**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**

**WEEK-1**

**Program-1**

**DOWNLOADING AND INSTALLATION OF JAVA**

## AIM: To download and install Java(JDK 21)

**PROCEDURE:**

This is the process for installation of JDK on windows.

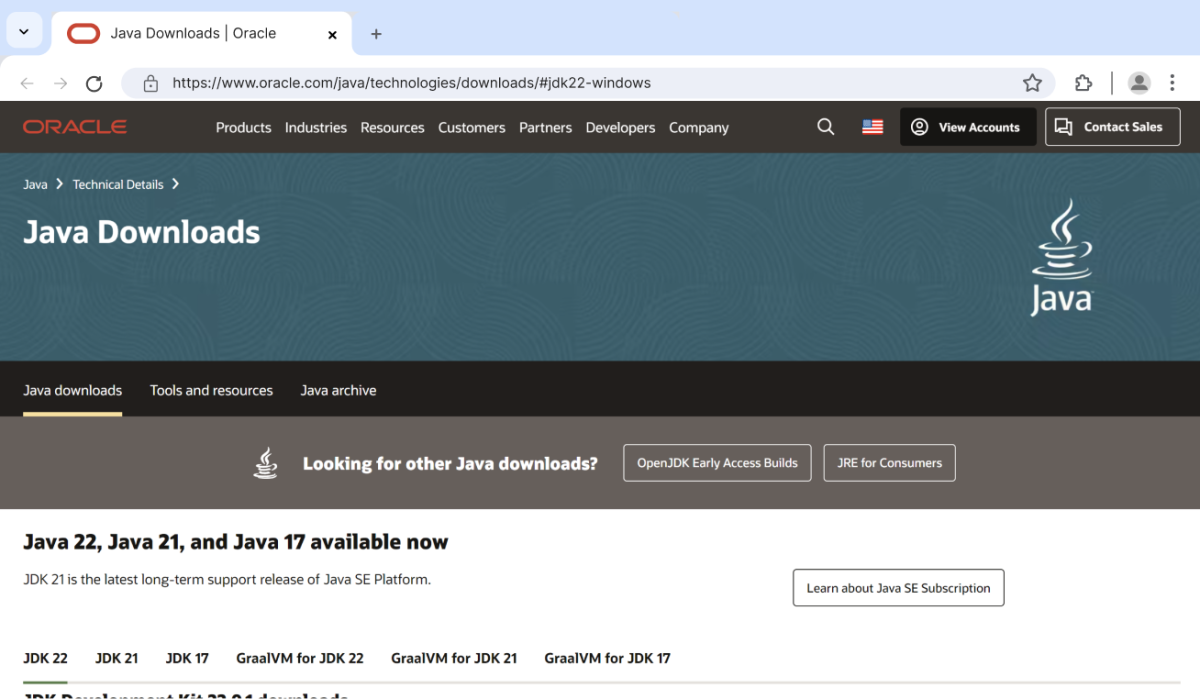
Follow the steps below to install Java on Windows:

* 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.

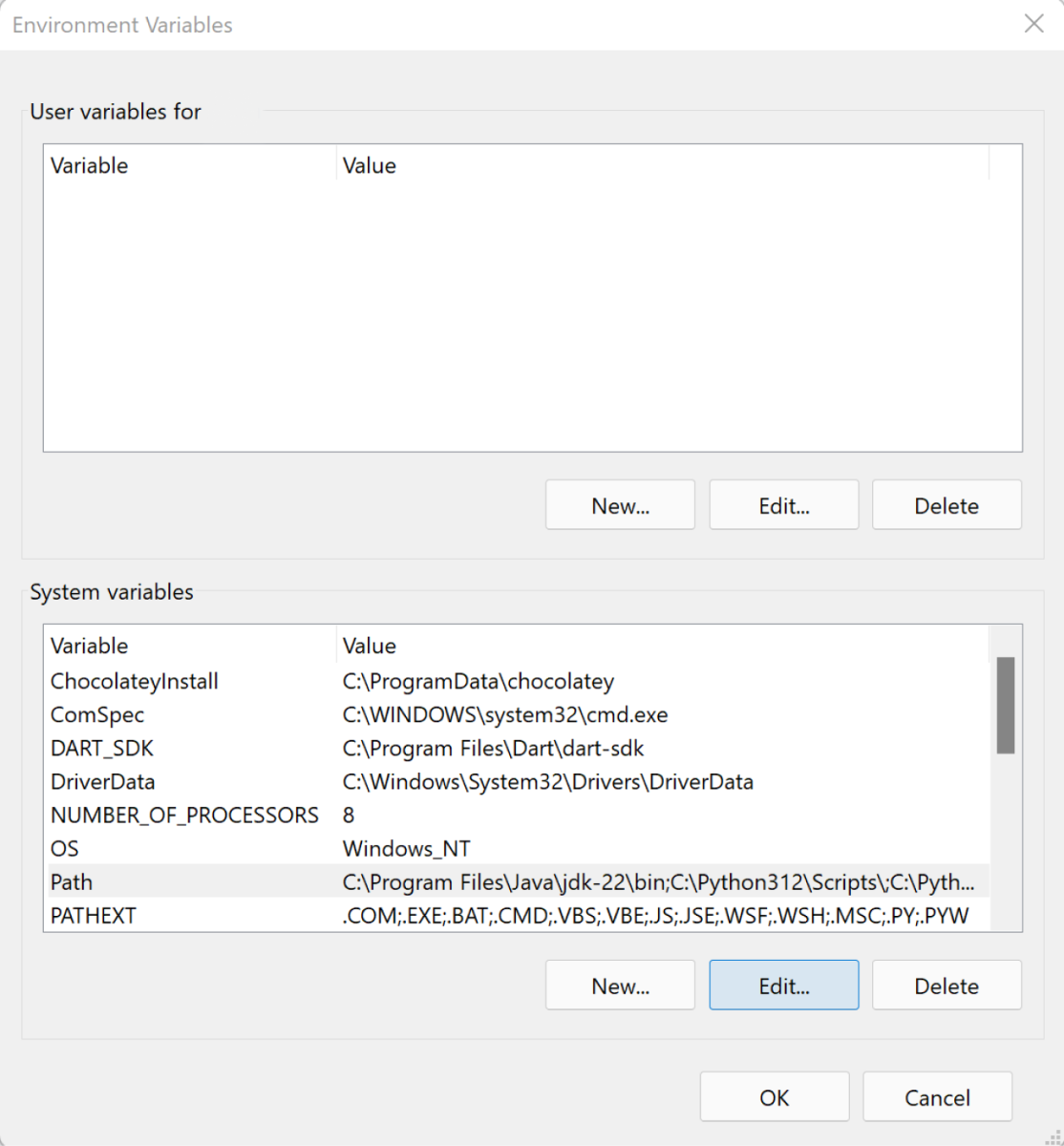
4) Select and copy the path at the address bar

**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**.

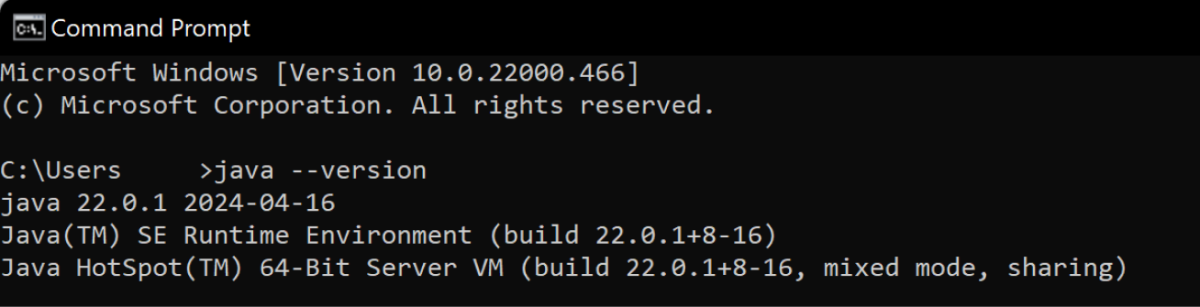
Then, click **New** and paste your JDK bin path (i.e. C:\Program Files\Java\jdk-22\bin).

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.

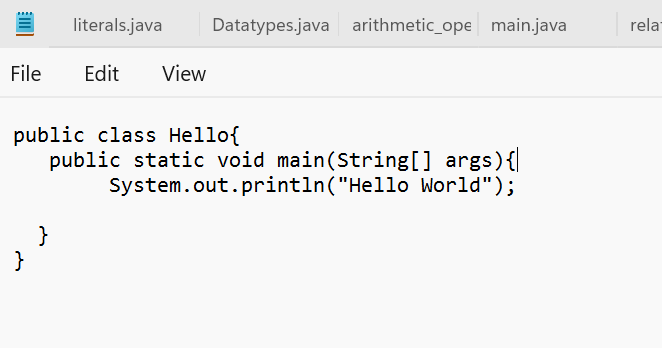
java –version

.

