**23CSE111**

**OBJECT ORIENTED PROGRAMMING**

**LAB REPORT**

****

**Department of Computer Science Engineering**

**Amrita School of Computing**

**Amrita Vishwa Vidyapeetham, Amaravati Campus**

**Name: K. NANDINI**

**Verified By Roll No: AV.SC.U4CSE24205**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.NO | TITLE | DATE | PAGE.NO. | SIGNATURE |
| 1. | WEEK-1 |  |  |  |
| i) | Downloading and install java software |  |  |  |
| ii) | Write a Java program to print the message “Hello World.” |  |  |  |
| iii) | Write a Java Program that prints Name, Roll No, Section of a student |  |  |  |
| 2. | WEEK-2 |  |  |  |
| i) | Write a java program to find the simple interest where all the inputs are given by the user. |  |  |  |
| ii) | Write a java program to find the factorial of a number where all the inputs are given by the user. |  |  |  |
| iii) | Write a java program to convert the temperature from Celsius to Fahrenheit and Celsius to Fahrenheit. |  |  |  |
| iv) | Write a java program to find the Fibonacci series of a given number where all the inputs are taken from the user. |  |  |  |
| v) | Write a java program to find the area of a rectangle where the inputs are given by the user. |  |  |  |
| vi) | Write a java program to find the area of a triangle where the inputs are given by the user |  |  |  |
| 3. | WEEK-3 |  |  |  |
| i) | Create a class car. Create four attributes named car\_color, car\_brand, fuel\_type, mileage. Create three methods start() stop() service().Create 3 objects named car1 car2 car3 |  |  |  |
| ii) | Create a class bank account with method deposit() and withdrawal(). |  |  |  |
| 4. | WEEK-4 |  |  |  |
| i) | Write a java program with class named book. The class should contain various attributes such as title, author, year of publication. It should also contain a constructor with parameters which initializes title, author, Year of publication. Create a method which displays the details of book. Display the details of two books. |  |  |  |
| ii) | Create a java program with class name Myclass with a starting variable count of int type, initialized to zero and a constant variable “pi” of type double initialized to 3.14 as attributes of that class. Define a constructor for “myclass” that increments the count variable each time an object of myclass is created. Finally print the final values of count and pi variables. Create three objects |  |  |  |
| 5. | WEEK-5 |  |  |  |
| i) | Create a calculator using the operations including addition, subtraction, multiplication and division using Multilevel Inheritance and display the desired output**.** |  |  |  |
| ii) | A vehicle rental company wants to develop a system that maintains information about different types of vehicles available for rent. The company rents out cars and bikes and they need a program to store details about each vehicle such as brand and speed   * Cars should have an additional property: number of doors * Bike should have a property indicating whether they have gears or not * The system should also include a function to display details about each vehicle and indicate when a vehicle is starting   Every class should have a constructor   1. Which OOP concept is used in the above program? Explain why it is useful in this scenario. 2. If the company decides to add a new type of vehicle truck, how would you modify the program? Truck should include an additional property capacity(in tons). Create a showTruckDetails() method to display the truck’s capacity. Write a constructor for truck that initializes all properties. 3. Implement the truck class and update the main method to create a truck object, also create an object for car and bike subclassed. Finally display its details. |  |  |  |
| 6. | WEEK-6 |  |  |  |
| i) | Write a java program to create a Vehicle class with a method displayInfo(). Override this method in the Car subclass to provide specific information about a car |  |  |  |
| ii) | A college is developing an automated admission system that verifies students eligibility for undergraduate(UG) and postgraduate(PG) programs. Each program has different eligibility criteria based on the student's percentage in their previous qualification |  |  |  |
| iii) | Create a calculator with overloaded methods to perform addition:  i) Add two integers  ii) Add two doubles  iii) Add three integers |  |  |  |
| iv) | Create a Shape class with a method calculateArea() that is overloaded for different shapes (e.g., square, rectangle).  Then create a subclass Circle that overrides the calculateArea() method for a circle. |  |  |  |
| 7. | WEEK-7 |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**WEEK-1**

**PROGRAM-1**

## AIM: Downloading and installing Java(JDK 21)

PROCEDURE:

This is the process for installation of JDK on windows.

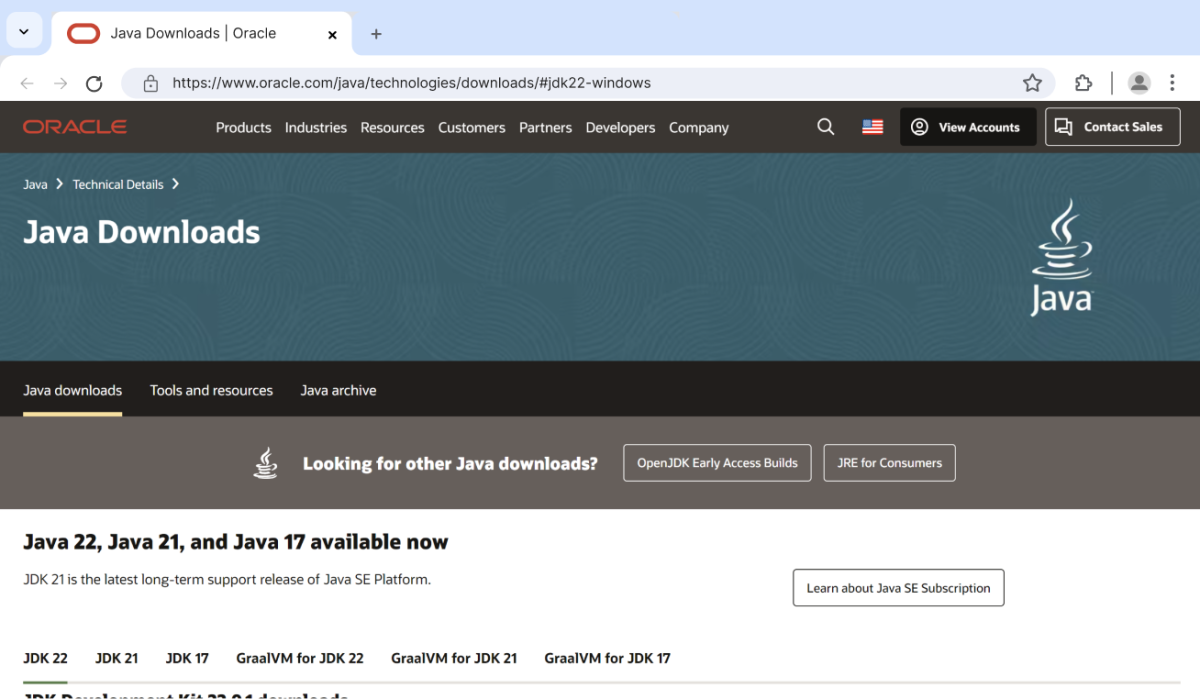
Follow the steps below to install Java on Windows:

* Download JDK(Java Development Kit)
* Run the Installer
* Configure Environment Variables
* Update the path variable
* Verify Installation in Command Prompt

Here's a detailed explanation of each of the steps.

### Step 1: Download JDK

1. Go to the official oracle website in the google search to download the JDK.
2. Locate the downloaded jdk-21\_windows-x64\_bin.exe file.
3. Double-click to launch the installer.
4. Click Next on the setup wizard.
5. Choose the installation path (default is C:\Program Files\Java\jdk-21).
6. Click Next, then click Install.
7. Wait for the installation to complete.
8. Click Close once the installation is finished
9. Choose **x64 MSI Installer** on the windows tab and click on download link.



Step 2: Run the Installer

Now, go to your **downloads** folder and run the installer you just downloaded.

The screen below will be seen.



Simply click **Next** to proceed.Next you will be prompted a another screen simply click next on that also.

Step 3: Configure Environment Variables

After installation, you will need to tell your system where to find Java. This is done by setting environment variables.

These are the ways to follow:

1. Go to file manager on your laptop or pc

2) Go to “Windows C” Drive in File manager

3) Choose Program Files, select Java, then JDK 22, then select Bin.

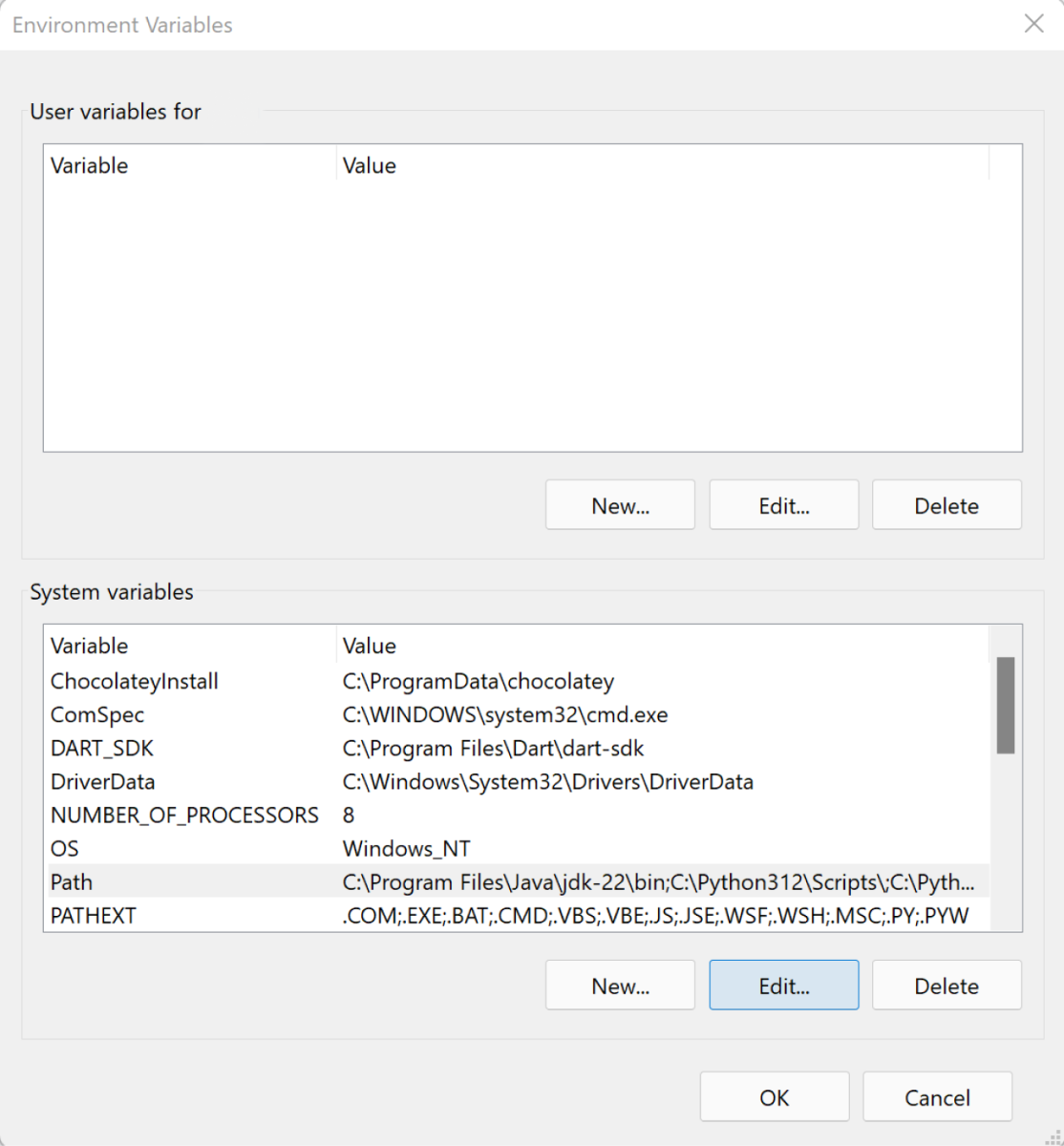
4) Select and copy the path at the address bar

**Locate JDK Path**: Navigate through your file explorer to reach the JDK installation directory. Normally, it is located at

C:\Program Files\Java\jdk-22\bin

Copy this path

**Access Environment Variables**: Search **environment variable** on the terminal. In system properties, click on environment variables. You will be prompted to the screen below.



**Step 4: Update the Path Variable:**

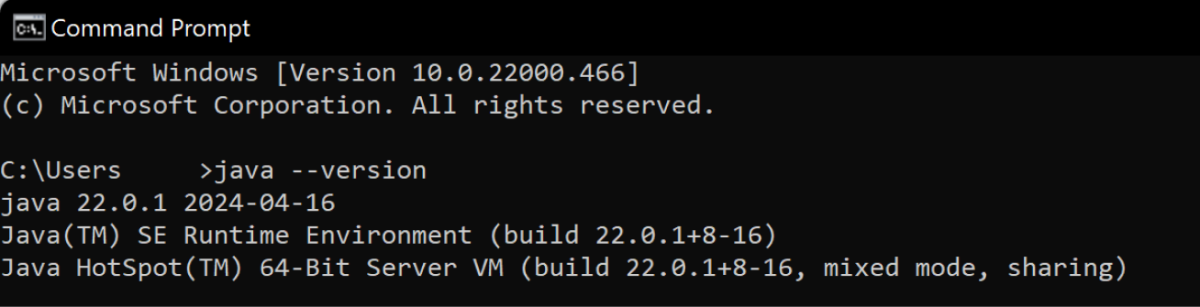
Find the **Path** variable in the System variables section and click on **Edit**.

Then, click **New** and paste your JDK bin path (i.e. C:\Program Files\Java\jdk-22\bin).

Finally, click **Ok** to close each window.

### Step 5: Verify your Installation

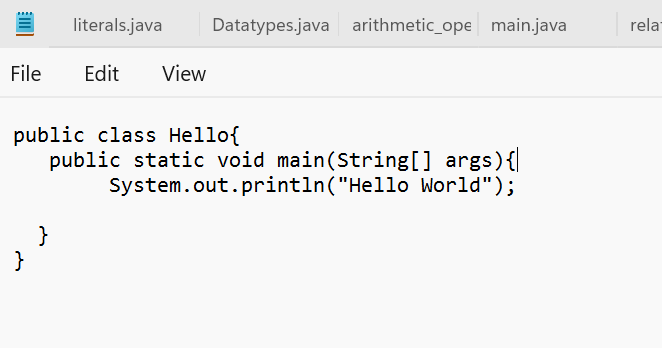
After the installation, you can verify whether Java is installed by using the following command in the command prompt.

.

If Java is installed successfully, it will print the version information; otherwise, it will produce an error message indicating that the command is not recognized.

