
| @TestMethodOrder | Used to configure the [test method execution order](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-execution-order-methods) for the annotated test class; similar to JUnit 4’s @FixMethodOrder. Such annotations are inherited. |
| @TestInstance | Used to configure the [test instance lifecycle](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-instance-lifecycle) for the annotated test class. Such annotations are inherited. |
| @DisplayName | Declares a custom [display name](https://junit.org/junit5/docs/current/user-guide/#writing-tests-display-names) for the test class or test method. Such annotations are not inherited. |
| @DisplayNameGeneration | Declares a custom [display name generator](https://junit.org/junit5/docs/current/user-guide/#writing-tests-display-name-generator) for the test class. Such annotations are inherited. |
| @BeforeEach | Denotes that the annotated method should be executed before **each** @Test, @RepeatedTest, @ParameterizedTest, or @TestFactory method in the current class; analogous to JUnit 4’s @Before. Such methods are inherited unless they are overridden. |
| @AfterEach | Denotes that the annotated method should be executed after **each** @Test, @RepeatedTest, @ParameterizedTest, or @TestFactory method in the current class; analogous to JUnit 4’s @After. Such methods are inherited unless they are overridden. |
| @BeforeAll | Denotes that the annotated method should be executed before **all** @Test, @RepeatedTest, @ParameterizedTest, and @TestFactory methods in the current class; analogous to JUnit 4’s @BeforeClass. Such methods are inherited (unless they are hidden or overridden) and must be static (unless the "per-class" [test instance lifecycle](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-instance-lifecycle) is used). |
| @AfterAll | Denotes that the annotated method should be executed after **all** @Test, @RepeatedTest, @ParameterizedTest, and @TestFactory methods in the current class; analogous to JUnit 4’s @AfterClass. Such methods are inherited (unless they are hidden or overridden) and must be static (unless the "per-class" [test instance lifecycle](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-instance-lifecycle) is used). |
| @Nested | Denotes that the annotated class is a non-static [nested test class](https://junit.org/junit5/docs/current/user-guide/#writing-tests-nested). @BeforeAll and @AfterAll methods cannot be used directly in a @Nested test class unless the "per-class" [test instance lifecycle](https://junit.org/junit5/docs/current/user-guide/#writing-tests-test-instance-lifecycle) is used. Such annotations are not inherited. |
| @Tag | Used to declare [tags for filtering tests](https://junit.org/junit5/docs/current/user-guide/#writing-tests-tagging-and-filtering), either at the class or method level; analogous to test groups in TestNG or Categories in JUnit 4. Such annotations are inherited at the class level but not at the method level. |
| @Disabled | Used to [disable](https://junit.org/junit5/docs/current/user-guide/#writing-tests-disabling) a test class or test method; analogous to JUnit 4’s @Ignore. Such annotations are not inherited. |
| @Timeout | Used to fail a test, test factory, test template, or lifecycle method if its execution exceeds a given duration. Such annotations are inherited. |
| @ExtendWith | Used to [register extensions declaratively](https://junit.org/junit5/docs/current/user-guide/#extensions-registration-declarative). Such annotations are inherited. |
| @RegisterExtension | Used to [register extensions programmatically](https://junit.org/junit5/docs/current/user-guide/#extensions-registration-programmatic) via fields. Such fields are inherited unless they are shadowed. |
| @TempDir | Used to supply a [temporary directory](https://junit.org/junit5/docs/current/user-guide/#writing-tests-built-in-extensions-TempDirectory) via field injection or parameter injection in a lifecycle method or test method; located in the org.junit.jupiter.api.io package. |

Annotations in Java

Annotations – Nothing but meta data (Data about data)

Real time Example : Book (The Java Complete Reference – All info about Java programming lang)

MetaData – 1) TOC (Table of Contents) Info about chapter wise details, topics in each chapter along with the page number

1. Index Page (The concepts in alphabetical order, the page numbers where they are defined)

Annotation in Java starts with a @symbol.

Annotations can be applied any where in the java code.

Built-in Annotations, User-defined /custom annotations

Annotations are introduced in Java 1.5

Annotations help to avoid writing complex config files using xml.