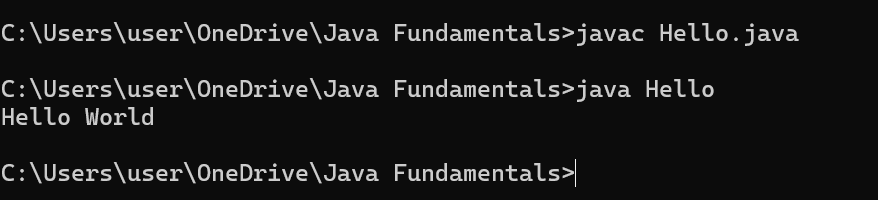
If Java is installed successfully, it will print the version information; otherwise, it will produce an error message indicating that the command is not recognized.

**PROGRAM-2:**

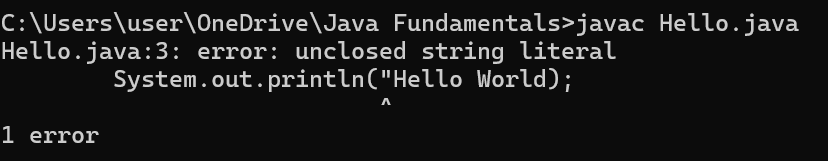
**AIM:** Write a Java program to print the message “Hello World”.

**Code**: **OUTPUT:**

**POSITIVE CASE:**

****

**NEGATIVE CASE:**

****

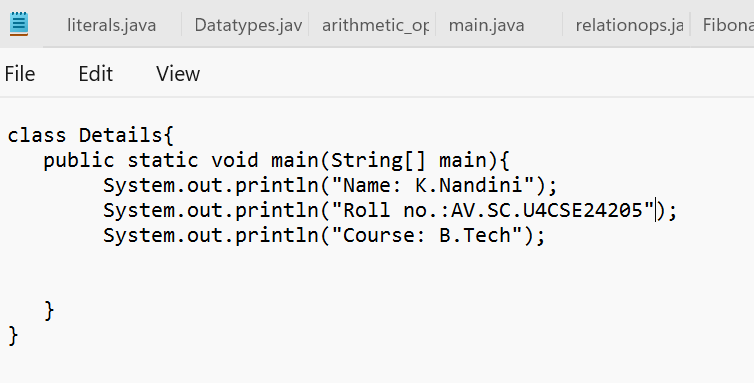
**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | ERRORS | RECTIFICATION |
| 1 | Double codes at the end of print statement in the | The error is rectified by writing s in uppercase letter |

**PROGRAM-3:**

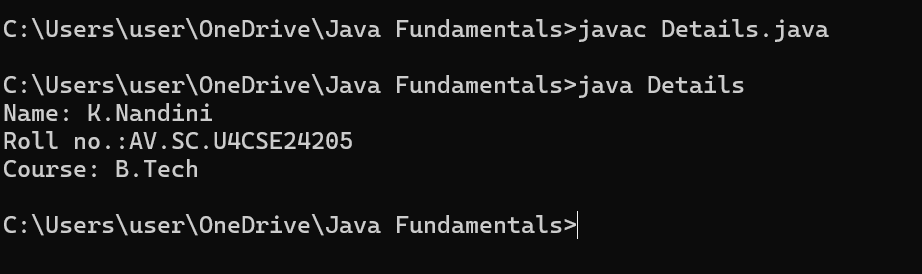
**AIM:** Write a Java Program that prints Name, Roll No, Section of a student.

**CODE:**

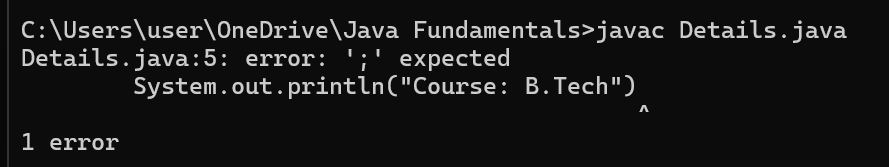


**OUTPUT:**

**POSITIVE CASE:**

****

**NEGATIVE CASE:**

****

**ERROR TABLE:**

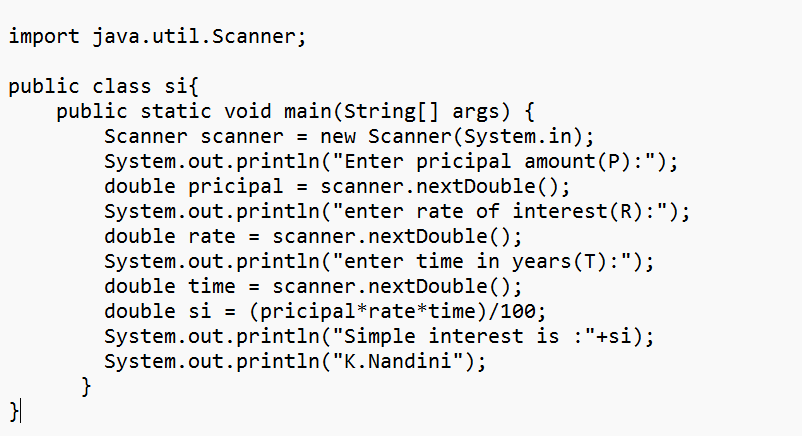
|  |  |  |
| --- | --- | --- |
| S.No | ERROR | RECTIFICATION |
| 1 | In the statement at the end ; is not mentioned | Rectified by keeping ; at the end of the ststement |

**WEEK – 2:**

**PROGRAM-1:**

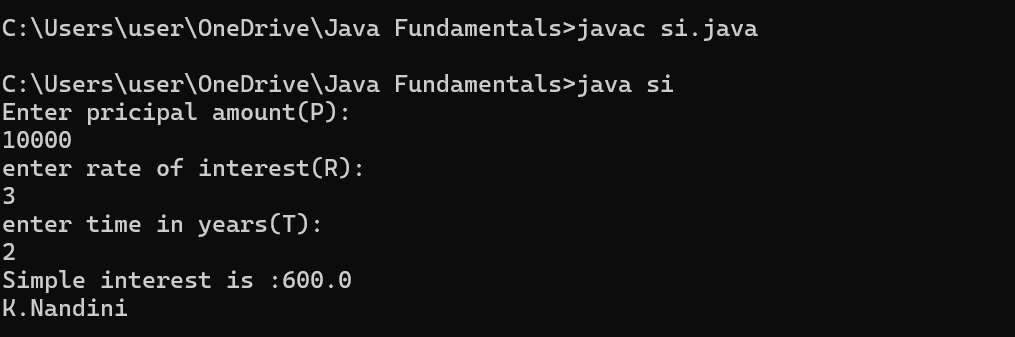
**AIM**: Write a java program to find the simple interest where all the inputs are taken from the user.

**CODE:**

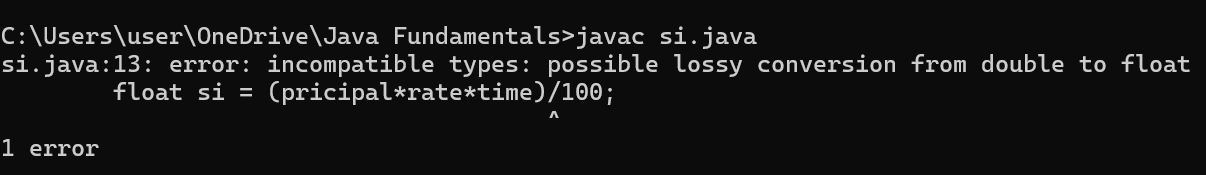


**OUTPUT:**

**POSITIVE CASE**:



**NEGATIVE CASE:**



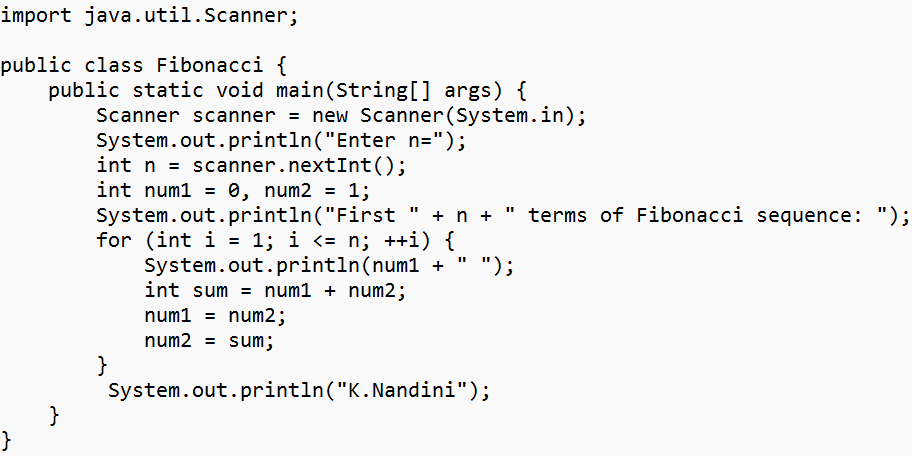
**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| 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 |