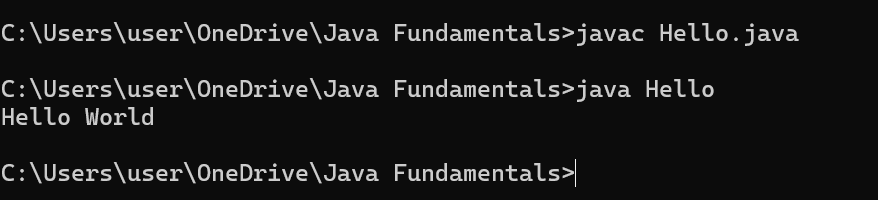
**PROGRAM-2:**

**AIM:** Write a Java program to print the message “Hello World”.

**Code:** 

**OUTPUT:**

**POSITIVE CASE:**

****

**NEGATIVE CASE:**

****

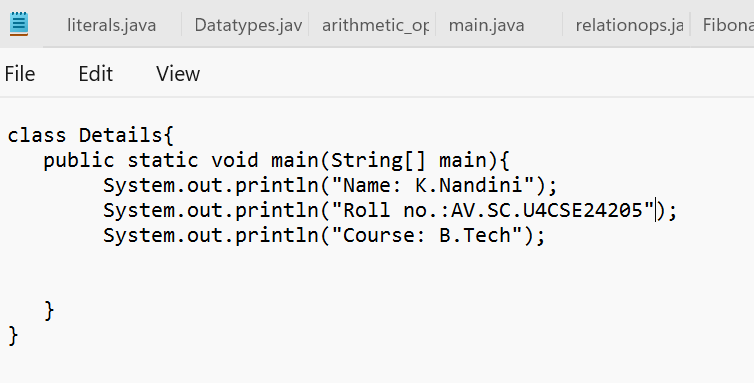
**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | ERRORS | RECTIFICATION |
| 1 | Double codes at the end of print statement is not given | The error is rectified by giving double codes in print statement |

**PROGRAM-3:**

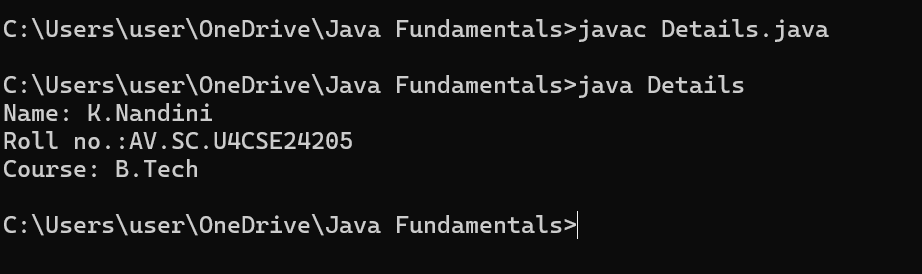
**AIM:** Write a Java Program that prints Name, Roll No, Section of a student.

**CODE:**



**OUTPUT:**

**POSITIVE CASE:**

****

**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | ERROR | RECTIFICATION |
| 1 | In the statement at the end ; is not mentioned | Rectified by keeping ; at the end of the ststement |

**IMPORTANT POINTS:**

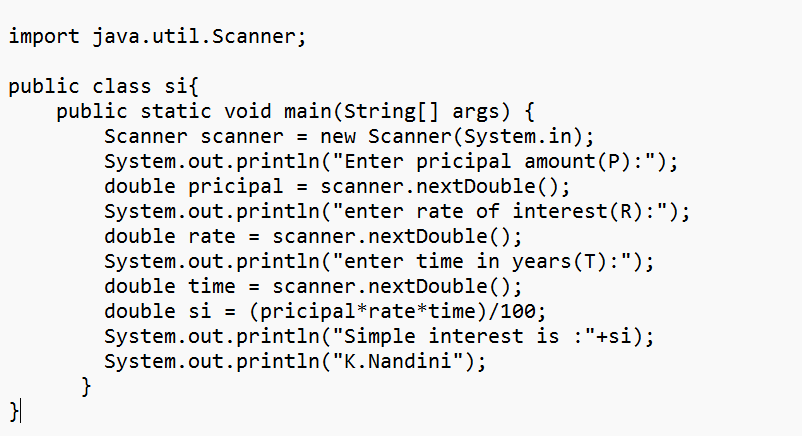
System.out.println(“ “); - It is used to print string inside the quotes. After printing, the cursor moves to the beginning of the next line.

**WEEK – 2**

**PROGRAM-1:**

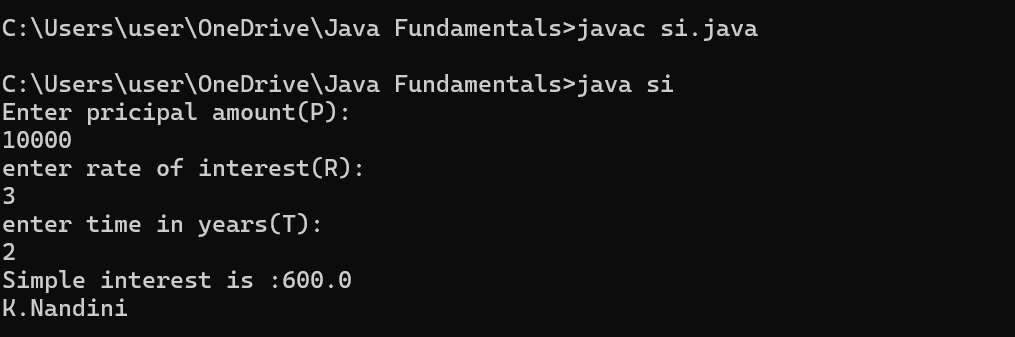
**AIM**: Write a java program to find the simple interest where all the inputs are taken from the user.

**CODE:**

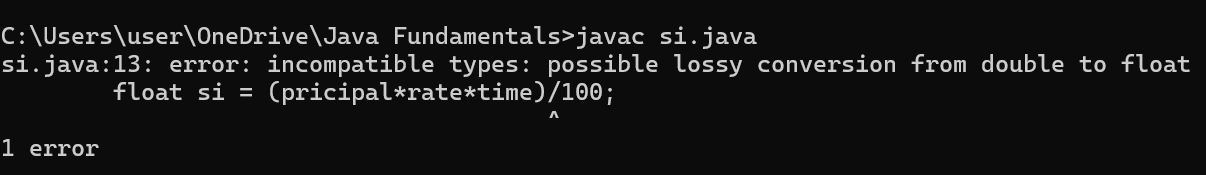


**OUTPUT:**

**POSITIVE CASE**:



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | ERROR | RECTIFICATION |
| 1 | In the statement the type conversion from double to float is impossible | Rectified by keeping double inplace of float variable |

**IMPORTANT POINTS:**

1. import java.util.Scanner; - To accept input from user.

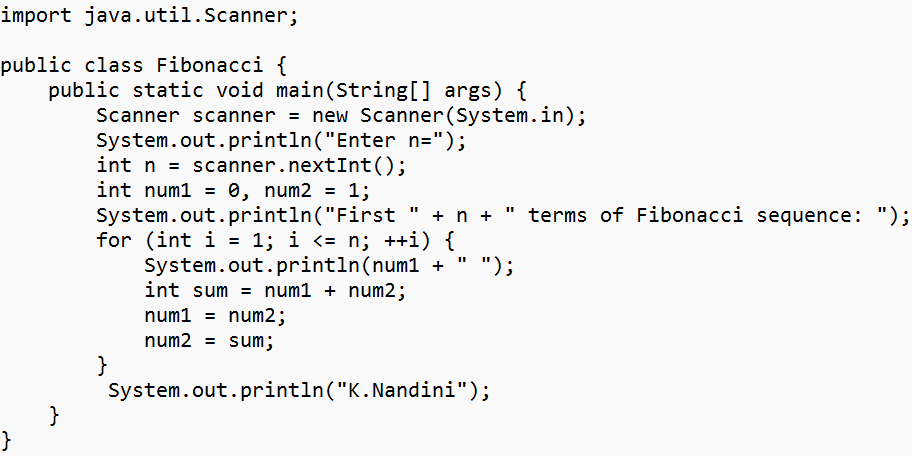
2. Scanner scanner=new Scanner(System.in); - Used to create a Scanner object

3. double=scanner.nextDouble(); - Used to read the integer data type stored under the object created

**PROGRAM-2:**

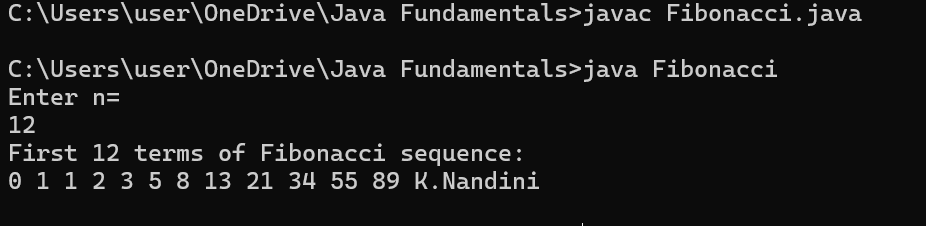
**AIM:** Write a java program to find the Fibonacci sequence of a given number

**CODE**:

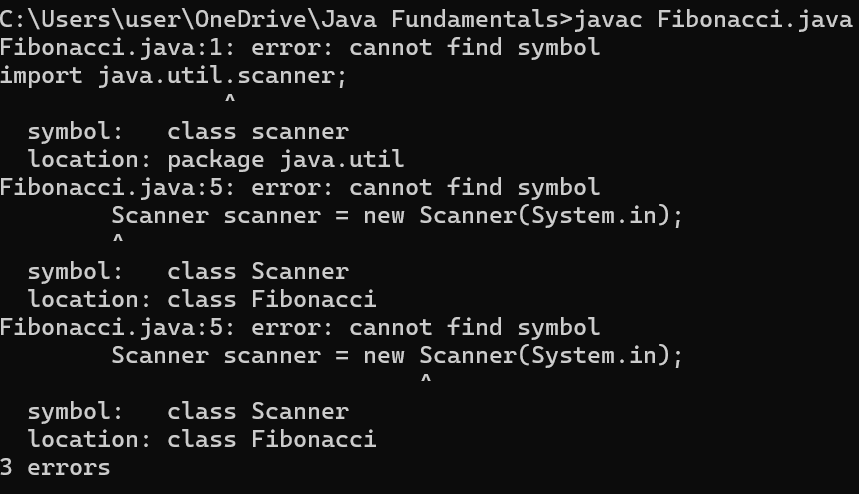


**OUTPUT:**

**POSITIVE CASE**:



**NEGATIVE CASE:**



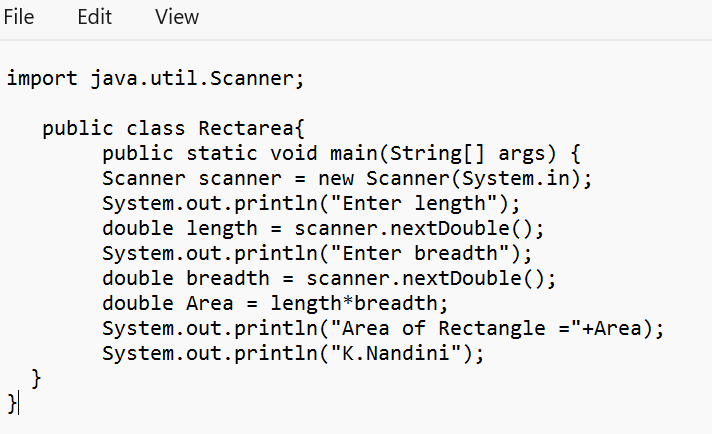
**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | ERROR | RECTIFICATION |
| 1 | While importing scanner from java library s in scanner is given in lowercase letter | Rectified by keeping Uppercase letter S while importing scanner |

**PROGRAM-3:**

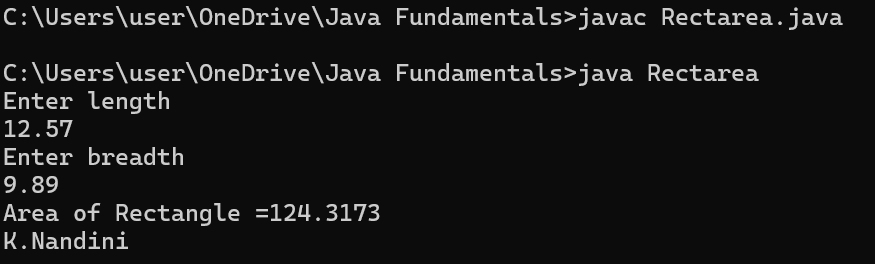
**AIM**: Write a java program to find the area of rectangle and triangle.

**CODE:**

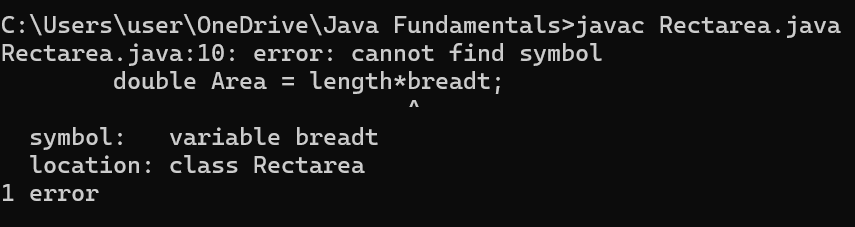


**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**

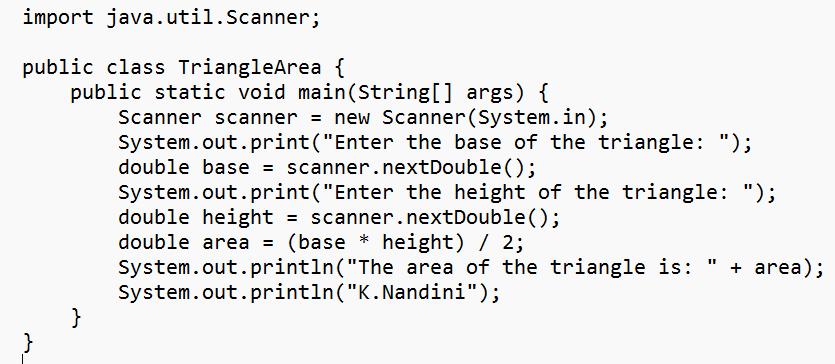


**ERROR TABLE:**

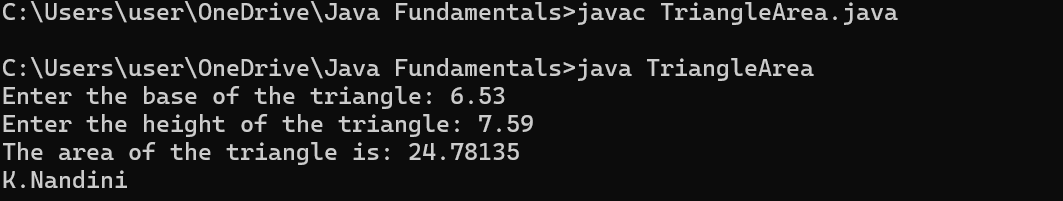
|  |  |  |
| --- | --- | --- |
| S.No | ERROR | RECTIFICATION |
| 1 | The variable name breadth has spelling wrong | Rectified by correcting the spelling |

