23CSE111

OBJECT-ORIENTED PROGRAMMING

LAB REPORT



Department of Computer Science and Engineering

Amrita School of Engineering

Amrita Vishwa Vidyapeetham, Amaravati Campus

Verified by: NAME: G.Sahithi

ROLL NO:AV.SC.U4CSE24122

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No** | **Title** | Date | Page No. | Signature |
| Week 1 |  |  |  |  |
| 1. | How to download and install Java Software. |  |  |  |
| 2. | Write a Java Program to print the message “Welcome to Java Programming”. |  |  |  |
| 3. | Write a Java program that prints: Name, Roll.no. section of a student. |  |  |  |
| Week 2 |  |  |  |  |
| 1. | Write a java program to calculate the area of a rectangle. |  |  |  |
| 2. | Write a java program to temperature from Celsius to Fahrenheit and vice-versa. |  |  |  |
| 3. | Write a java program to calculate the simple interest. |  |  |  |
| 4. | Write a java program to find the largest of three numbers, using ternary operator. |  |  |  |
| 5. | Write a java program to find the factorial of a number. |  |  |  |
| Week 3 |  |  |  |  |
|  | To create a java program with the following instructions:   1. Create a class name “car” 2. Create 4 attributes named car\_color, car\_brand, fuel\_type and mileage 3. Create 3 methods names start (), stop () and service (). 4. Create 3 objects named car1, car2 and car3 5. Create a constructor which should print “welcome to car garage”. |  |  |  |
|  | To write a java program to create a class named BankAccount, with 2 methods deposit () and withdraw ().   1. Deposit (): Whenever an amount is deposited, it has to be updated the current amount. 2. Withdraw (): Whenever an amount is withdrawn, it has to be less than the current amount, else print (“Insufficient funds”). |  |  |  |
| Week 4 |  |  |  |  |
| 1. | Write a java program with class named “book”. The class should contain various attributes such as “title\_of\_the\_book”, “author”, “year\_of\_publication”. It should also contain a constructor with the parameters which initializes “title\_of\_the\_book”, “author”, “year\_of\_publication”. Create a method which displays the details of the book, that is “author”, “title\_of\_the\_book”, “year\_of\_publication”. Display the details of two books by creating two objects. |  |  |  |
| 2. | To create a java program with class name “my\_class” with a static variable “count” of “int” type, initialized to zero and a constant variable “pi” of type double initialized to “3.1415” as attributes of that class. Define a constructor for “my\_class” that increments the count variable each time an object of my\_class is created. Finally print the values of “count” and “pi” |  |  |  |