**PROGRAM-2:**

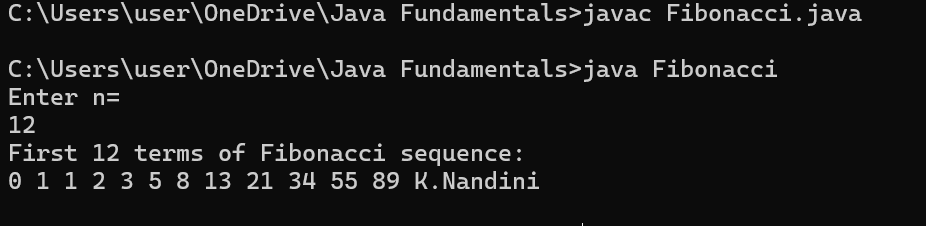
**AIM:** Write a java program to find the Fibonacci sequence of a given number

**CODE**:

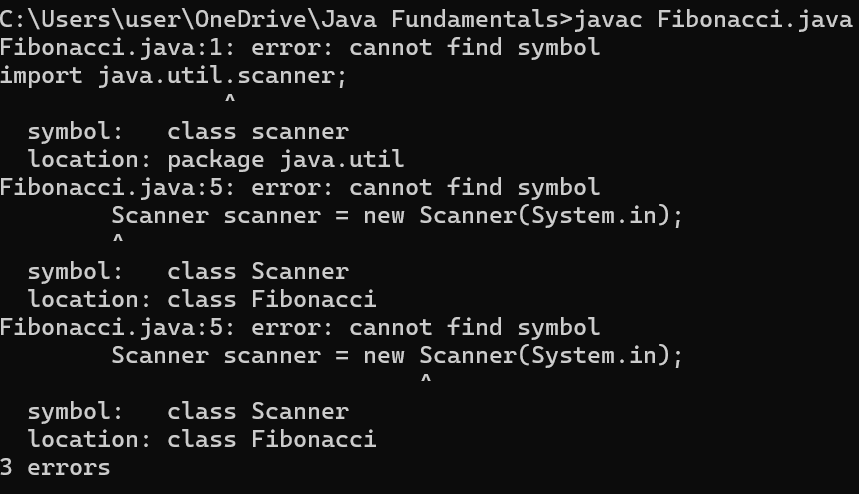


**OUTPUT:**

**POSITIVE CASE**:



**NEGATIVE CASE:**



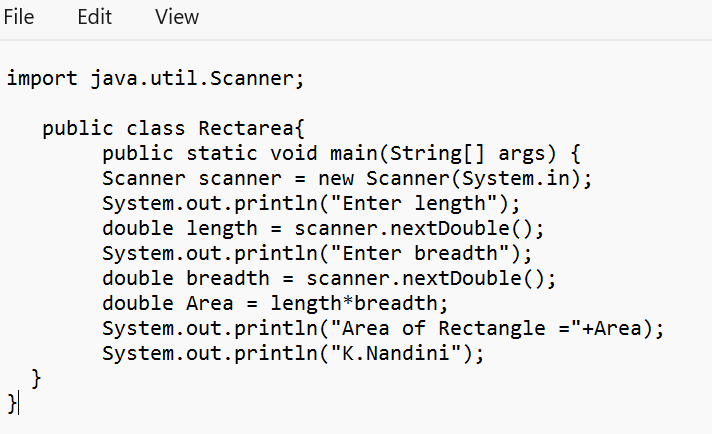
**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| 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:**

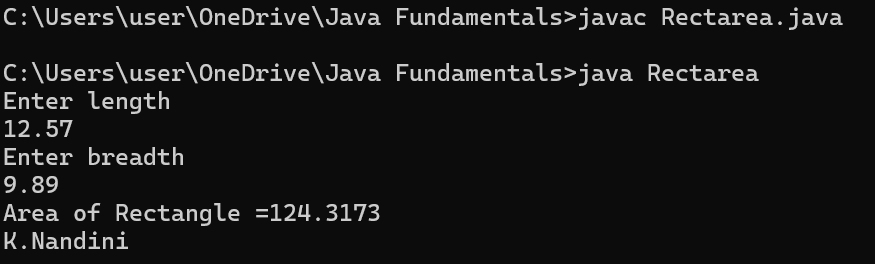
**AIM**: Write a java program to find the area of rectangle and triangle.

**CODE:**

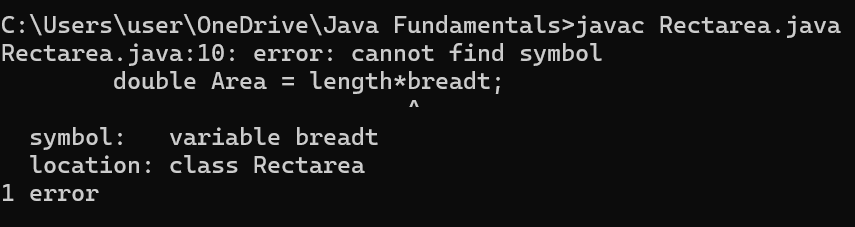


**OUTPUT:**

**POSITIVE CASE:**

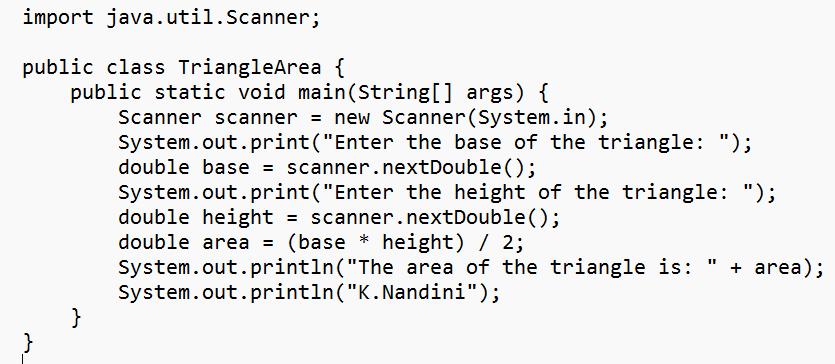


**NEGATIVE CASE:**

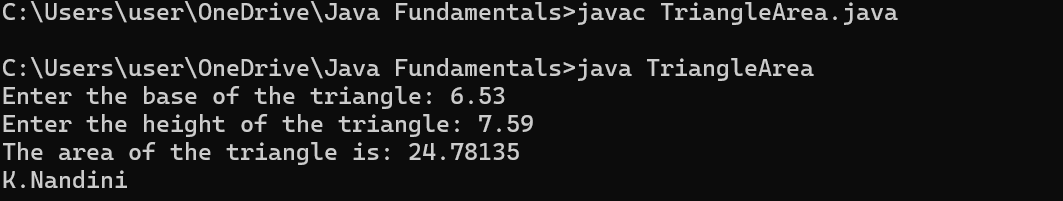


**ERROR TABLE:**

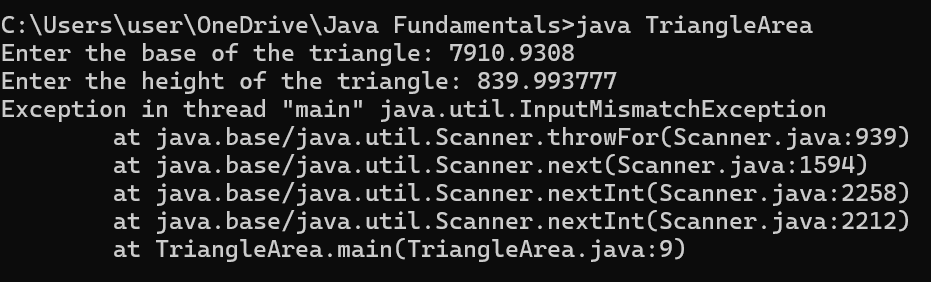
|  |  |  |
| --- | --- | --- |
| S.No | ERROR | RECTIFICATION |
| 1 | The variable name breadth has spelling wrong | Rectified by correcting the spelling |