**Finding Triangle Area**

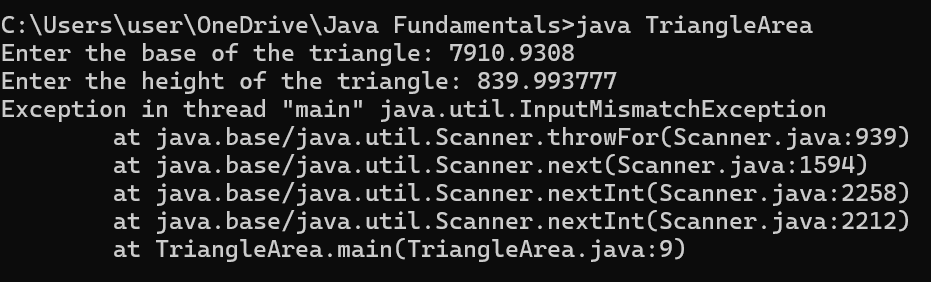
**CODE:**



**OUTPUT:**

**POSITIVE CASE:**  


**NEGATIVE CASE:**

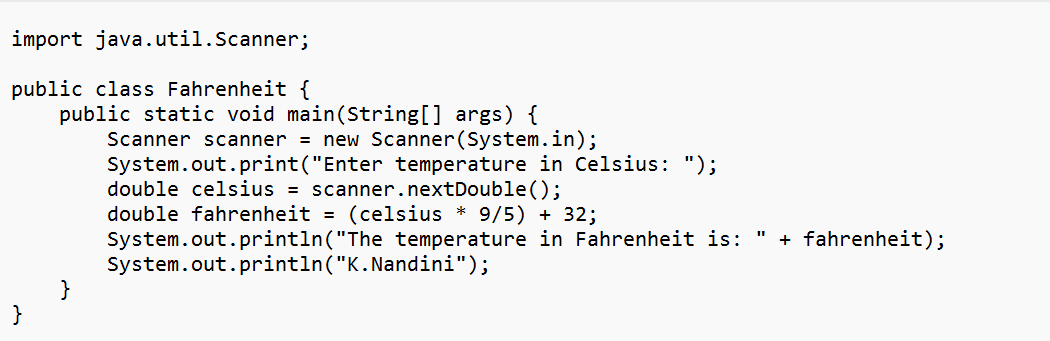


**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | ERROR | RECTIFICATION |
| 1 | In the statement the variable datatype is given as double but while taking the input in scanner it was given next.Int | Rectified by keeping next.Double as given datatype |

**PROGRAM-4:**  
**AIM:** Write a java code to convert the temperature from Celsius to Fahrenheit and from Fahrenheit to Celsius.

**CODE:**

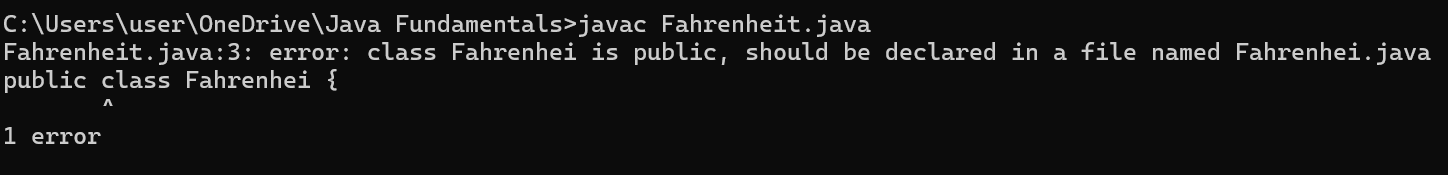


**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**

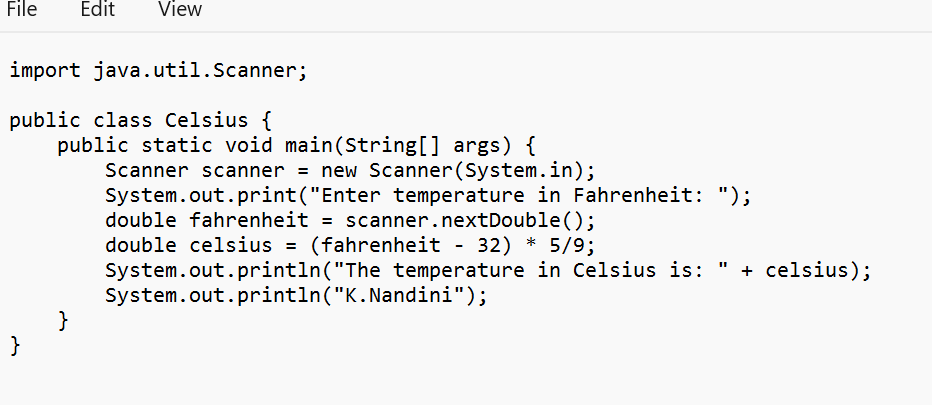


**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | ERROR | RECTIFICATION |
| 1 | The class name written in the file and the class name given as the file name are not same. | Rectified the error by giving same class names both in the file and the file name |

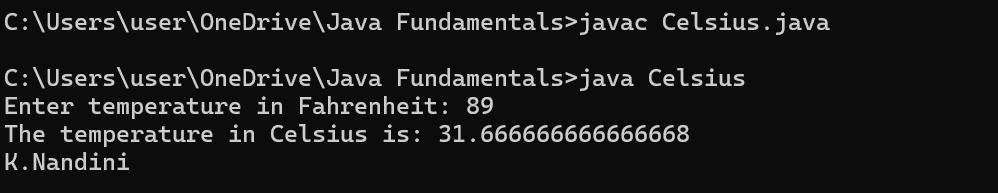
**Fahrenheit to Celcius**

**CODE:**

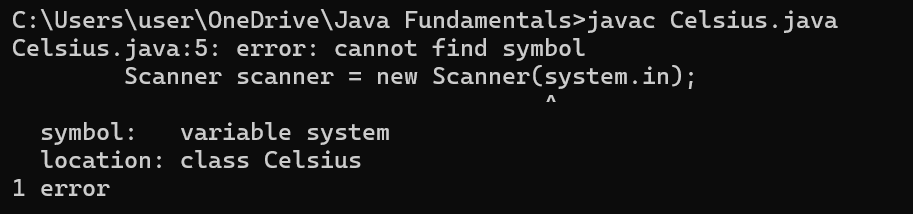


**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**



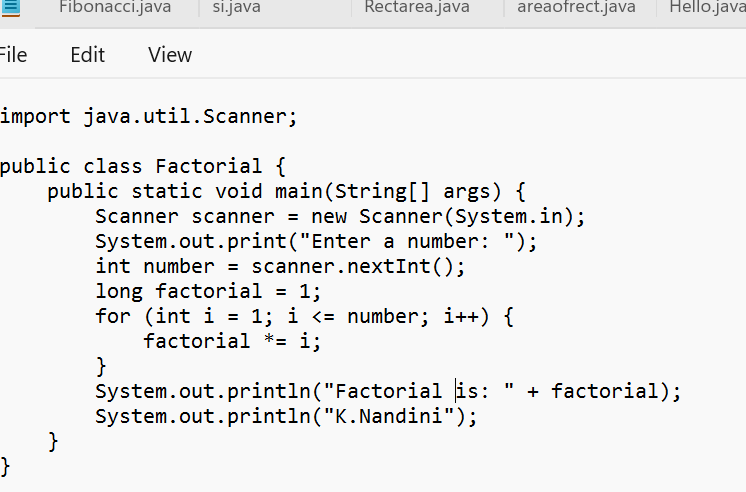
**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | ERROR | RECTIFICATION |
| 1 | In the statement for reading the inputs from the user the s in system.in is given in lowercase letter | Rectified by placing uppercase letter S in place of lowercase letter in System.in  In scanner |

**PROGRAM-5**

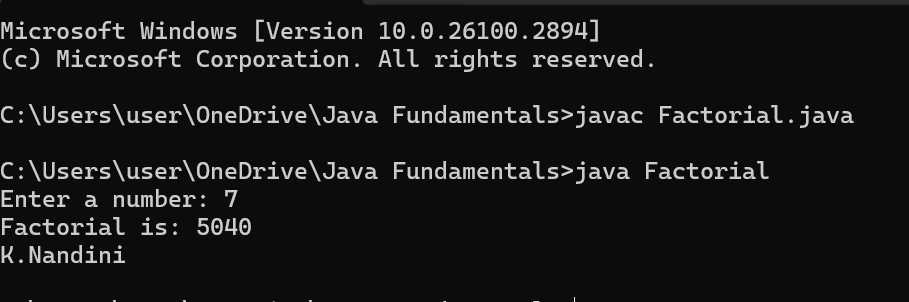
**AIM**: Write a java code to find factorial of a number by taking input.

**CODE:**

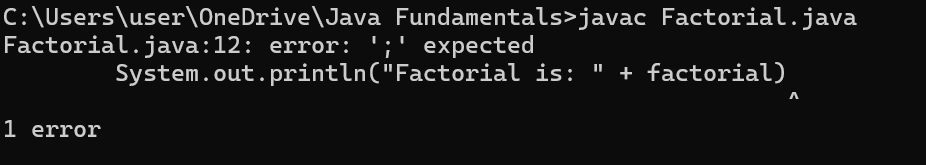


**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | The datatype for factorial is given float which may not be appropriate for getting the required output of the factorial of a given number | Rectifying the datatype as long so that any long integer can be accepted |
| 2 | Semicolon after System.out.println is not given | The error is rectified by keeping the semicolon |

**IMPORTANT POINTS:**

1.import java.util.Scanner; - To accept input from user, Scanner class under util package has to be imported.

2. Scanner input=new Scanner(System.in); - Used to create a Scanner object

3. int ln=input.nextInt(); - Used to read the integer data type stored under the object created

**WEEK-3**

**PROGRAM-1**

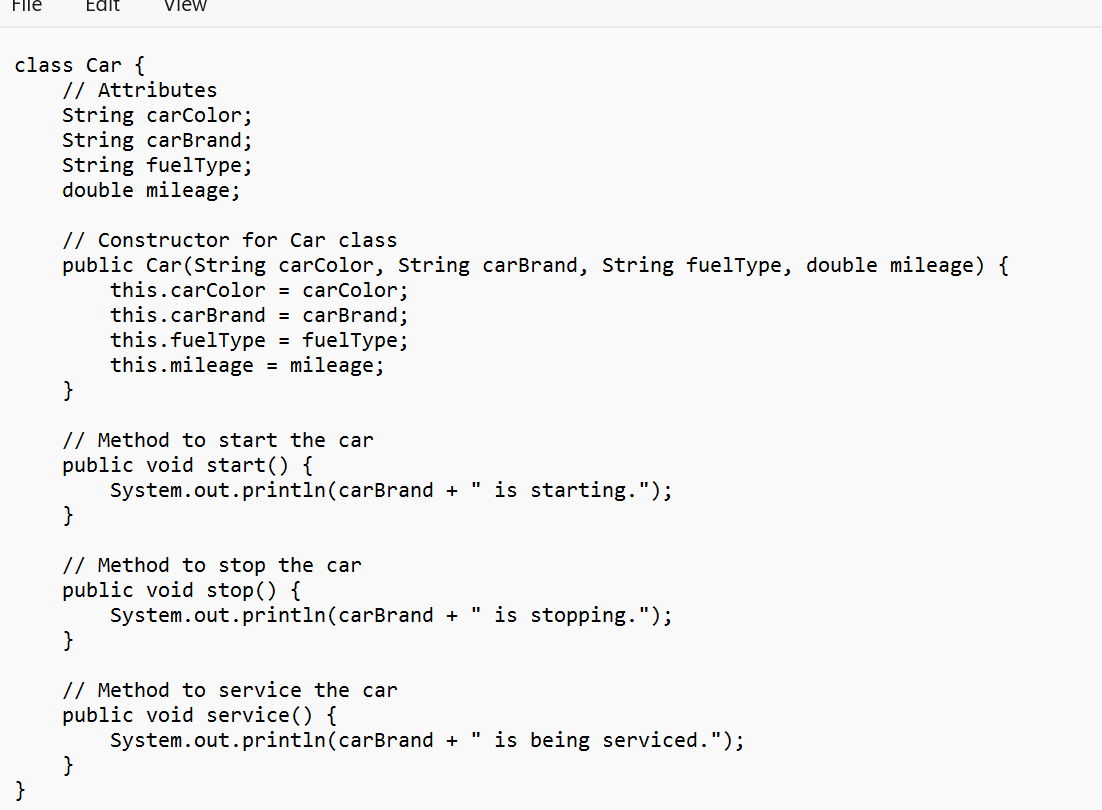
**AIM**: Create java program with following instructions

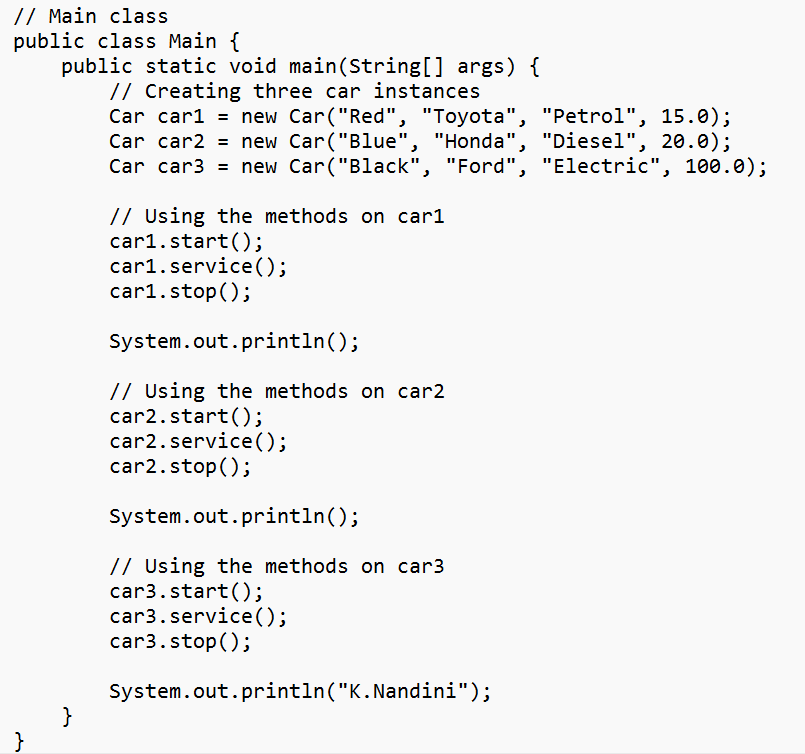
1. Create a class with name Car.
2. Create four attributes named Car\_color, Car\_brand, fuel\_type, mileage.
3. Create three methods named start(), stop(), service().
4. Create three objects named Car1, Car2, Car3.