Annotations can be applied on instance variables, local variables, arguments, parameters passed to a function/method, it can also be applied to methods, classes etc.,

**package** com.revature;

**import** java.util.Date;

**public** **class** AnnotationsDemo {

@SuppressWarnings({ "deprecation", "unused" })

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

@SuppressWarnings("deprecation")

Date today = **new** ~~Date~~(122, 0, 23);

System.***out***.println(today);

// Date in java, it starts from 1900

**int** a;

java.sql.Date myDay = **new** java.sql.~~Date~~(122,01,21);

System.***out***.println("SQL Date is = " + myDay);

}

}

Output :

Sun Jan 23 00:00:00 IST 2022

SQL Date is = 2022-02-21

Annotations are meta-data used to provide more info to the JVM.

Annotations are mainly used in Frameworks like springBoot, Hibernate etc.,

Annotation help you to avoid writting web.xml (deployment descriptor)

Custom annotations – User-defined annotations

Meta Annotations – Annotations used in custom annotation class (@Target, @Retention & @Documented )

Generics – It’s a way of Self-evaluation the java code by the JVM.

Generic is always used along with collection api.

Generics also help to provide meta data while creating any type collection class/interface.

The main purpose of generics is compile time safety.

Naming Convention of Generics

E – element

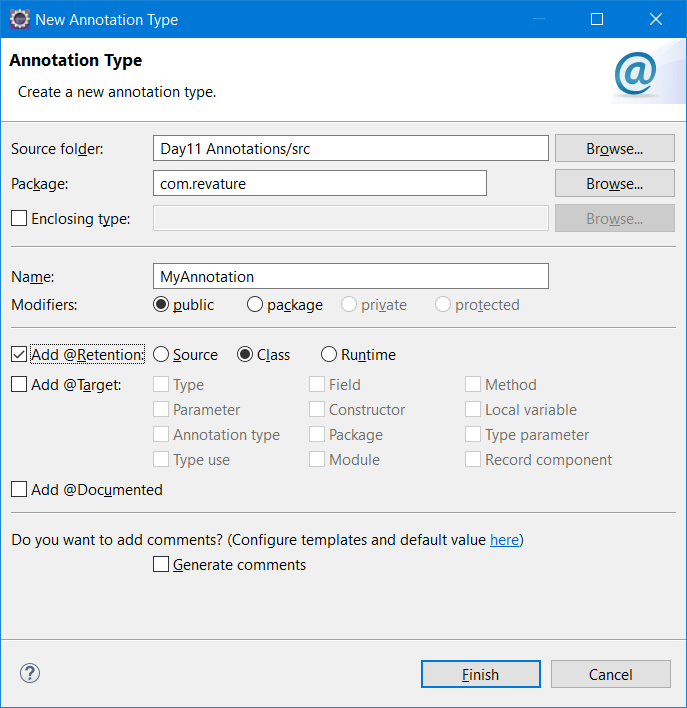
K – Key in Map

V – value in Map

T – Template

4 important built-in annotations

1. @Deprecated
2. @Override
3. @FunctionalInterface
4. @SuppressWarnings



import static java.lang.annotation.ElementType.CONSTRUCTOR;

import static java.lang.annotation.ElementType.LOCAL\_VARIABLE;

import static java.lang.annotation.RetentionPolicy.CLASS;

import java.lang.annotation.Retention;