**CODE:**  


**OUTPUT:**

**POSITIVE CASE:**  


**NEGATIVE CASE:**

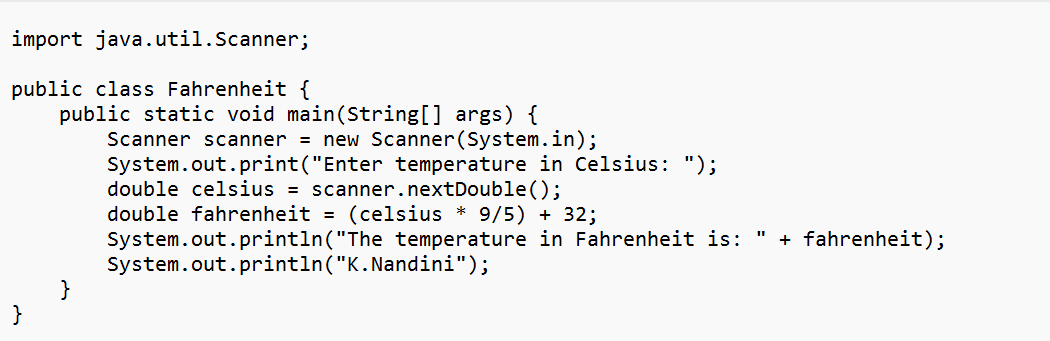


**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| 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 |

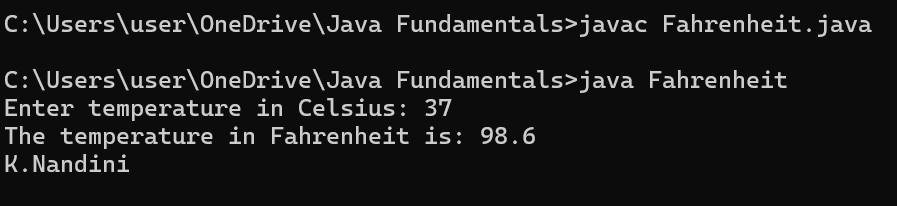
**PROGRAM-4:**  
AIM: Write a java code to convert the temperature from Celsius to Fahrenheit and from Fahrenheit to Celsius.

**CODE:**

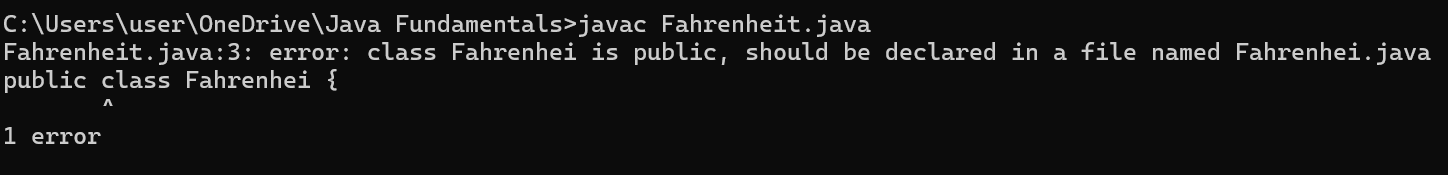


**OUTPUT:**

**POSITIVE CASE:**

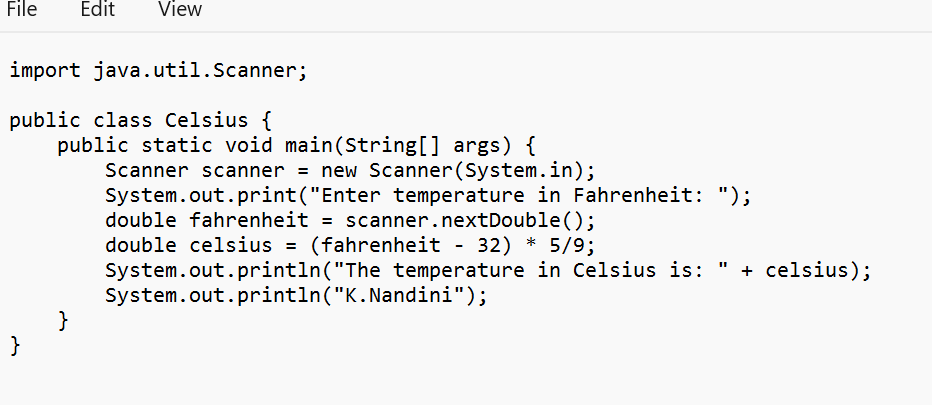


**NEGATIVE CASE:**



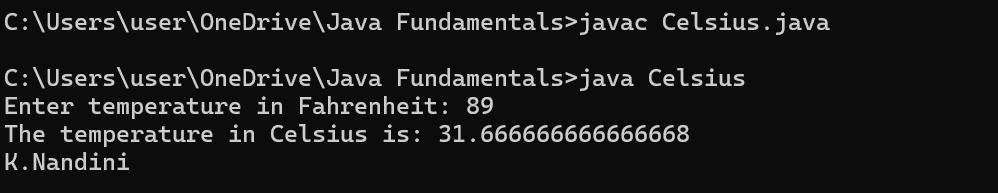
**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| 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 |

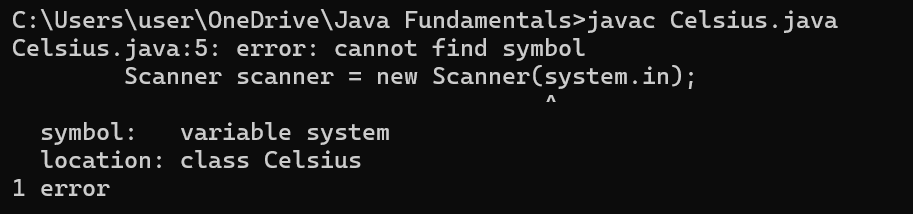
**CODE:**  


**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**



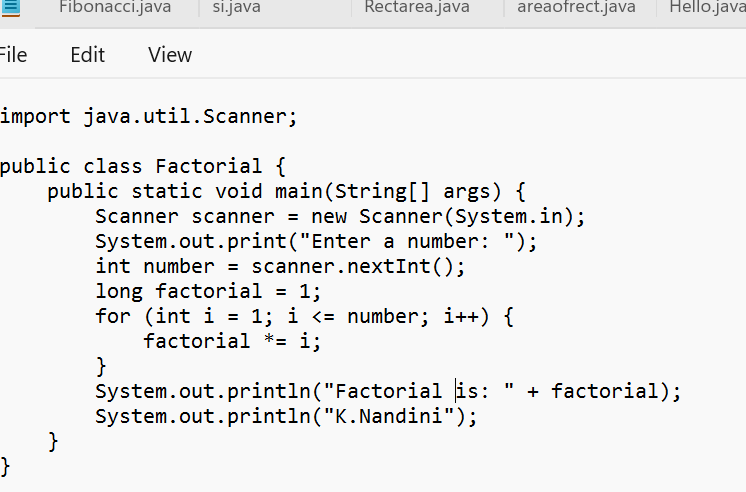
**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| 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**

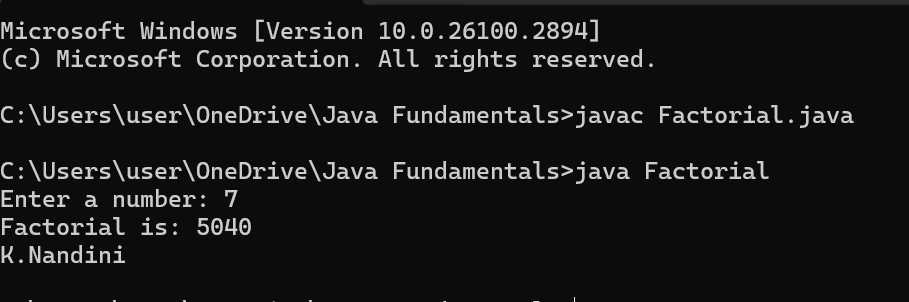
**AIM**: Write a java code to find factorial of a number by taking input.