**CLASS DIAGRAM**:

|  |
| --- |
| Car |
| + Car\_color: String  + Car\_brand: String  - fuel\_type: String  - mileage: double |
| + start(): void  + stop(): void  - service(): void  + Car1(): void  + Car2(): void  + Car3(): void |

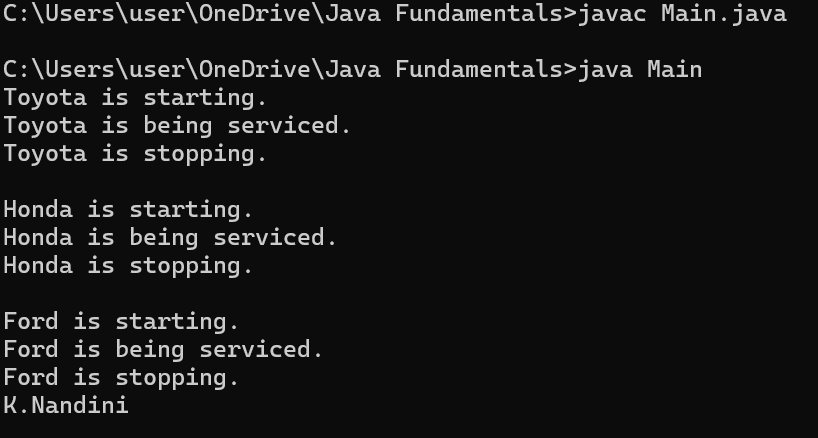
**CODE**



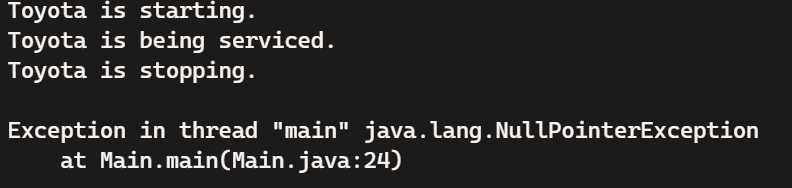


**OUTPUT**:

**POSITIVE CASE**:

****

**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | Calling Null Methods which are not defined leads to Exception | Rectifying the Expection by giving correct method names |
| 2 | Semicolon after System.out.println is not given | The error is rectified by keeping the semicolon |

**Important Points:**

* Classes and Objects:
  + Class: A blueprint for creating objects (a particular data structure), containing methods and variables.
  + Object: An instance of a class.
* Attributes:
  + Variables within a class that represent the state or properties of an object.
* Constructor:
  + A special method used to initialize objects. It is called when an object of a class is created.
* Methods:
  + Functions defined within a class that represent the behavior or actions that an object can perform.

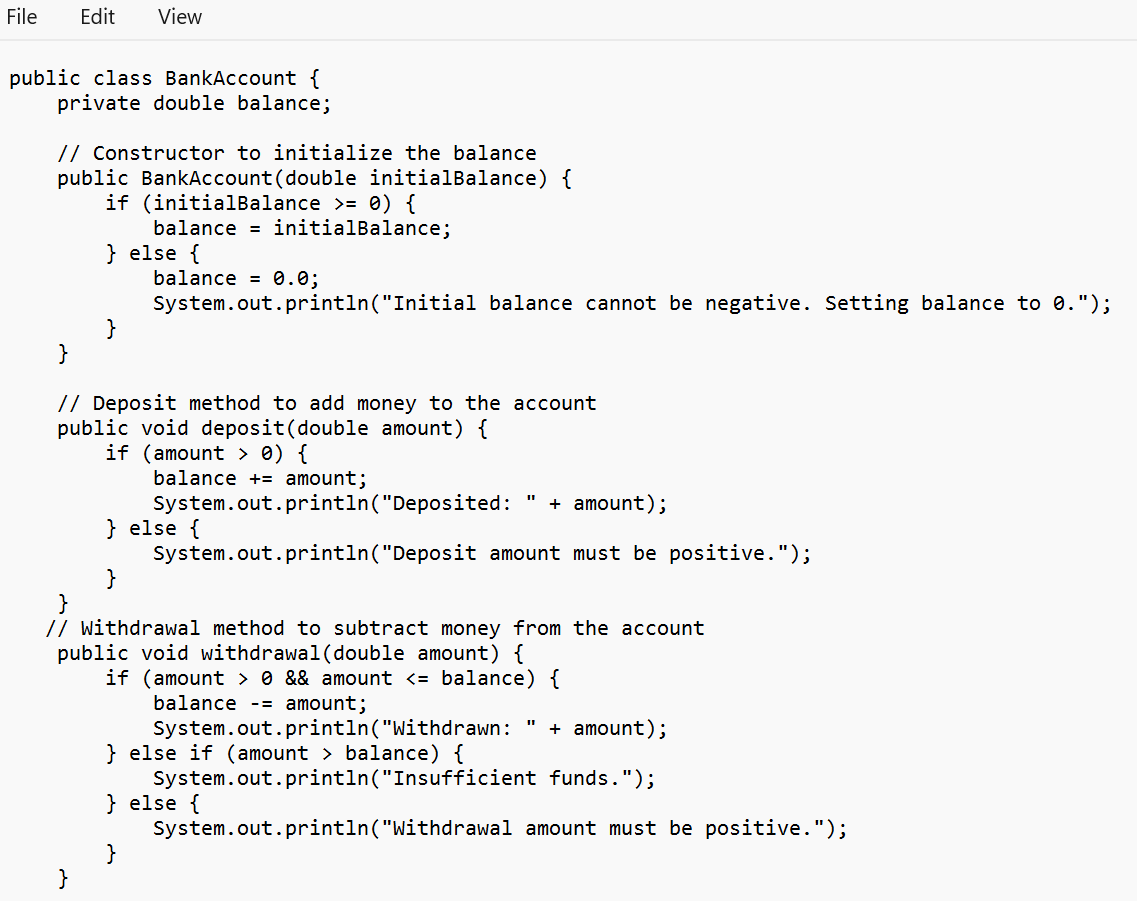
**PROGRAM-2:**

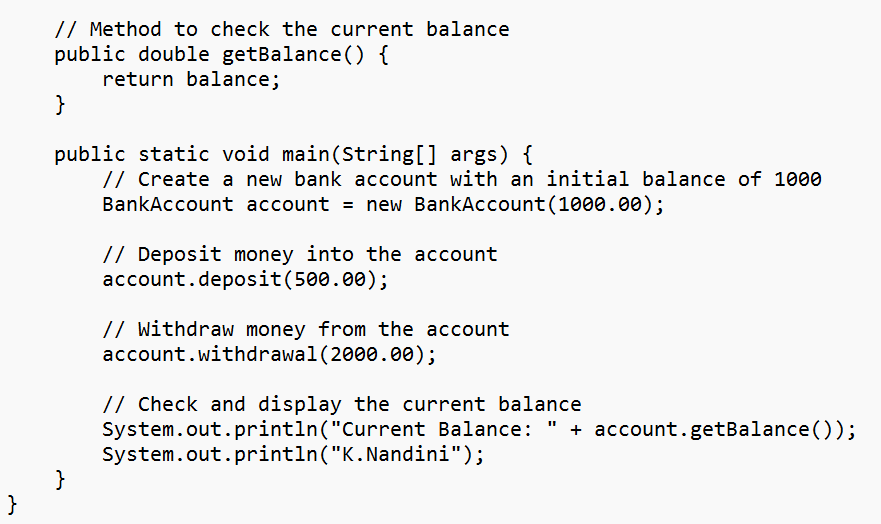
**AIM**: Create a class bank account with method deposit() and withdrawal().

**CLASS DIAGRAM:**

|  |
| --- |
| BankAccount |
| - balance: double |
| + BankAccount(initialBalance: double)  + deposit(amount: double): void  + withdrawal(amount: double): void  + getBalance(): double |

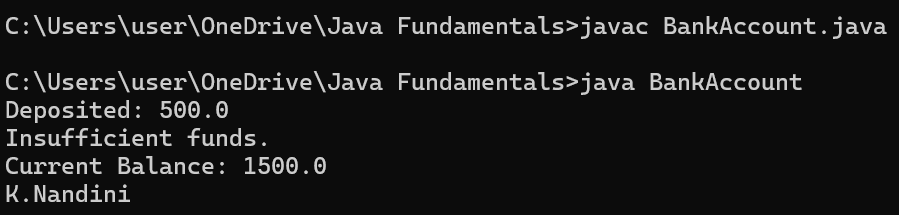
**CODE**:



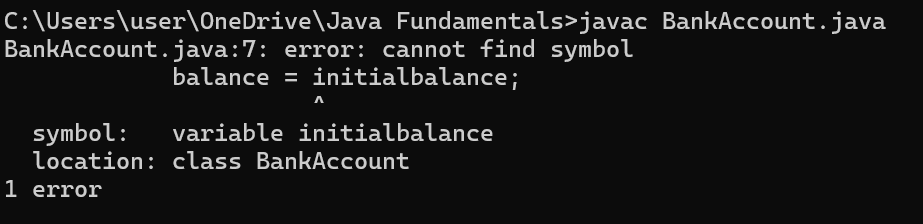


**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | Giving Wrong variable name | Rectifying the error by giving correct variable name |

**IMPORTANT POINTS:**

Encapsulation:

* Using private fields and public methods to protect the data and control access.

**Conditionals**:

* Using if statements to enforce rules and conditions

**WEEK-4**

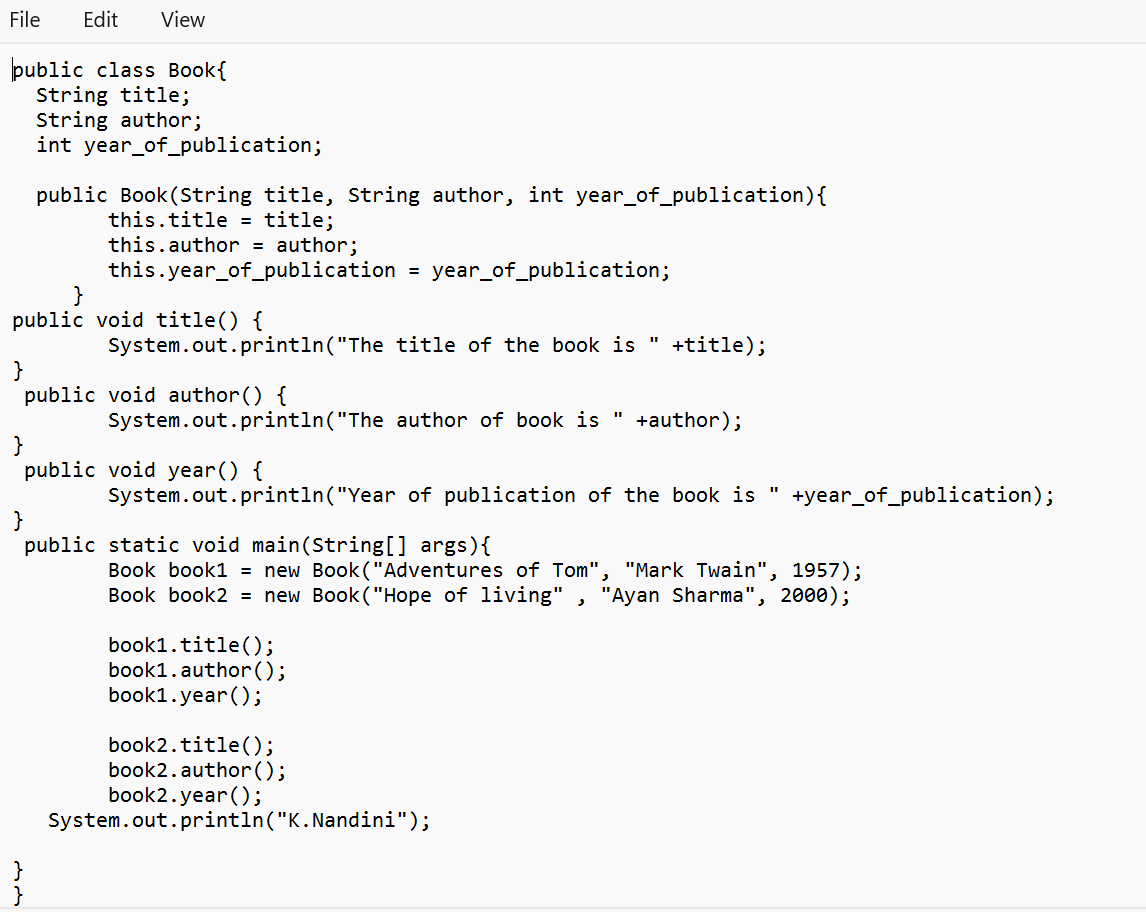
**PROGRAM-1**

**AIM:** Write a java program with class named book. The class should contain various attributes such as title, author, year of publication. It should also contain a constructor with parameters which initializes title, author and year of publication. Create a method which displays the details of the book. Display the details of two books.

**CLASS DIAGRAM:**

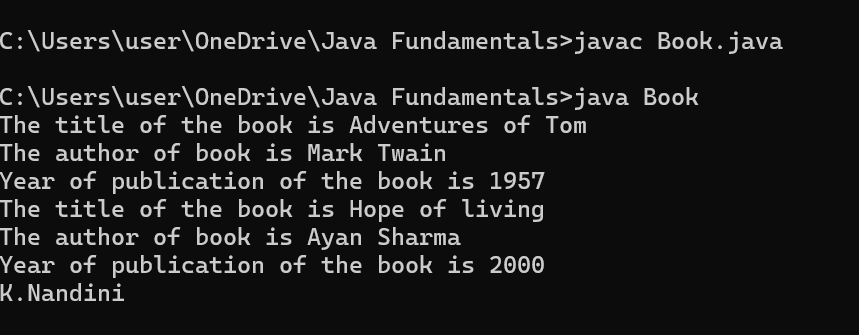
|  |
| --- |
| Book |
| - title: String  - author: String  - year\_of\_publication: int |
| + Book(String, String, int)  + title(): void  + author(): void  + year(): void  + main(String[]): static void |

**CODE:**

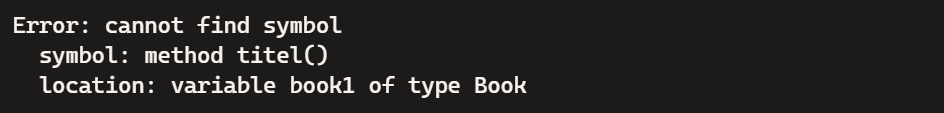
****

**OUTPUT**

**POSITIVE CASE:**



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | Calling Non-Existent Methods  Ex: By typing titel instead of title while calling the method | Rectifying the method name as title |
| 2 | Semicolon after System.out.println is not given | The error is rectified by keeping the semicolon |

**IMPORTANT POINTS:**

Using a constructor to initialize object attributes when a new object is created. Defining methods (title(), author(), year()) to perform specific tasks within the class.Using the main method as the entry point to the program.