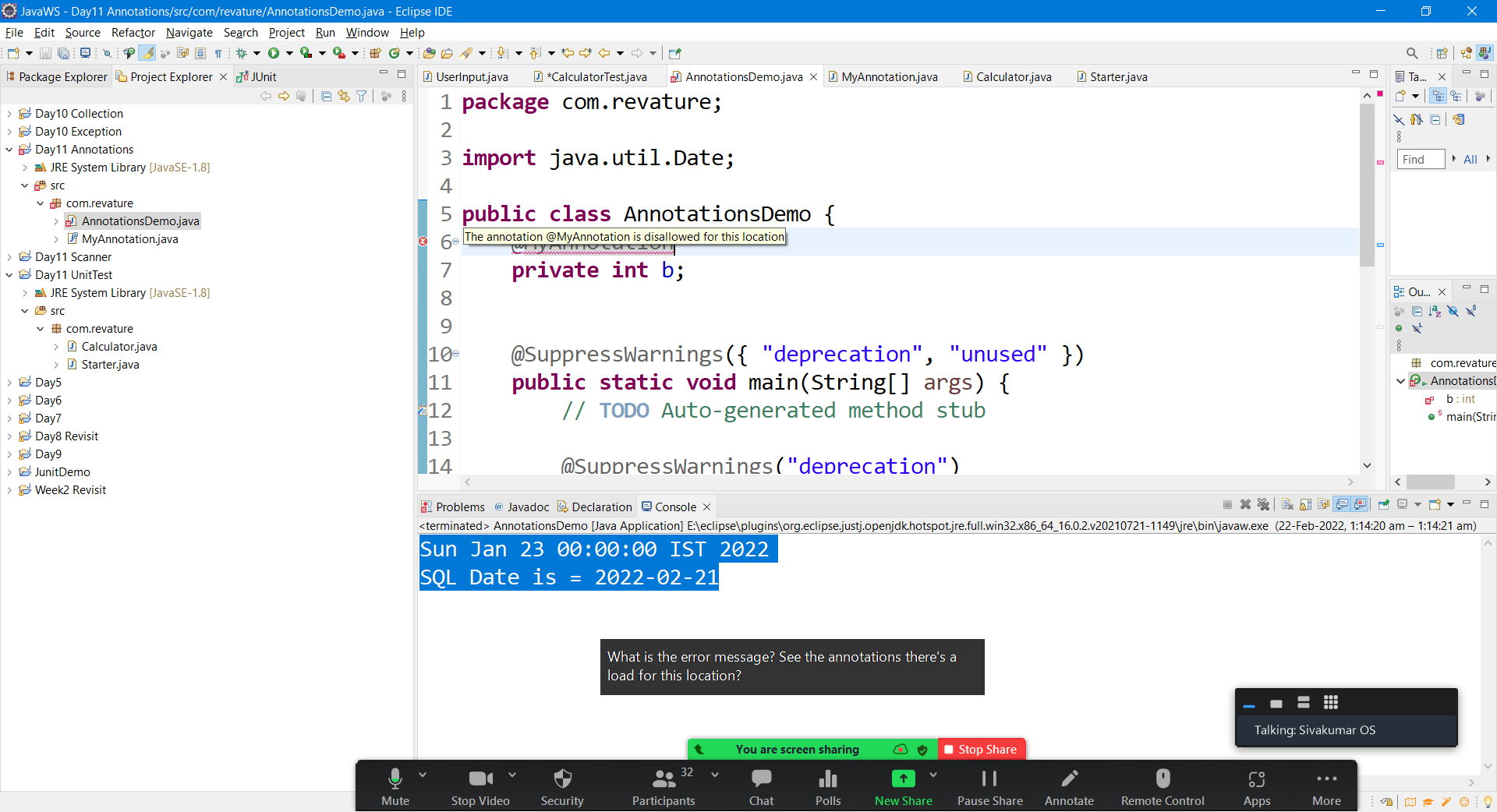
import java.lang.annotation.Target;

@Retention(CLASS)

@Target({ CONSTRUCTOR, LOCAL\_VARIABLE })

public @interface MyAnnotation {

}



Collections can’t use primitive data type.

In Java, primitives will be converted into corresponding objects.

The process of converting primitive to it’s corresponding Object is called Auto-Boxing.

With the help of Wrapper classes, primitive data types can be converted in to objects.

|  |  |  |
| --- | --- | --- |
| Sl No | Primitive Data type | Wrapper Class |
| 1 | boolean | Boolean |
| 2 | byte ( 8 bits) | Byte |
| 3 | short (16 bits) | Short |
| 4 | int (32 bits) | Integer |
| 5 | char (16 bits) | Character |
| 6 | float (32 bits) | Float |
| 7 | double | Double |
| 8 | Long | Long |

In wrapper class there are few methods that will help you to extract the data

Integer.parseInt();