**CODE:**

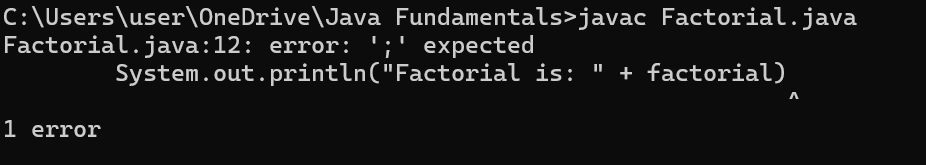


**OUTPUT:**

**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 |

**WEEK-3**

**Program-1**

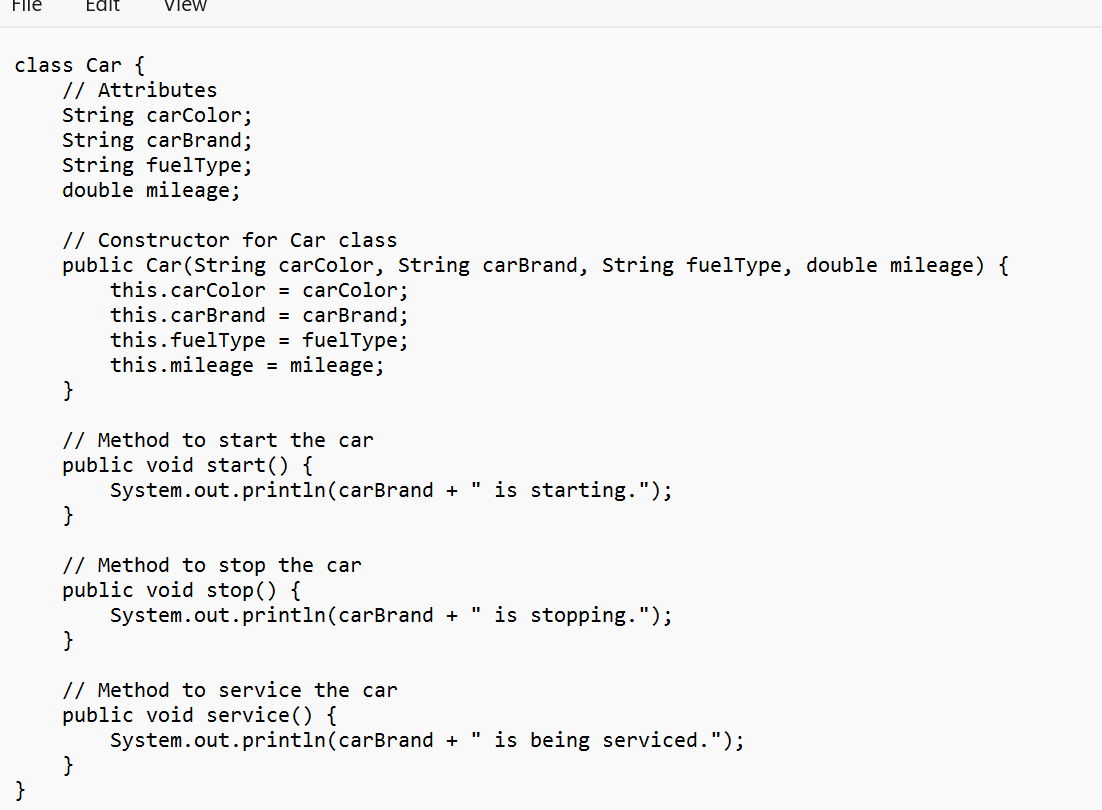
**AIM :** To create java program with following instructions

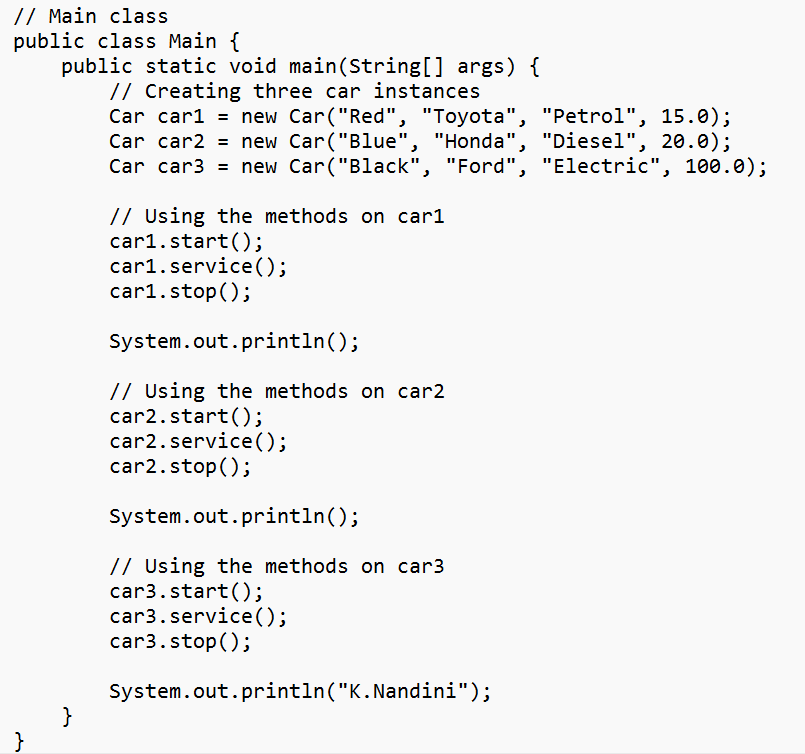
1. Create a class with name Car.
2. Create four attributes named Car\_color, Car\_brand, fuel\_type, mileage.
3. Create three methods named start(), stop(), service().
4. Create three objects named Car1, Car2, Car3.