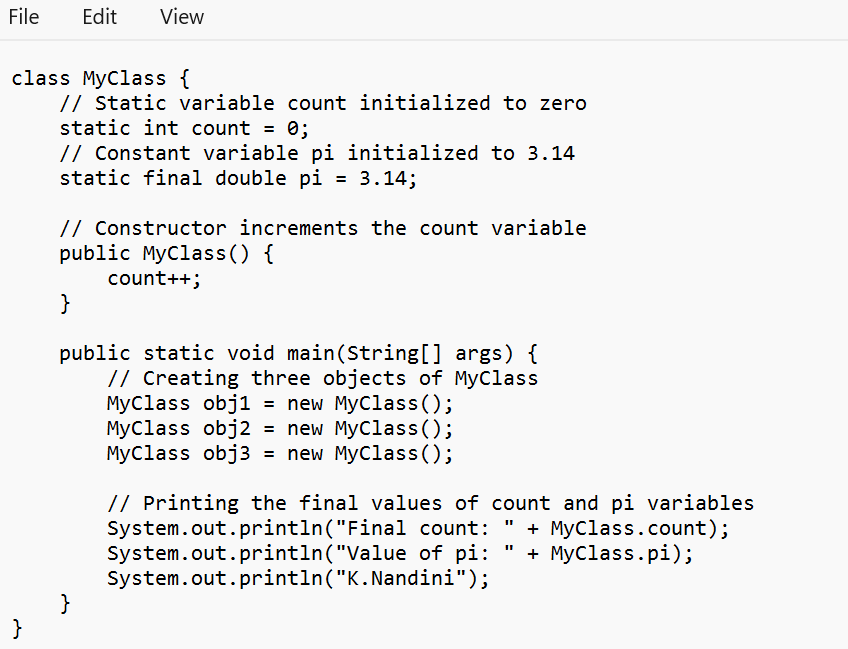
**PROGRAM-2**

**AIM:** Create a java program with class named “MyClass” with a static variable count of int type initialized to zero and a constant variable ‘pi’ of type double initialized to 3.14 as attributes of that class. Now define a constructor for “MyClass” that increments the count variable each time an object of “MyClass” is created. Finally print the final values of count and pi variables. Create three variables.

**CLASS DIAGRAM:**

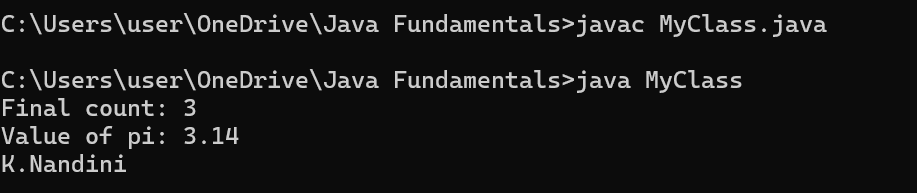
|  |
| --- |
| MyClass |
| - count: static int  - pi: final double |
| + MyClass()  + main(String[]): static void |

**CODE:**



**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | While creating the object Parenthesis are not given for MyClass() | Rectifying the error by giving the Parenthesis to MyClass |

**Important Points:**

1. Static Variables:
   * count: A static variable shared among all instances of the class. It keeps track of the number of objects created.
   * pi: A static constant variable, its value remains the same for all instances.
2. Constructor:
   * Increments the count variable each time an object is created.
3. Static Context:
   * Accessing static variables and methods using the class name (MyClass.count, MyClass.pi)

**WEEK-5**

**PROGRAM-1**

**AIM:** Create a calculator using the operations including addition, subtraction, multiplication and division using Multilevel Inheritance and display the desired output.

**CLASS DIAGRAM:**

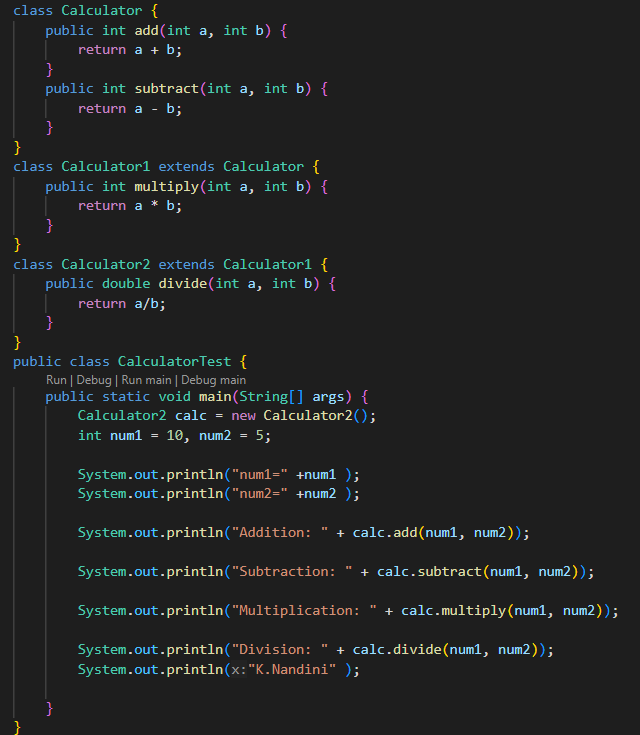
**MULTILEVEL INHERITANNCE**

|  |
| --- |
| Calculator |
| - num1: int  - num2: int |
| + add(): int  + subtract(): int |

|  |
| --- |
| Calculator1 |
| - num1: int  - num2: int |
| + multiply(): int |

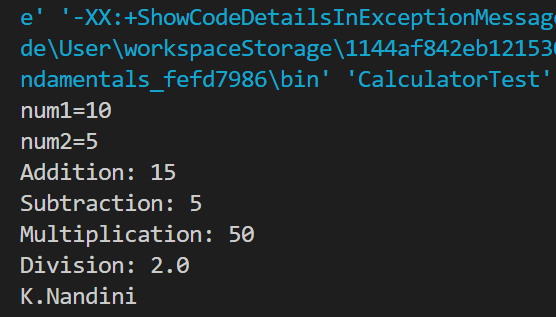
|  |
| --- |
| Calculator2 |
| - num1: int  - num2: int |
| + divide(): int |

**CODE:**

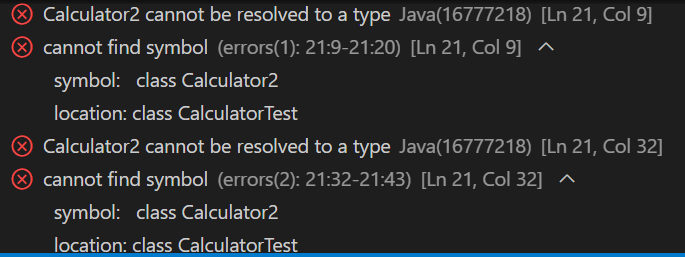


**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | Given wrong class name while object is created | Rectifying the given wrong class name |

**IMPORTANT POINTS**

1. Multilevel Inheritance:

* Multilevel inheritance occurs when a class is derived from another class, and then a third class is derived from the second class, forming a chain.

2. Code Reusability:

* The add() and subtract() methods are defined once in the Calculator class and can be reused by both the Calculator1 and Calculator2 classes without rewriting the code.
* The multiply() method is defined in Calculator1 and inherited by Calculator2
* The divide() method is added to the Calculator2 class.
* This shows how inheritance helps to reduce redundant code and increases reusability.

3. Polymorphism:

* The code demonstrates polymorphism (though not explicitly in the form of method overriding). Even though the object calc is of the class Calculator2, it can call methods from the base class (Calculator) and intermediate class (Calculator1) due to inheritance.

For example, calling calc.add(num1, num2) and calc.subtract(num1, num2) works because Calculator2 inherits from Calculator

**PROGRAM-2**

**AIM:** A vehicle rental company wants to develop a system that maintains information about different types of vehicles available for rent. The company rents out cars and bikes and they need a program to store details about each vehicle such as brand and speed

* Cars should have an additional property: number of doors
* Bike should have a property indicating whether they have gears or not
* The system should also include a function to display details about each vehicle and indicate when a vehicle is starting
* Every class should have a constructor

1. Which OOP concept is used in the above program? Explain why it is useful in this scenario.
2. If the company decides to add a new type of vehicle truck, how would you modify the program? Truck should include an additional property capacity(in tons). Create a showTruckDetails() method to display the truck’s capacity. Write a constructor for truck that initializes all properties.
3. Implement the truck class and update the main method to create a truck object, also create an object for car and bike subclassed. Finally display its details.

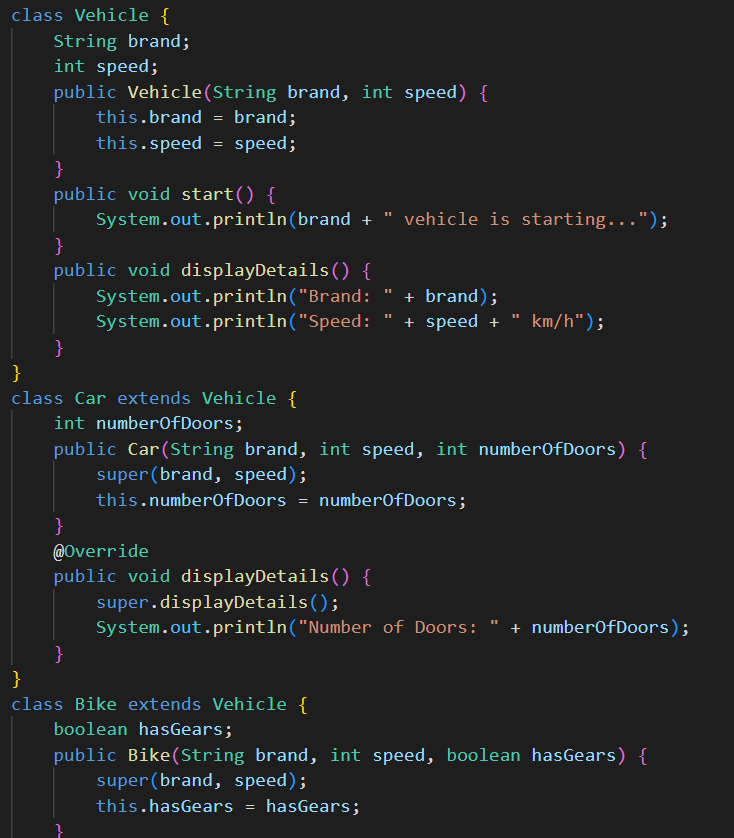
**CLASS DIAGRAM**:

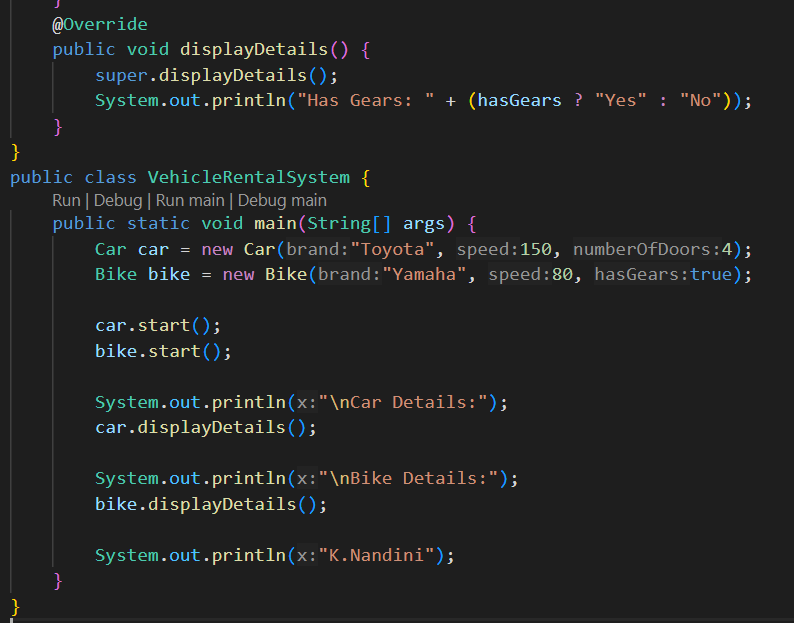
|  |
| --- |
| Vehicle |
| + brand: String  + speed: int |
| + Vehicle(brand: String, speed: int)  + start(): void  + displayDetails(): void |

|  |
| --- |
| Car |
| +numberOfDoors: int |
| +Car(brand:String, speed: int, numberOfDoors: int)  + displayDetails(): void |

|  |
| --- |
| Bike |
| + hasGears: boolean |
| + Bike(brand: String, speed: int, hasGears: boolean) +displayDetails(): void |

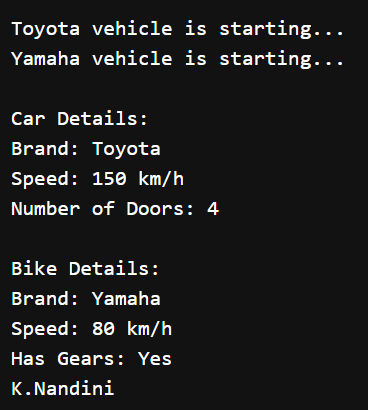
**CODE:**

****

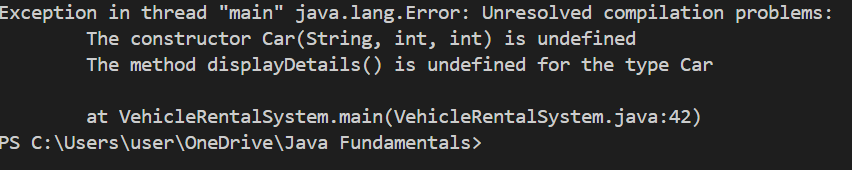


**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | The constructor of Car class is not defined. This gives an Exception | Rectifying the Exception by defining the constructor to Car class |

ANSWERS FOR THE GIVEN QUESTIONS

**1.**The OOP (Object-Oriented Programming) concepts used in the program are:

**Inheritance**:

* 1. Inheritance is a key concept here, as the Car, Bike, and Truck classes will all extend a common base class, Vehicle
  2. Usefulness: Inheritance allows common properties (like brand and speed) to be defined in the Vehicle class and inherited by the Car, Bike, and Truck classes. This avoids code duplication and allows new vehicle types to easily be added in the future without having to repeat common code.

**Encapsulation**:

* 1. Encapsulation is achieved by bundling the data (attributes such as brand, speed, etc.) and the behavior (methods like start(), displayDetails()) together into a single unit (the Vehicle class and its subclasses).
  2. Usefulness: It helps to hide internal details and protect the integrity of data by only exposing the necessary methods for interaction.