WEEK 1

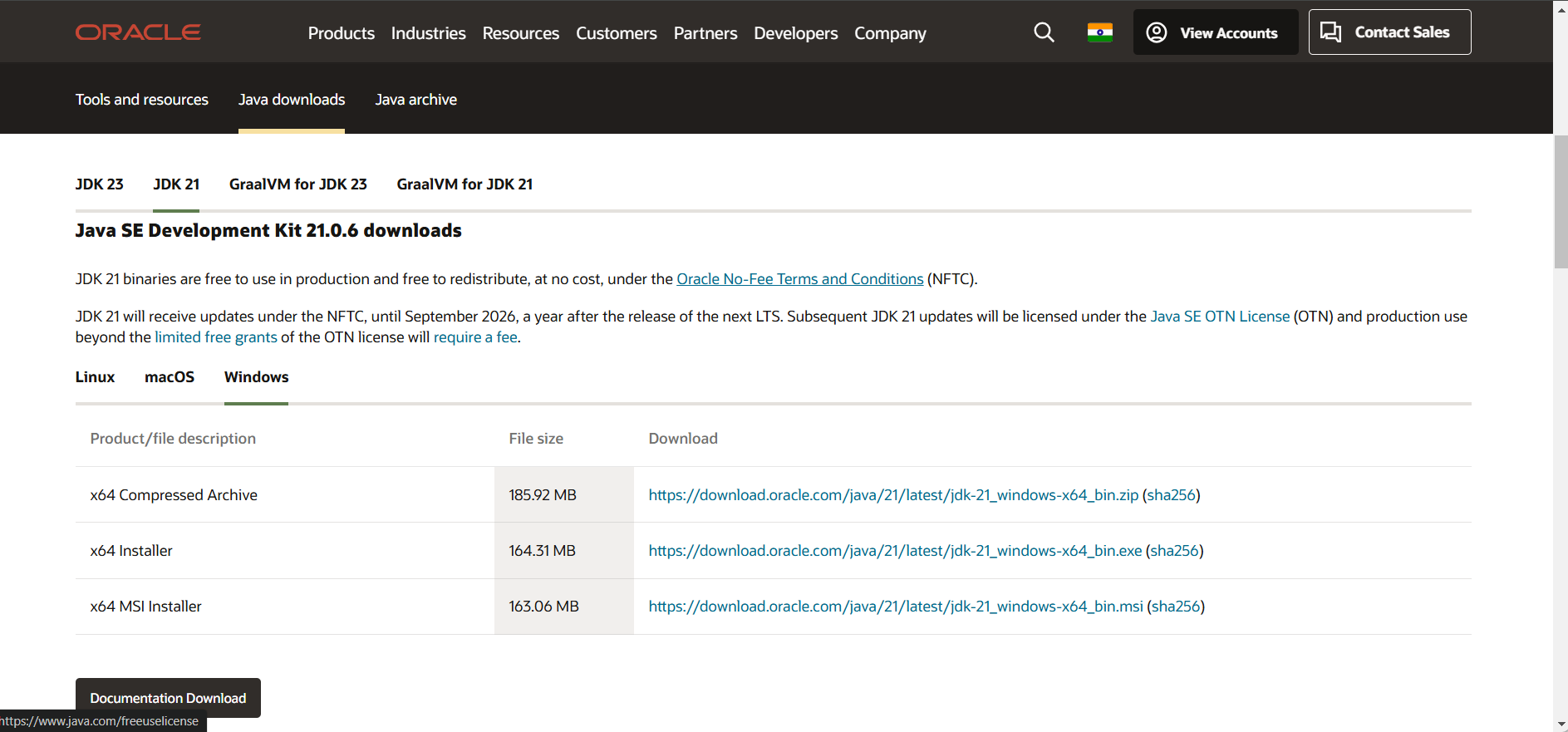
PROGRAM – 1

AIM: How to download and install java software

PROCEDURE: The procedure for downloading the software is written below.

Step 1: Downloading JDK21

1. Open your default web browser and go to the search bar and type Oracle JDK Downloads page.
2. Scroll down and click on Java SE Development Kit21 section.
3. Choose “Windows x64” Installer version.
4. Click on Downloads and wait for the whole installation to get completed.