**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 |

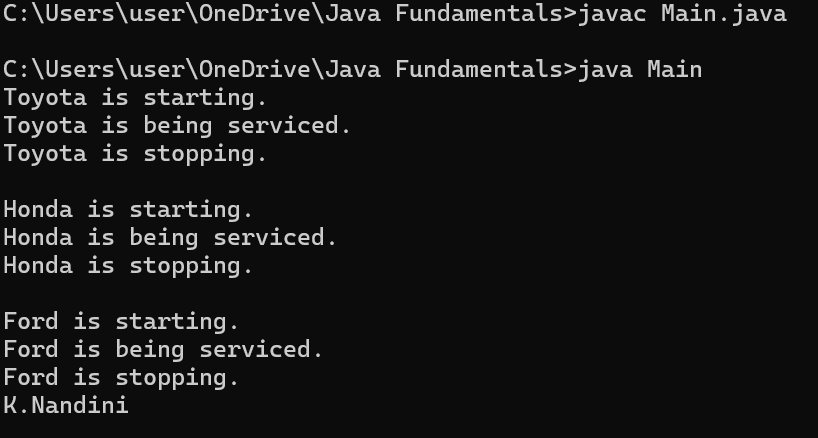
**CODE**



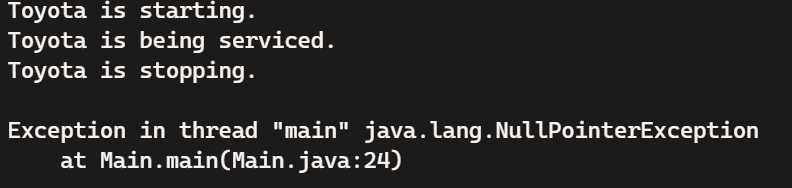


**OUTPUT**:

**POSITIVE CASE**:

****

**NEGATIVE CASE:**

****

**ERROR TABLE:**

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

**Important Points:**

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

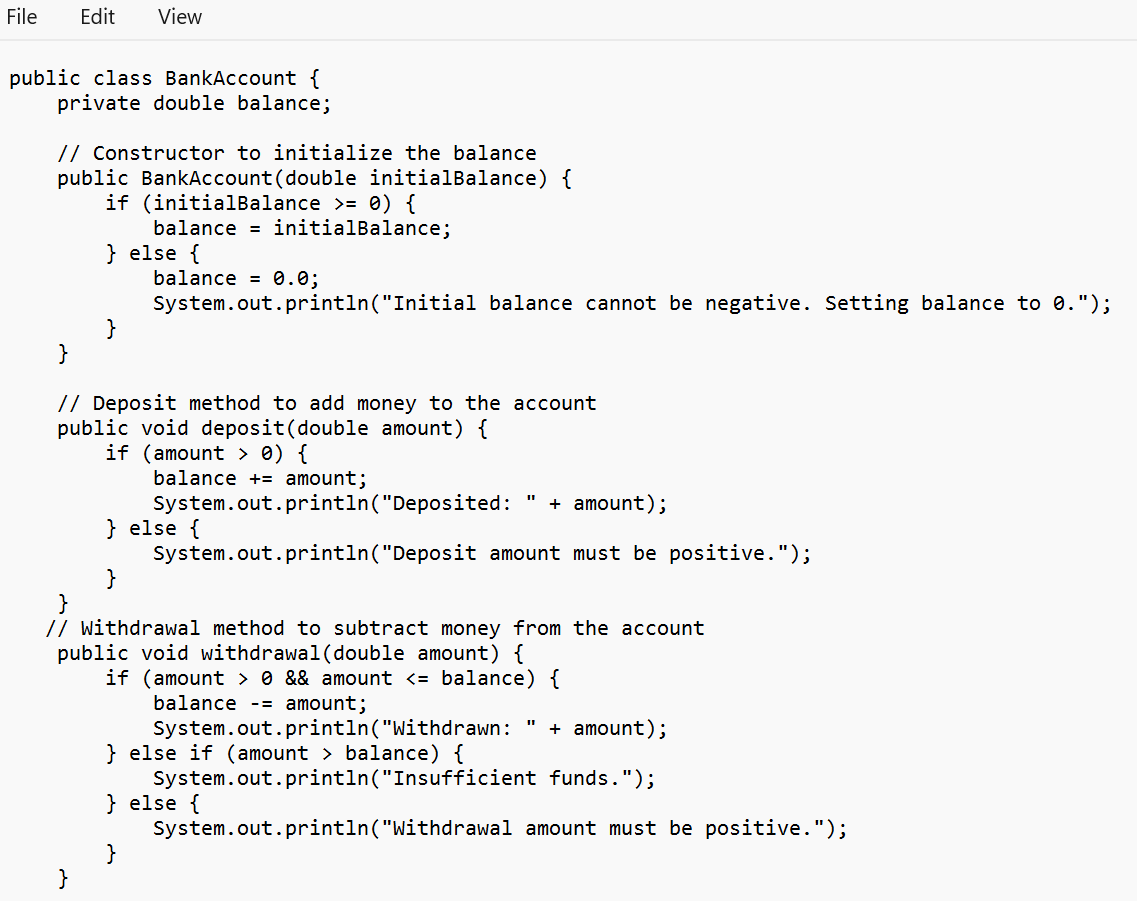
**Program-2:**

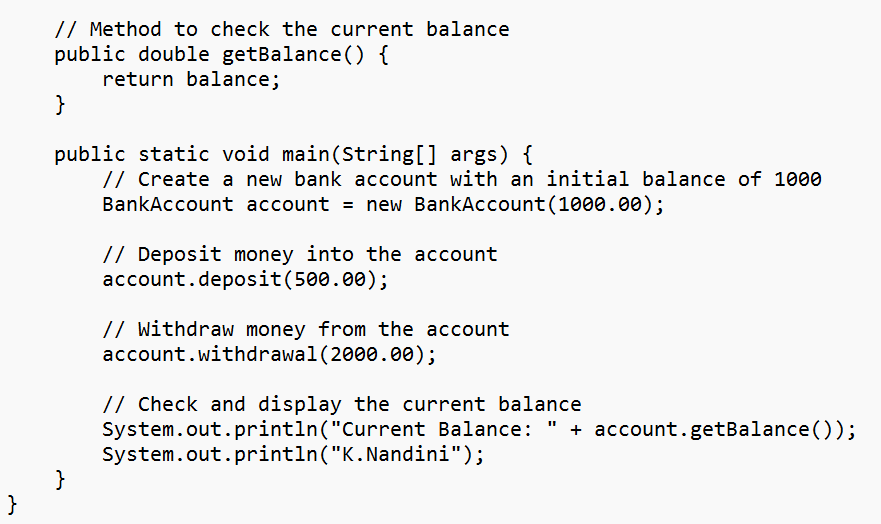
**Aim**: To create a class bank account with method deposit() and withdrawal().

**CLASS DIAGRAM:**

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

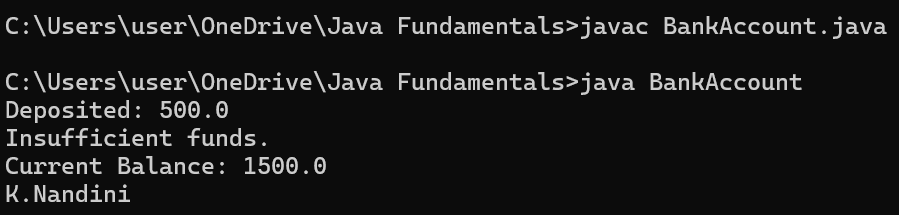
**CODE**:



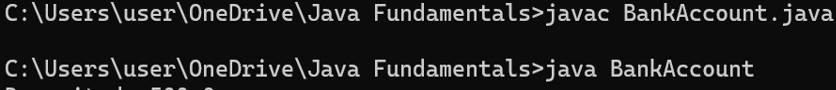


**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**





**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | 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:**

**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**

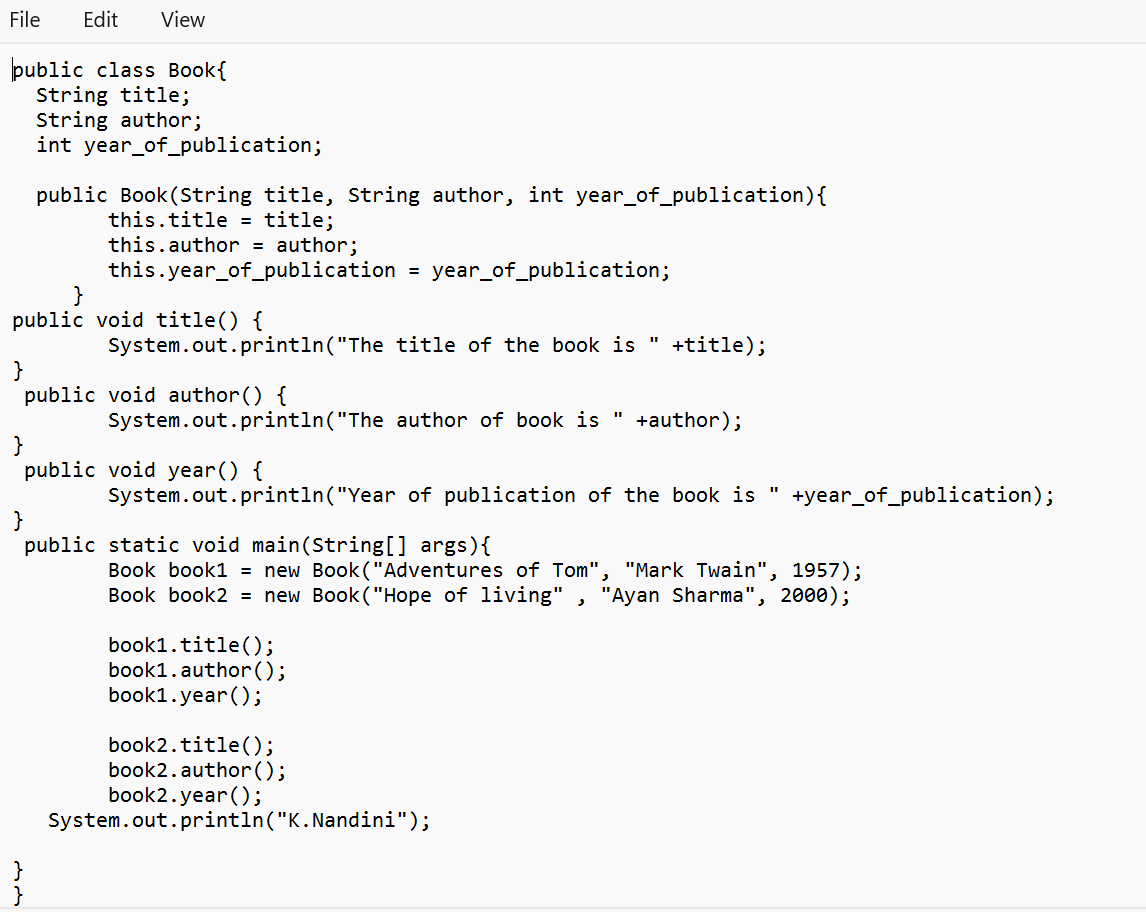
**PROGRAM-1**

**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:**

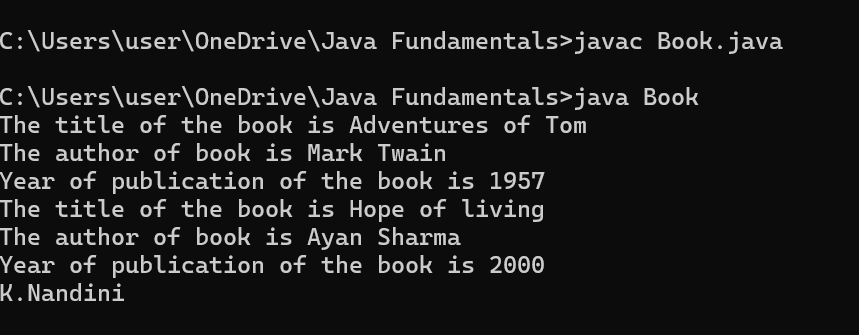
|  |
| --- |
| Book |
| - title: String  - author: String  - year\_of\_publication: int |
| + Book(String, String, int)  + title(): void  + author(): void  + year(): void  + main(String[]): static void |