**Polymorphism**:

* 1. Polymorphism allows us to treat objects of different classes (Car, Bike, Truck) uniformly as objects of type Vehicle. Each subclass can override or extend methods like displayDetails() to provide its own behavior.
  2. Usefulness: This enables the program to handle different vehicle types generically while allowing specific behavior for each vehicle

**2 & 3)**

**CLASS DIAGRAM**:

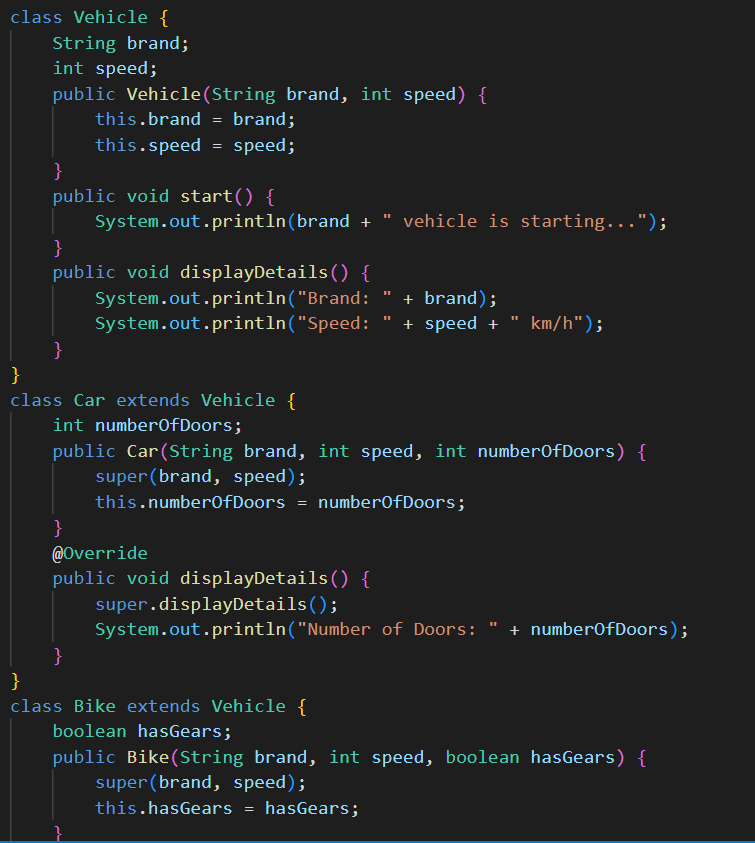
|  |
| --- |
| Vehicle |
| + brand: String  + speed: int |
| + Vehicle(brand: String, speed: int)  + start(): void  + displayDetails(): void |

|  |
| --- |
| Car |
| +numberOfDoors: int |
| +Car(brand:String, speed: int, numberOfDoors: int)  + displayDetails(): void |

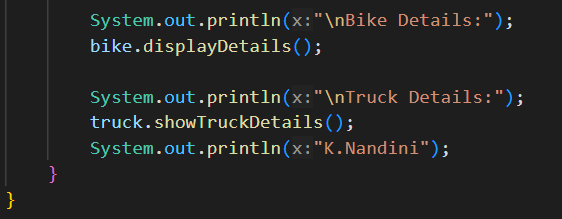
|  |
| --- |
| Bike |
| + hasGears: boolean |
| + Bike(brand: String, speed: int, hasGears: boolean) +displayDetails(): void |

|  |
| --- |
| Truck |
| + Capacity : double |
| +Truck(brand: String, speed: int, Capacity: double)  +showTruckDetails(): void |

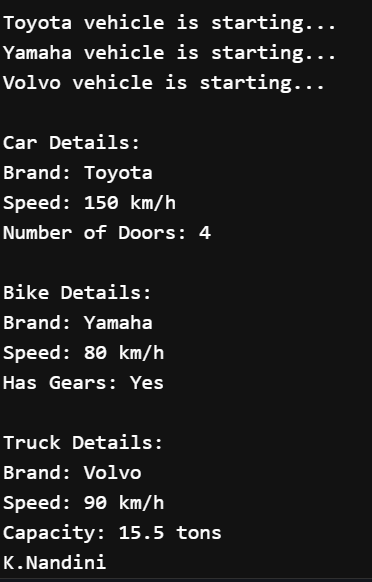
**CODE:**



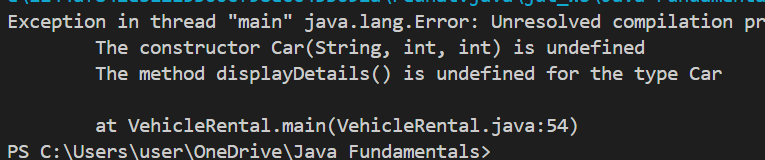




**POSITIVE CASE:**



**NEGATIVE CASE :**

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | The constructor of Car class is undefined. This causes an Exception | Rectifying the Exception by defining the Constructor for Car class |

**IMPORTANT POINTS:**

1. Class Inheritance:

* Vehicle Class: The Vehicle class serves as the base class. It contains common properties (like brand and speed) and methods (start() and displayDetails()) that are shared by all vehicle types.
* Car, Bike, and Truck Classes: These are subclasses that extend the Vehicle class. They inherit the properties and methods of the Vehicle class but can add their own specific properties (like numberOfDoors for Car, hasGears for Bike, and capacity for Truck).

2. Constructor Inheritance:

* The Car, Bike, and Truck classes use the super() keyword to invoke the constructor of the Vehicle class, ensuring the shared properties (brand and speed) are initialized properly. Each subclass has its own specific constructor to initialize its additional properties.

3. Method Overriding:

* displayDetails() Method: Each subclass (Car, Bike, Truck) overrides the displayDetails() method from the Vehicle class to add more specific details about the subclass. The super.displayDetails() is used to first display the general vehicle information, then additional subclass-specific details are printed.

**WEEK-6**

**PROGRAM-1**

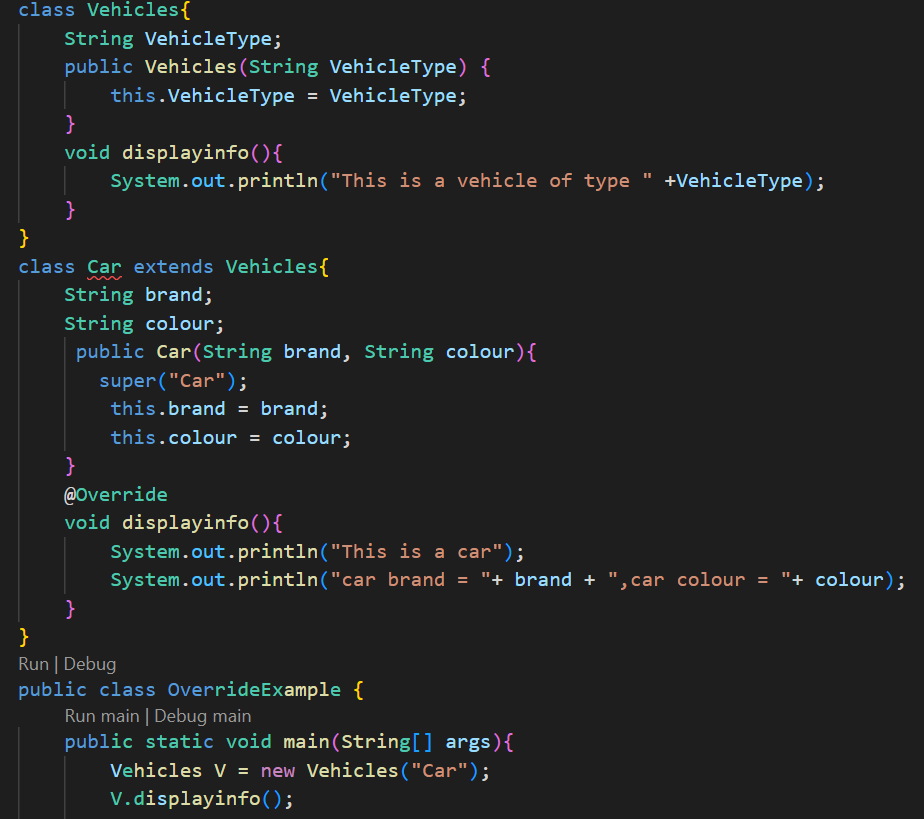
**AIM:** Write a java program to create a Vehicle class with a method displayInfo(). Override this method in Car subclass to provide specific information about a car.

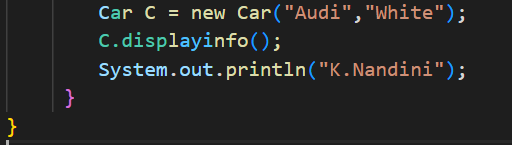
**CLASS DIAGRAM:**

|  |
| --- |
| Vehicles |
| + VehicleType: String |
| + Vehicle(VehicleType: String)  + displayinfo(): void |

|  |
| --- |
| Car |
| + Brand: String  +Colour:String |
| + Car(Brand: String,Colour: String)  + displayinfo(): void |

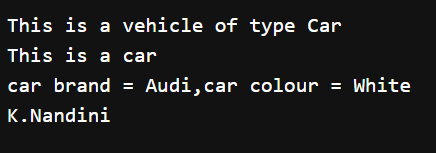
**CODE:**



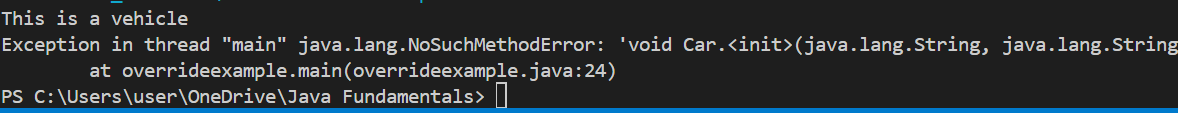
****

**OUTPUT**:

**POSITIVE CASE**:



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | While creating the object for Calculating class , given lower case letter c in class name Calculating. | Rectifying by giving uppercase letter C in the class name Calculating in object creation |

**IMPORTANT POINTS:**

Inheritance: Inheritance is the mechanism in Java where one class can inherit the fields and methods from another class. In the code, Car inherits from Vehicles.

Method Overriding: Method overriding occurs when a subclass provides its own implementation of a method that is already defined in its superclass.

The Car class overrides the displayinfo() method from Vehicles. The @Override annotation is used to indicate that the method is overriding the superclass method.

This is a form of polymorphism where the method called depends on the object type, not the reference type (i.e., whether it's a Vehicles or Car object)

**PROGRAM-2**

**AIM:** A college is developing an automated admission system that verifies students eligibility for undergraduate(UG) and postgraduate(PG) programs. Each program has different eligibility criteria based on the student's percentage in their previous qualification.

UG Admissions require a minimum of 60%.

PG Admissions require a minimum of 70%.

**CLASS DIAGRAM:**

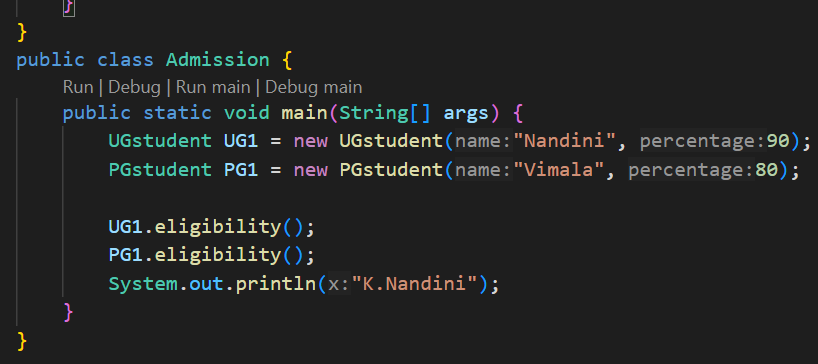
|  |
| --- |
| College |
| + Name: String  +Percentage : Double |
| + College(Name: String, Percentage: Double) |

|  |
| --- |
| UGstudent |
| + UGstudent(name: String, percentage: double) |
| + eligibility(): void |

|  |
| --- |
| PGstudent |
| + PGstudent(name: String, percentage: double) |
| + eligibility(): void |

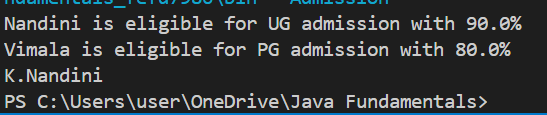
**CODE:**

****

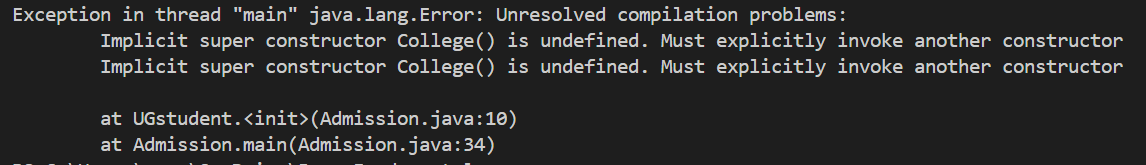
****

**OUTPUT:**

**POSITIVE CASE:**

****

**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors & Exceptions | Rectification |
| 1 | Didn’t defined the constructor for UGstudent and PGstudent classes. This causes an Exception | Rectifying by defining constructors to UGstudent and PGstudent classes and then assigning it to super class constructor |

**IMPORTANT POINTS:**

* Constructor Overloading: Both UGstudent and PGstudent classes use the constructor of the parent College class by calling super(name, percentage).
* Inheritance of Constructors: Subclasses can call constructors of the superclass using super() to initialize inherited fields.