**CODE:**

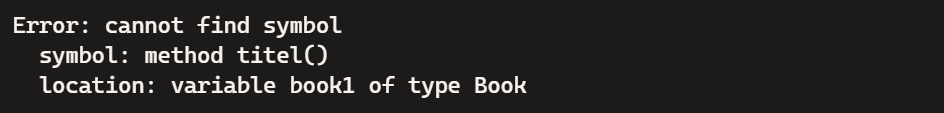
****

**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.

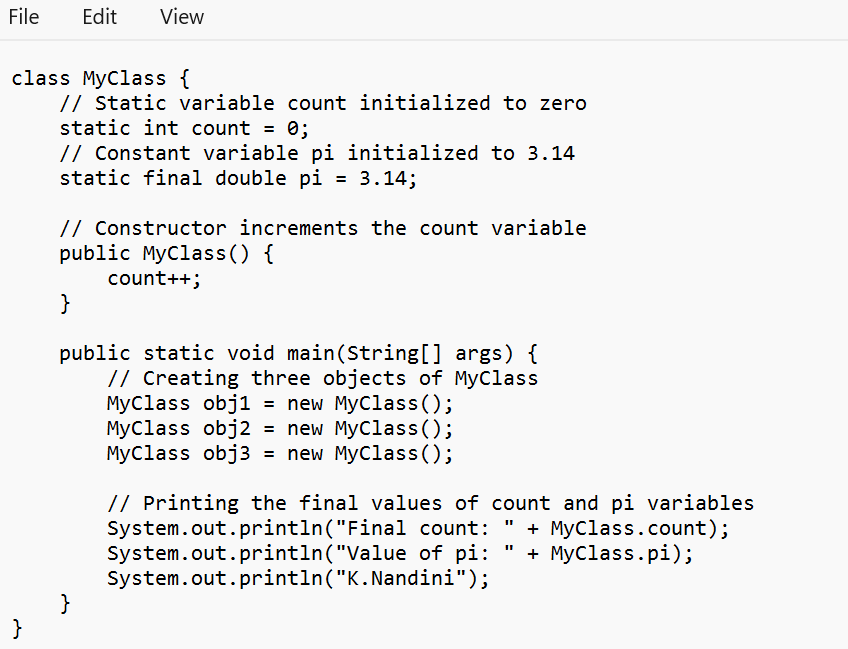
**PROGRAM-2**

**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:**

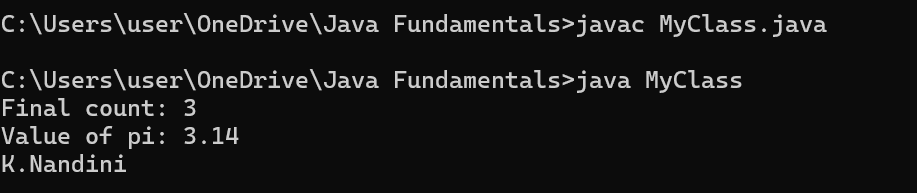
|  |
| --- |
| MyClass |
| - count: static int  - pi: final double |
| + MyClass()  + main(String[]): static void |

**CODE:**



**OUTPUT:**

**POSITIVE CASE:**



**NEGATIVE CASE:**



**ERROR TABLE:**

|  |  |  |
| --- | --- | --- |
| S.No | Errors | Rectification |
| 1 | Calling Non-Existent Methods  Ex: By typing Myclass instead of MyClass while calling the method | Rectifying the method name as MyClass |
| 2 | S is System.out.println is given in lowercase letters | The error is rectified by replacing the lowercase letter s into uppercase letter S |

**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:**

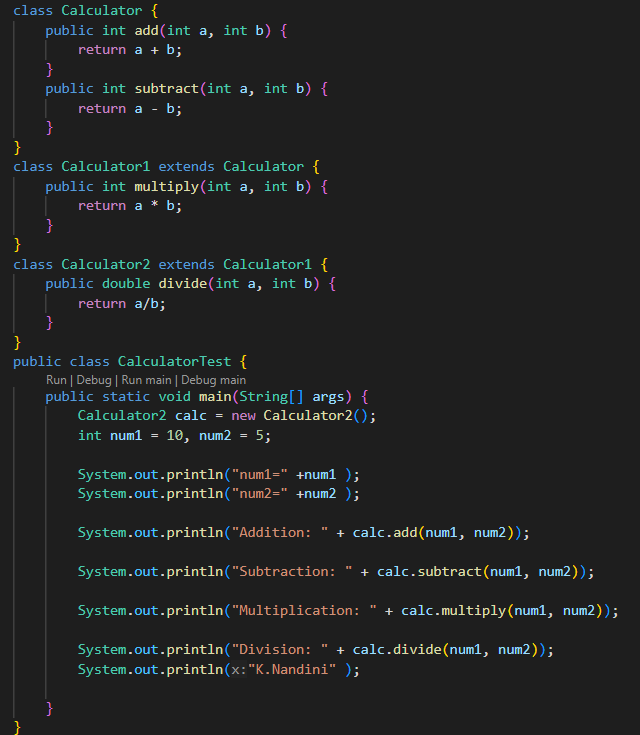
**MULTILEVEL INHERITANNCE**

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

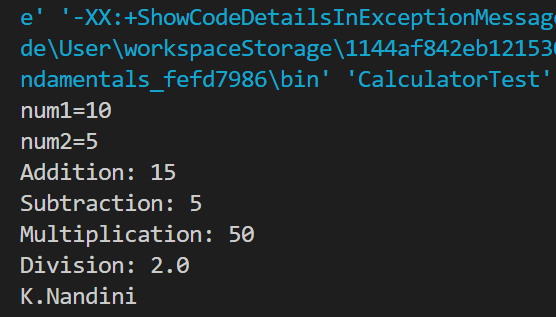
|  |
| --- |
| Calculator1 |
| - num1: int  - num2: int |
| + multiply(): int |

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

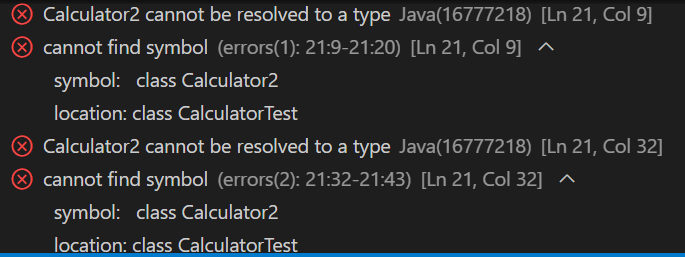
**CODE:**



**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 (Use of Parent Class Reference):

* 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.

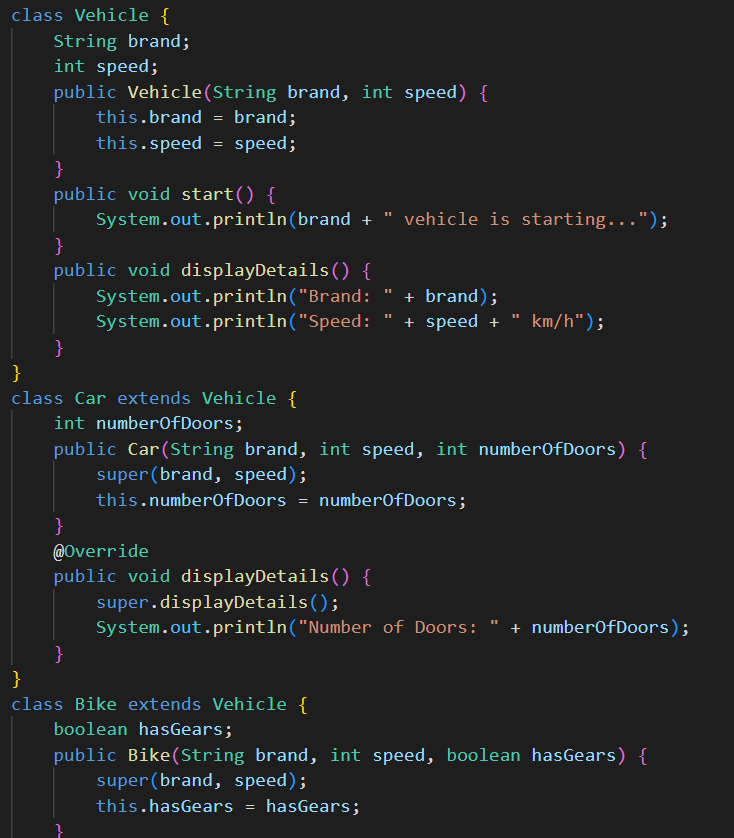
**CLASS DIAGRAM**:

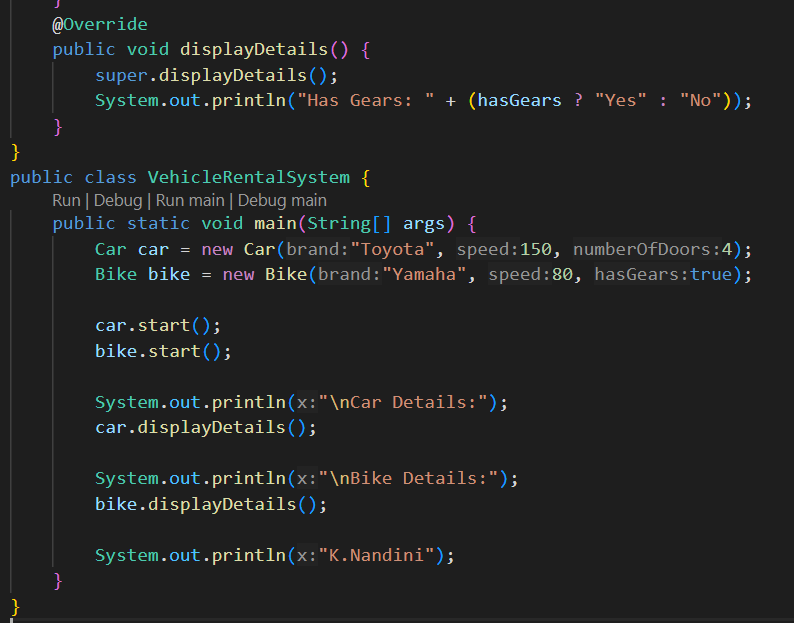
|  |
| --- |
| Vehicle |
| + brand: String  + speed: int |
| + Vehicle(brand: String, speed: int)  + start(): void  + displayDetails(): void |

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

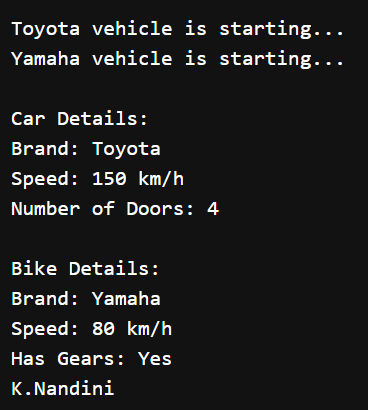
|  |
| --- |
| Bike |
| + hasGears: boolean |
| + Bike(brand: String, speed: int, hasGears: boolean) +displayDetails(): void |

**CODE:**

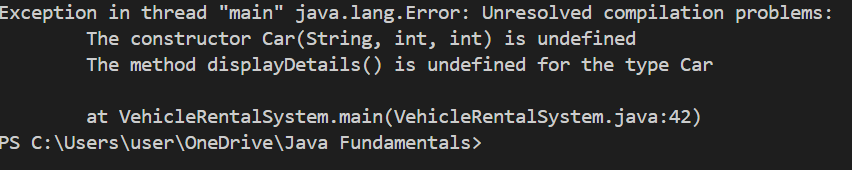
****



**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 |

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**:

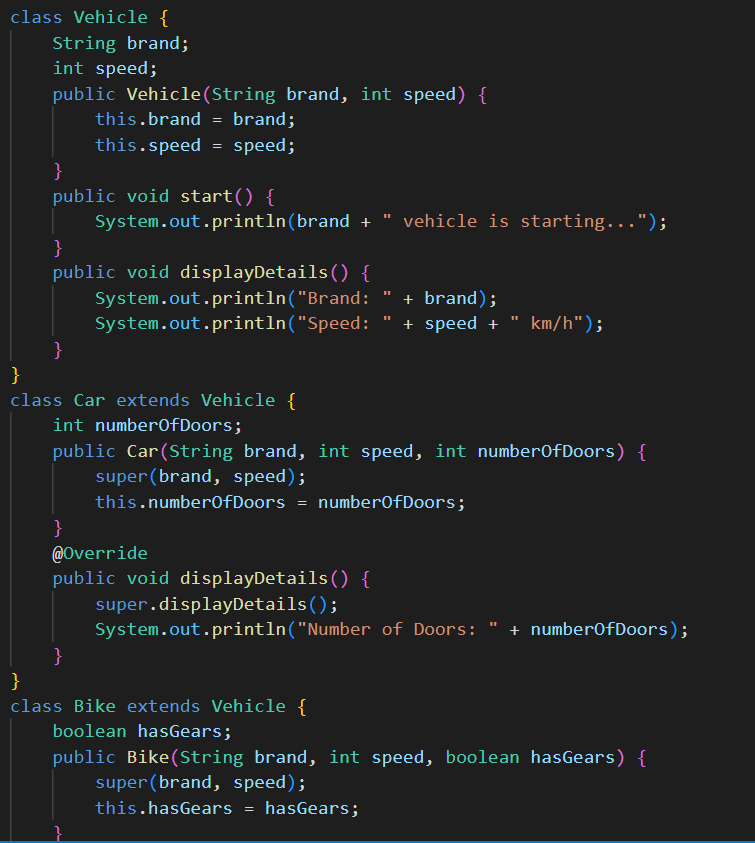
|  |
| --- |
| Vehicle |
| + brand: String  + speed: int |
| + Vehicle(brand: String, speed: int)  + start(): void  + displayDetails(): void |

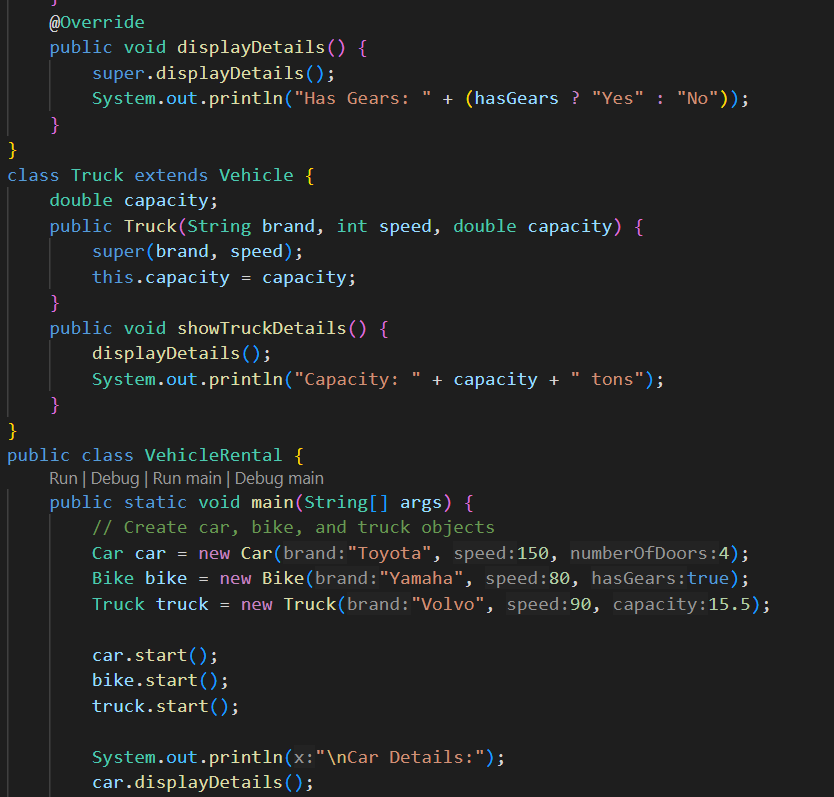
|  |
| --- |
| Car |
| +numberOfDoors: int |
| +Car(brand:String, speed: int, numberOfDoors: int)  + displayDetails(): void |

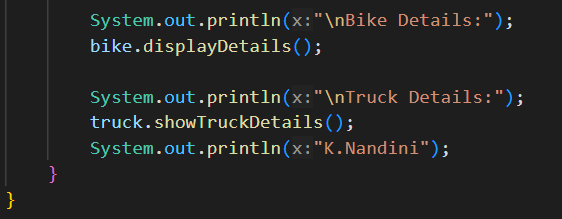
|  |
| --- |
| Bike |
| + hasGears: boolean |
| + Bike(brand: String, speed: int, hasGears: boolean) +displayDetails(): void |

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

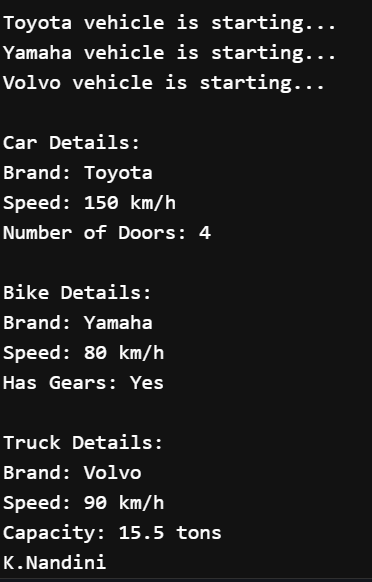
**CODE:**



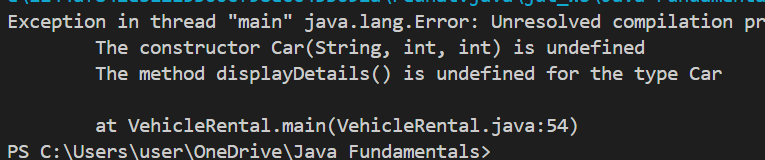




**POSITIVE CASE:**



**NEGATIVE CASE:**

**ERROR TABLE:**

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

**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.