**PROGRAM-3**

**AIM**: Create a Calculator class with overloaded methos to perform addition:

i) Add two integers

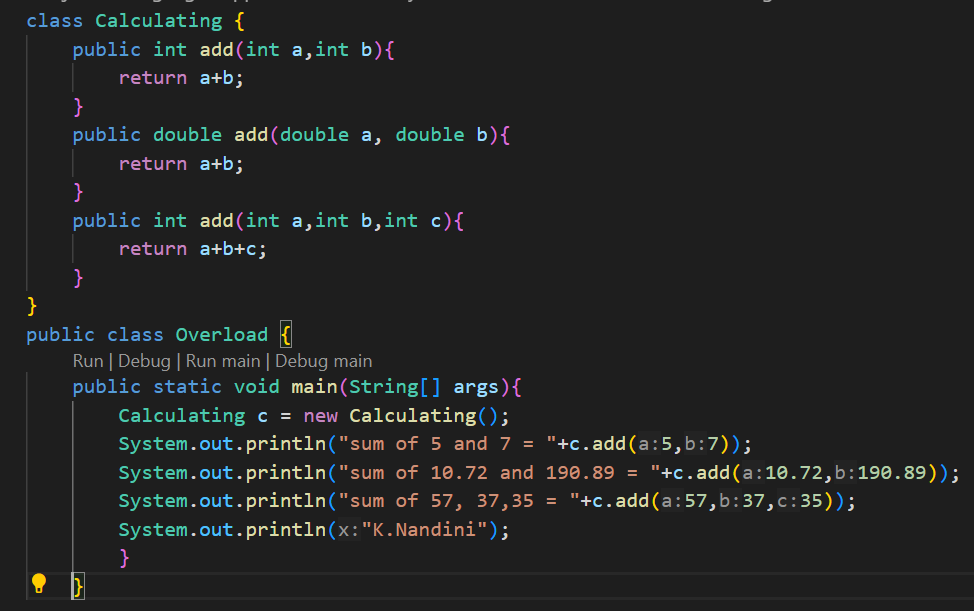
ii) Add two doubles

iii) Add three integers

**CLASS DIAGRAM:**

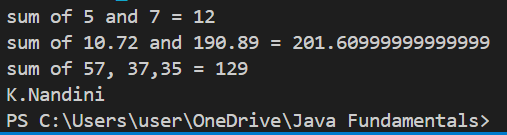
|  |
| --- |
| Calculating |
| +add(a: int, b: int): int  +add(a: double, b: double): double  +add(a: int, b: int, c: int): int |

**CODE:**



**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | While creating the object for Calculating class , given lower case letter c in class name Calculating. | Rectifying by giving uppercase letter C in the class name Calculating in object creation |

**IMPORTANT POINTS:**

Method Overloading:

* Method overloading is when you define multiple methods with the same name but with different parameter lists (either in type, number, or both).
* In this Code, The add() method is overloaded in the Calculating class in three ways:

add(int a, int b), add(double a, double b) and add(int a, int b, int c)

* This allows to perform addition with different types and numbers of arguments, demonstrating Compile-time polymorphism

**PROGRAM-4**

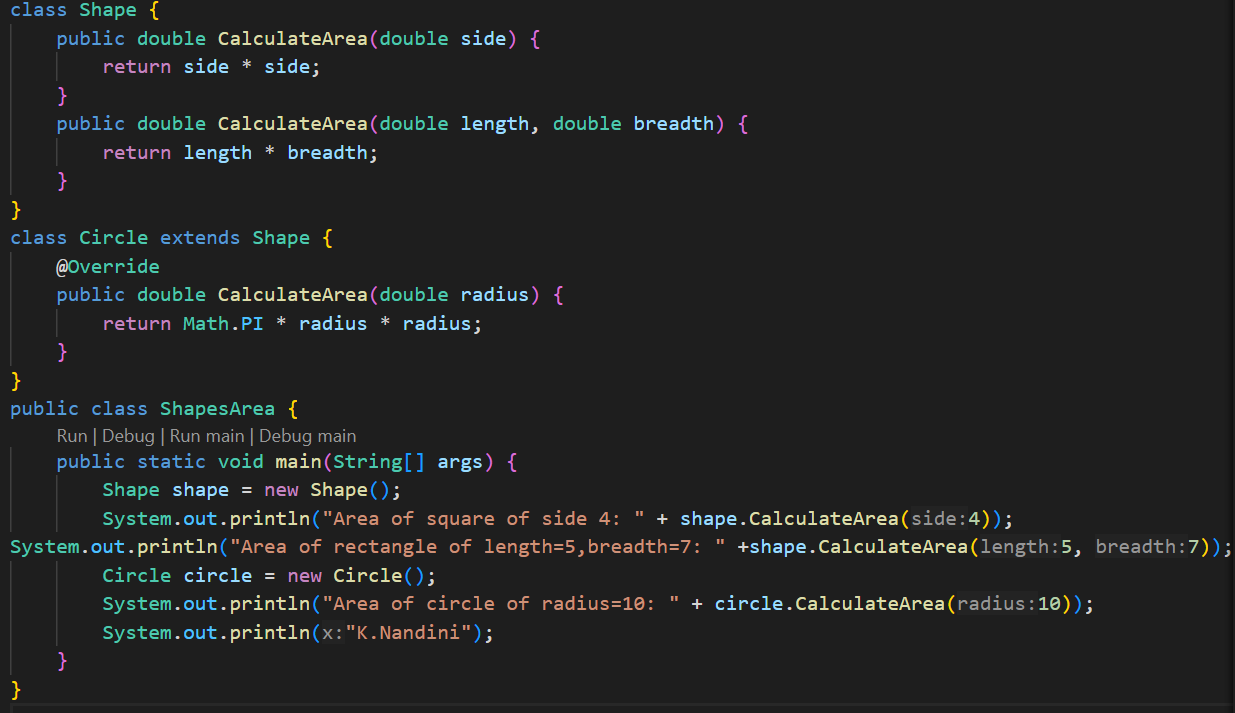
**AIM:** Create a shape class with a method calculateArea() that is overloaded for different shapes (e.g square,rectangle). Then create a subclass Circle that overrides the calculteArea().

**CLASS DIAGRAM**

|  |
| --- |
| Shape |
| +CalculateArea(double side): double  +CalculateArea(double length, double breadth): double |

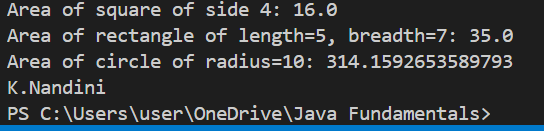
|  |
| --- |
| Circle |
| +CalculateArea(double radius): double |

**CODE:**

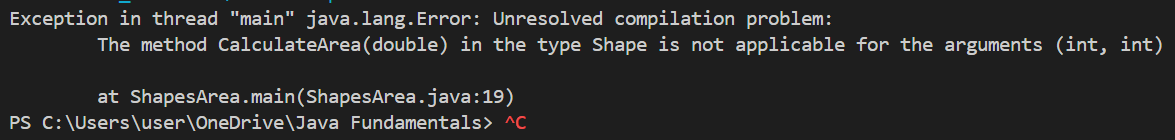


**OUTPUT**

**POSITIVE CASE:**



**NEGATIVE CASE:**



**ERROR TABLE**:

|  |  |  |
| --- | --- | --- |
| S.No | Errors & Exceptions | Rectification |
| 1 | While creating the object in main class , the values for length is only given in add(double length, double breadth) instead of giving two values | Rectifying by giving two double values for calculating area of rectangle |

**IMPORTANT POINTS:**

Method Overloading and Method Overriding

* In the Shape class, three methods named CalculateArea() are defined with different parameter lists. Both Overloading and Overriding occurs in this code. The methods in Shape Class undergoes Overloading and method in Circle class undergoes Overriding with the methods of Shape Class.
* This is compile-time polymorphism and run-time polymorphism.

**WEEK-7**

**PROGRAM-1**

**AIM:** Write a java program to create an abstract class Animal with an abstract method called sound(). Create subclasses Lion and Tiger that extend the Animal class and implement the sound() method to make a specific sound for each animal.

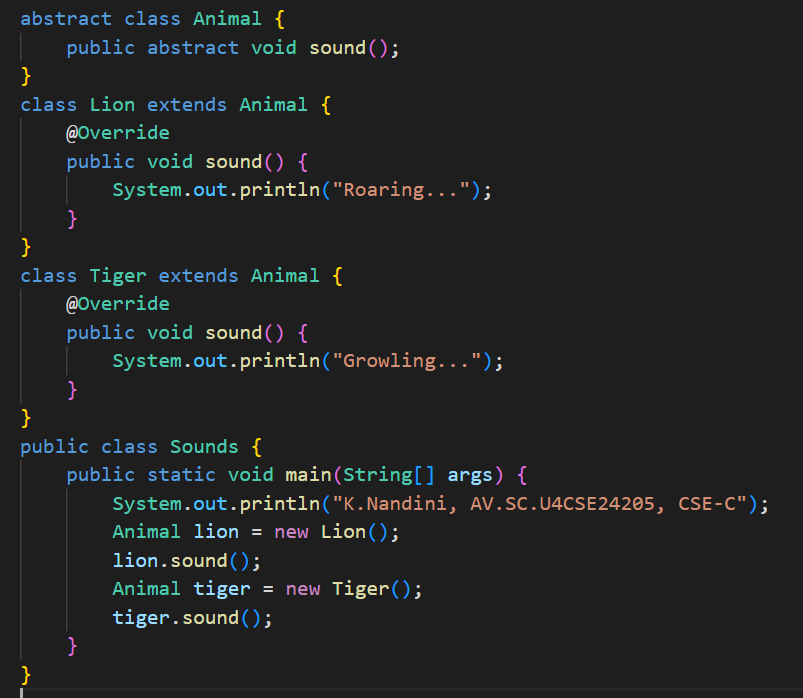
**CLASS DIAGRAM:**

|  |
| --- |
| <<abstract>>  Animal |
| +sound(): void |

|  |
| --- |
| Lion |
| +sound(): void |

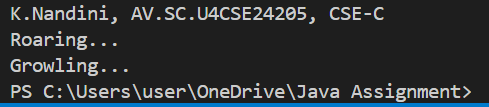
|  |
| --- |
| Tiger |
| +sound(): void |

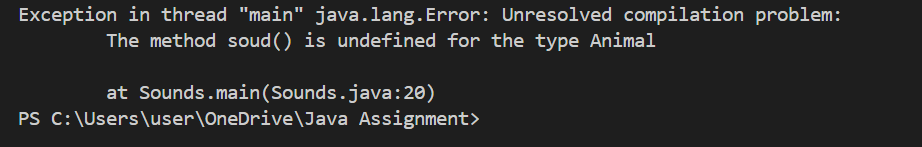
**CODE:**

****

**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:** 

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors & Exceptions | Rectification |
| 1 | Error while printing the variables. | Give the plus sign while printing |

**IMPORTANT POINTS:**

1.Abstraction

* The class Animal is declared as abstract and includes an abstract method sound(), which means:
  + It cannot be instantiated directly.
  + All subclasses must implement the sound() method.
* This defines a generic interface for all animals to follow without specifying *how* each animal sounds.

2. Method Overriding or Run-time Polymorphism

* Both Lion and Tiger override the sound() method from the abstract class:
  + Lion: "Roaring..."
  + Tiger: "Growling..."
* This allows each subclass to define its own version of the abstract behavior.
* We are using superclass references (Animal lion = new Lion();) to call subclass methods at runtime.
* This is called dynamic method dispatch, a core concept of runtime polymorphism in Java.

**PROGRAM-2**

**AIM:** Write a java program to create an abstract class Shape3D with abstract methods calculateVolume() and calculateSurfaceArea(). Create subclasses Sphere and Cube that extend the Shape3D class and implement the respective methods to calculate the volume and surface area of each shape.

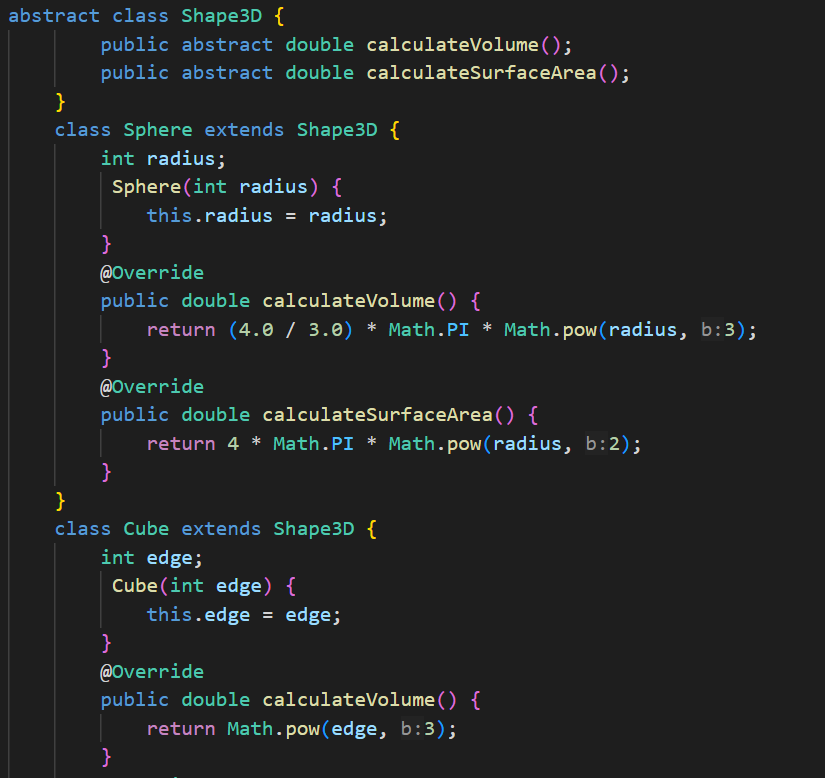
**CLASS DIAGRAM:**

|  |
| --- |
| <<abstract>>  Shape3D |
| +calculateVolume(): double  +calculateSurfaceArea(): double |

|  |
| --- |
| Sphere |
| +radius: int |
| +calculateVolume()  +calculateSurfaceArea() |

|  |
| --- |
| Cube |
| +edge: int |
| +calculateVolume()  +calculateSurfaceArea() |

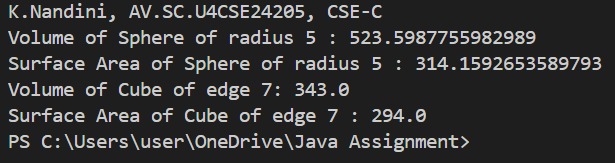
**CODE:**

****

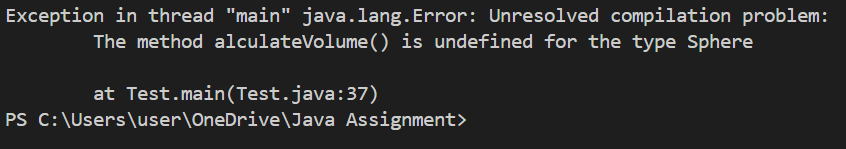
****

**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**

****

**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | ; is not given in print statement | Rectified by giving ; |

**IMPORTANT POINTS:**

1.Abstraction

* Shape3D is an abstract class that defines the blueprint for all 3D shapes:
  + calculateVolume() and calculateSurfaceArea() are abstract methods, meaning subclasses must implement them.

2.Method Overriding

* Both Sphere and Cube override the abstract methods:
* Each class provides its own formula for volume and surface area.
* This allows different behaviors for the same method signature.

3. Inheritance

* Sphere and Cube inherit from Shape3D, demonstrating "is-a" relationship.

Shows how subclasses can extend an abstract class to add specific behavior

**PROGRAM-3:**

**AIM:**Write a Java program using an abstract class to define a method for pattern printing.

Create an abstract class named PatternPrinter() with an abstract method printPattern(int n) and a concrete method to display the patter title.

Implement two subclasses

StarPattern -prints a right angled triangle of stars

NumberPattern - prints a right angled triangle of increasing numbers

In the main() method create objects of both subclasses and print the pattern for a given number of rows

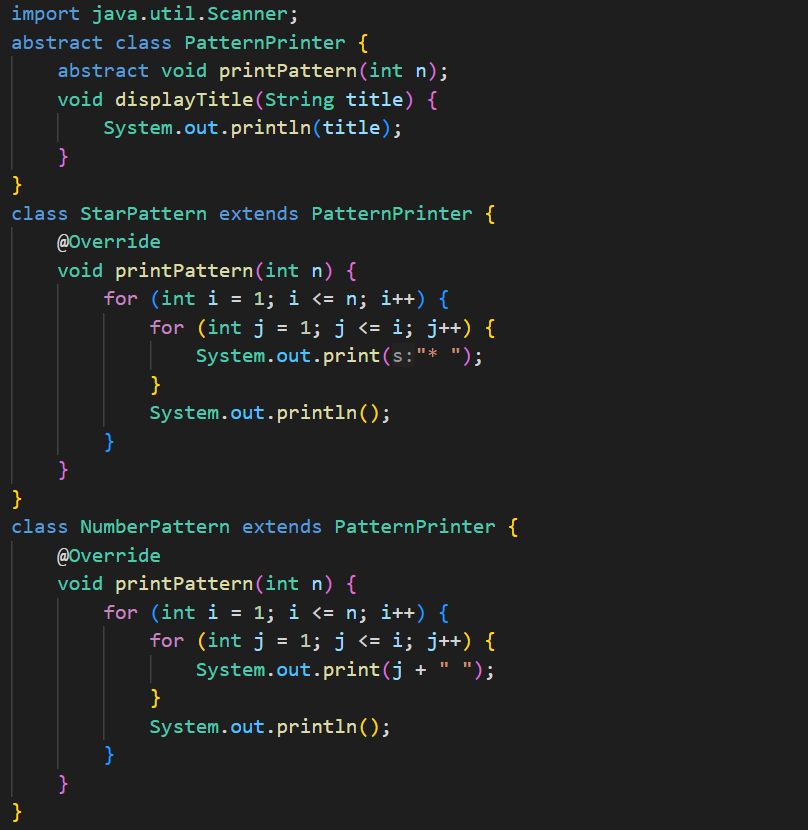
**CLASS DIAGRAM:**

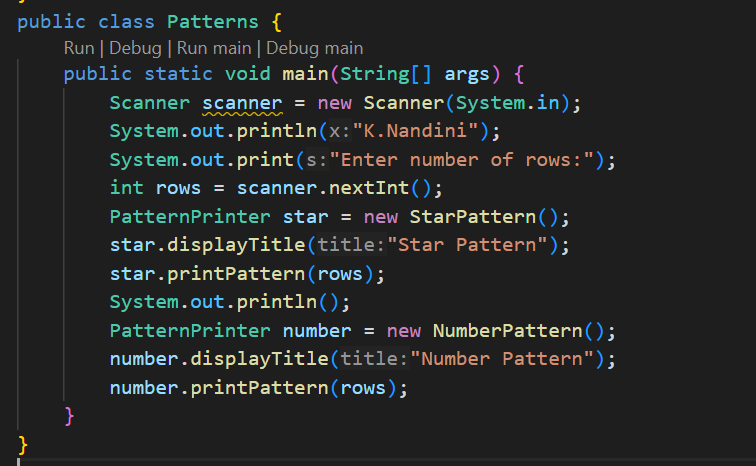
|  |
| --- |
| <<abstract>>  PatternPrinter |
| +printPattern(int): void <<abstract>>  +displayTitle(String):void |

|  |
| --- |
| StarPattern |
| +printPattern(n):void |

|  |
| --- |
| NumberPattern |
| +printPattern(n):void |

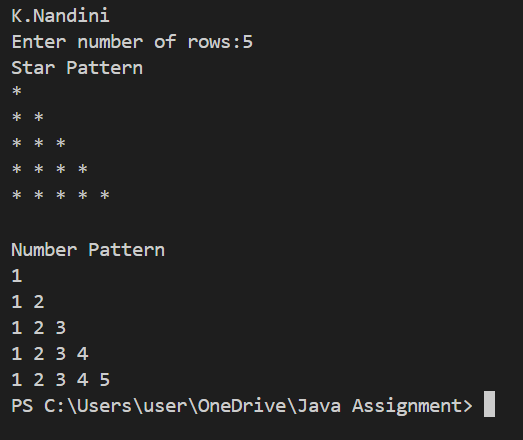
**CODE:**



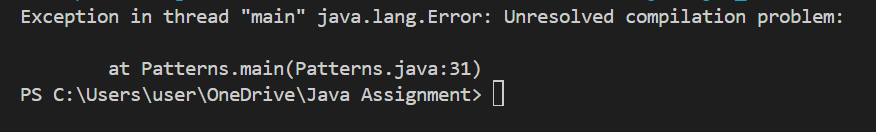


**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | Syntax errors in for loops | Rectified by giving correct syntax |

**IMPORTANT POINTS:**

1. Use abstract classes to enforce a common structure for pattern printing.
2. PatternPrinter is the abstract class defining the common template.
3. Subclasses (StarPattern, NumberPattern) provide specific implementations.
4. displayTitle() is a concrete method shared by all subclasses.

**WEEK-8**

**PROGRAM-1**

**AIM**:Write a java program to create an interface Shape with the getPerimeter() method. Create three classes Rectangle, Circle and Triangle that implement the Shape interface. Implement the getPerimeter() method for each of the three classes.

**CLASS DIAGRAM:**

|  |
| --- |
| <<interface>>  Shape |
| +getPerimeter(): double |

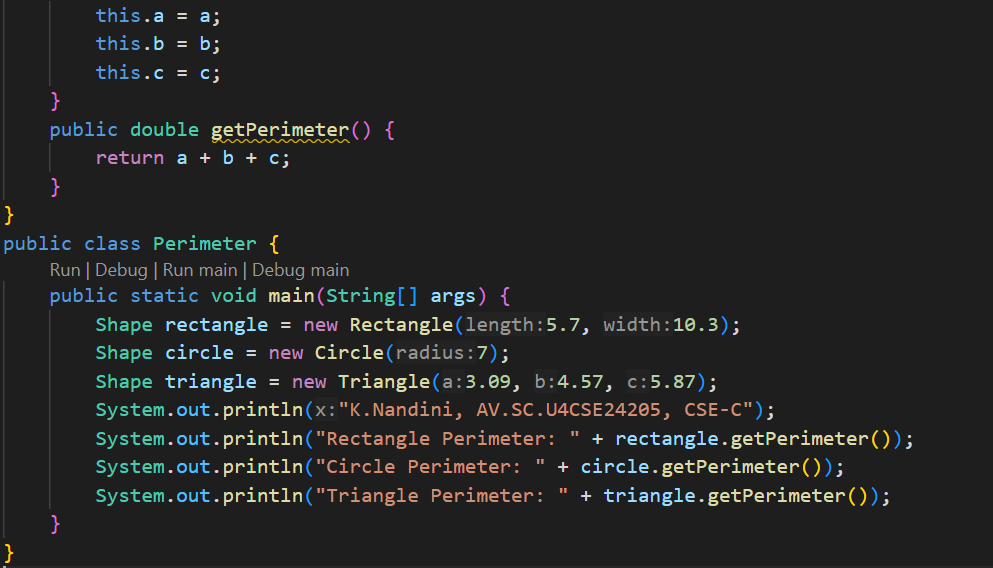
|  |
| --- |
| Circle |
| +radius: double |
| +getPerimeter() |

|  |
| --- |
| Triangle |
| +a: double  +b: double  +c: double |
| +getPerimeter() |

|  |
| --- |
| Rectange |
| +length: double  + width: double |
| +getPerimeter() |

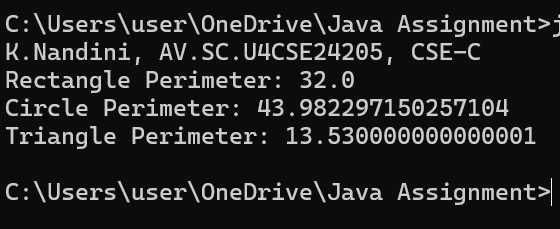
**CODE:**



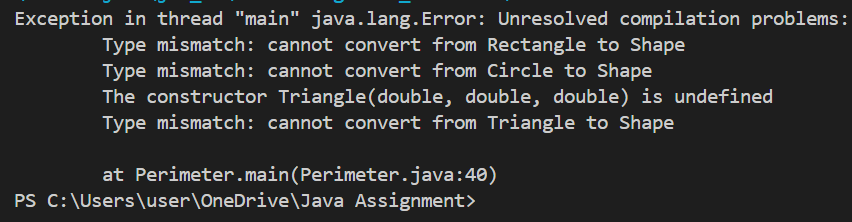


**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | Not giving interface keyword to the Shape class | Rectifying the error by giving interface keyword |

**IMPORTANT POINTS:**

1. Interfaces in Java

* The Shape interface defines a contract for all shapes that must implement getPerimeter().
* Promotes loose coupling and flexibility in design.
* Interfaces allow multiple classes to follow a common behavior without using inheritance.

2. Polymorphism (Runtime Polymorphism)

* You’re using interface references (Shape rectangle = new Rectangle(...)) to refer to different shape objects.
* This demonstrates dynamic method dispatch, where the appropriate getPerimeter() is called based on the actual object type at runtime.

3. Encapsulation

* Each class has private data (length, width, radius, sides), and the behavior (getPerimeter()) is encapsulated inside the class.
* This keeps data and behavior bundled together.

**PROGRAM-2**

**AIM:**Write a java program to create an interface Playable with a method play() that takes no argument and returns void. Create three classes Football, Volleyball and Basketball that implement the Playable interface and override the play() method to play the respective sports

**CLASS DIAGRAM**

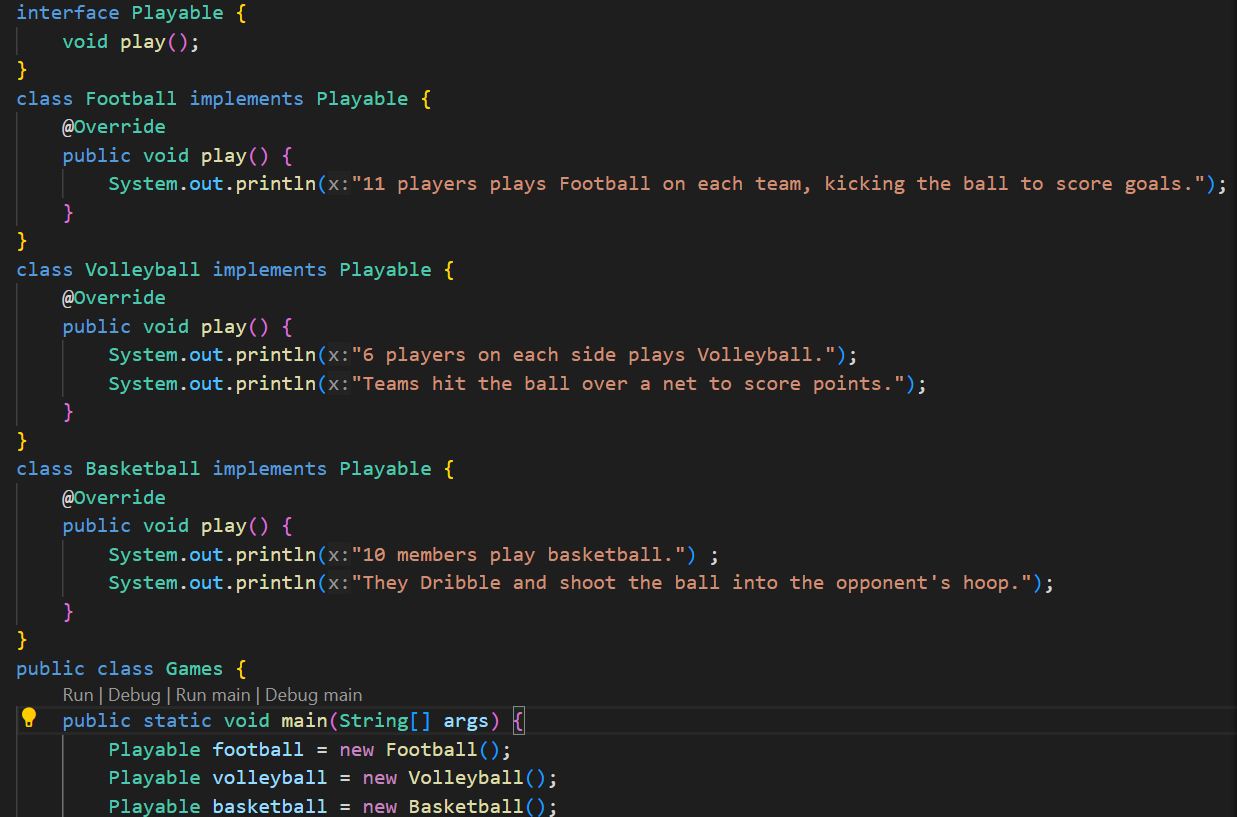
|  |
| --- |
| <<interface>>  Playable |
| +play(): void |

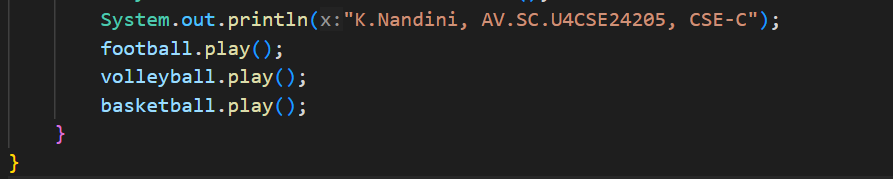
|  |
| --- |
| Volleyball |
| +play(): void |

|  |
| --- |
| BasketBall |
| +play(): void |

|  |
| --- |
| Football |
| +play(): void |

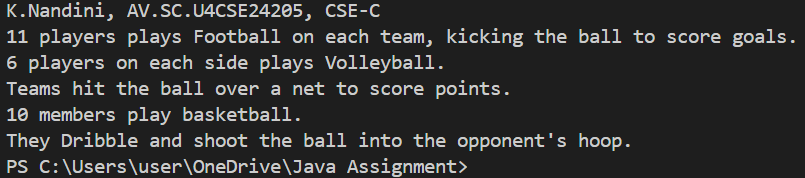
**CODE:**



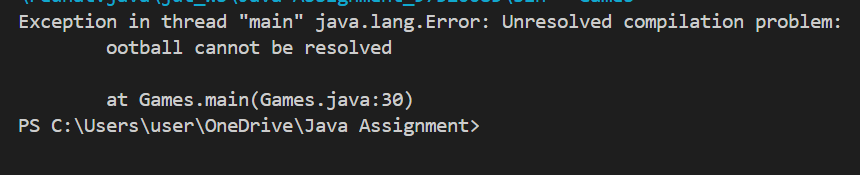


**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.NO | Erros | Rectification |
| 1 | FootBall football = new VolleyBall();  Ootball.play(); error occured | Rectified by giving correct name |

**IMPORTANT POINTS:**

1. Interfaces

* Playable is an interface that declares a play() method, which serves as a contract for any class that implements it.
* Interfaces allow you to define methods without providing the implementation, leaving it to the implementing class to define the behavior.

2. Implementation of Interface Methods

* Football, Volleyball, and Basketball are classes that implement the Playable interface. Each class overrides the play() method and provides its own specific implementation.
  + Football: Describes how the game is played with 11 players on each team, aiming to score goals by kicking the ball.
  + Volleyball: Describes a 6-player side game where teams try to hit the ball over the net to score points.
  + Basketball: Describes a 10-player game where players dribble and shoot the ball into the opponent's hoop.

3.Polymorphism

* The program demonstrates runtime polymorphism because:
  + The reference variable Playable is used for different game objects (Football, Volleyball, Basketball), but the appropriate play() method is called dynamically at runtime.