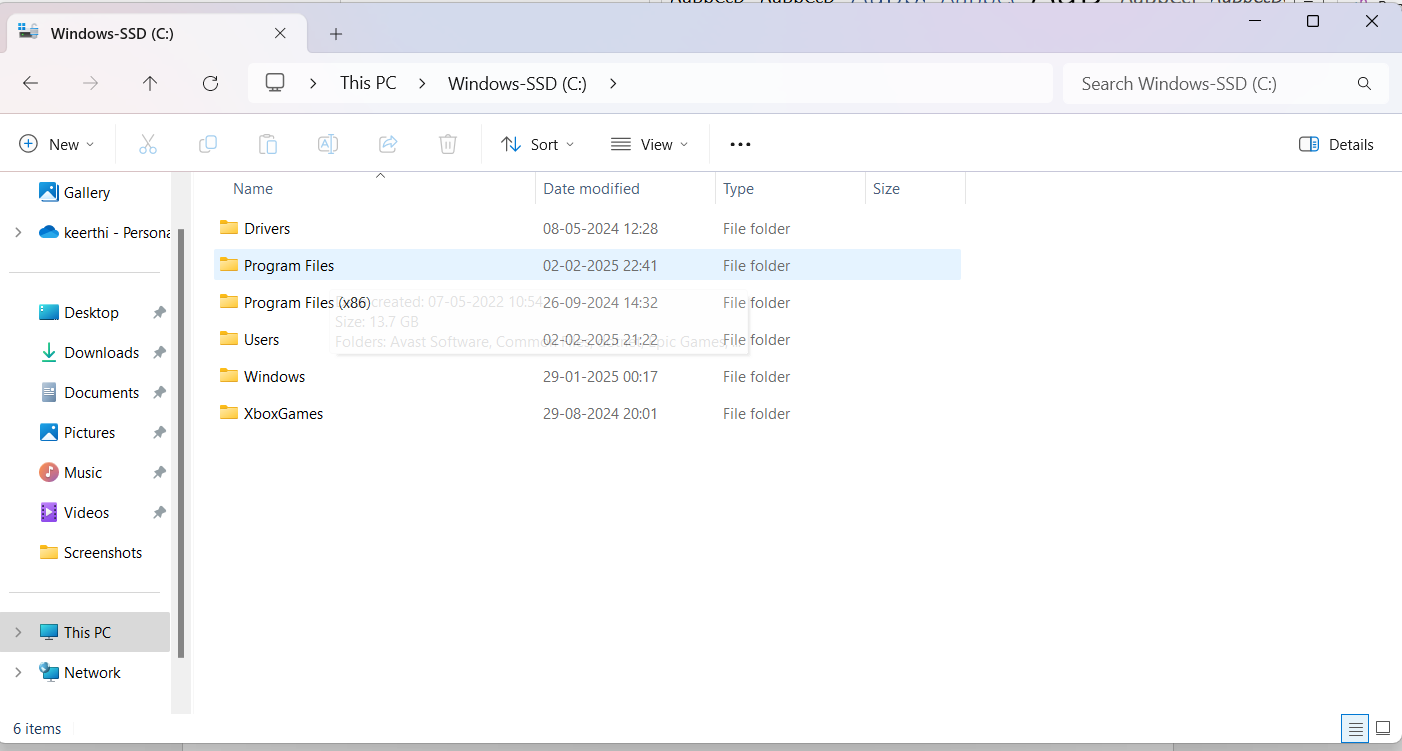


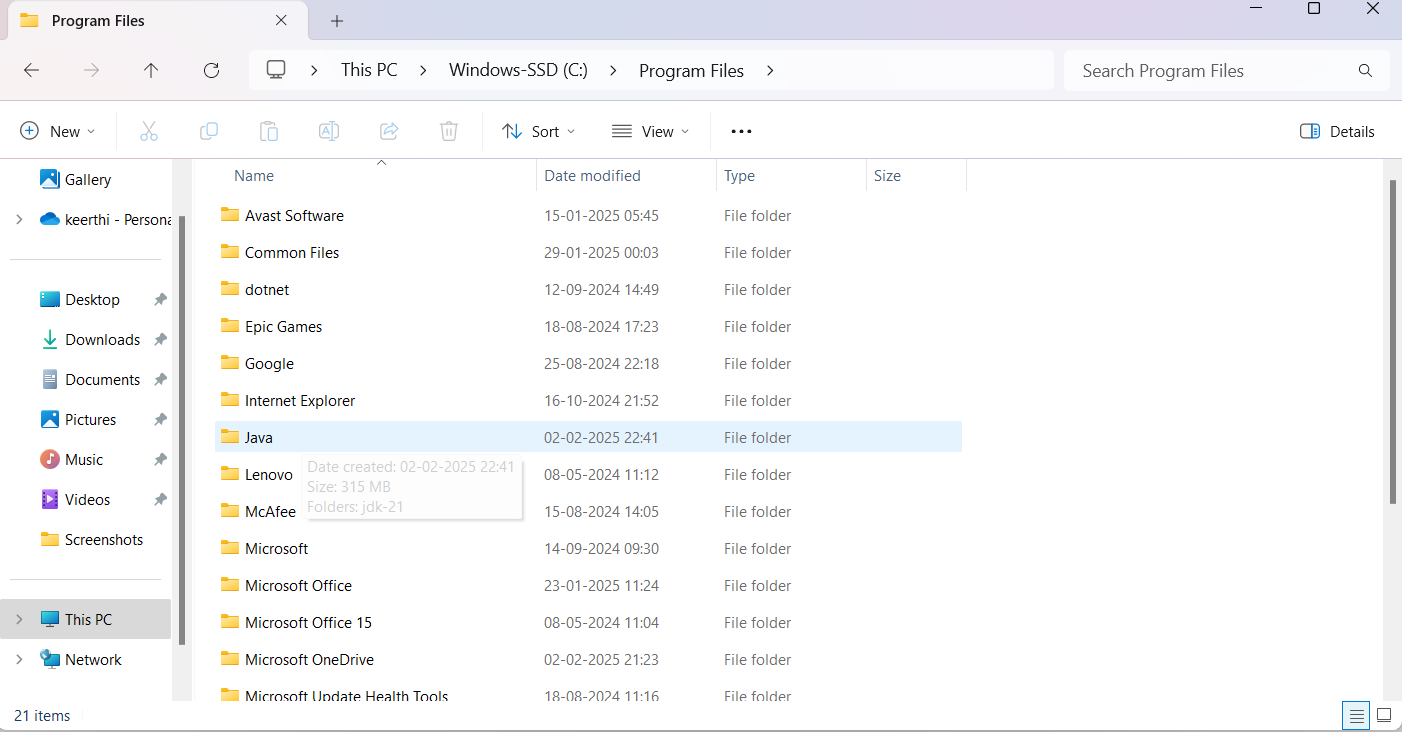
Step 2: Installing of JDK21

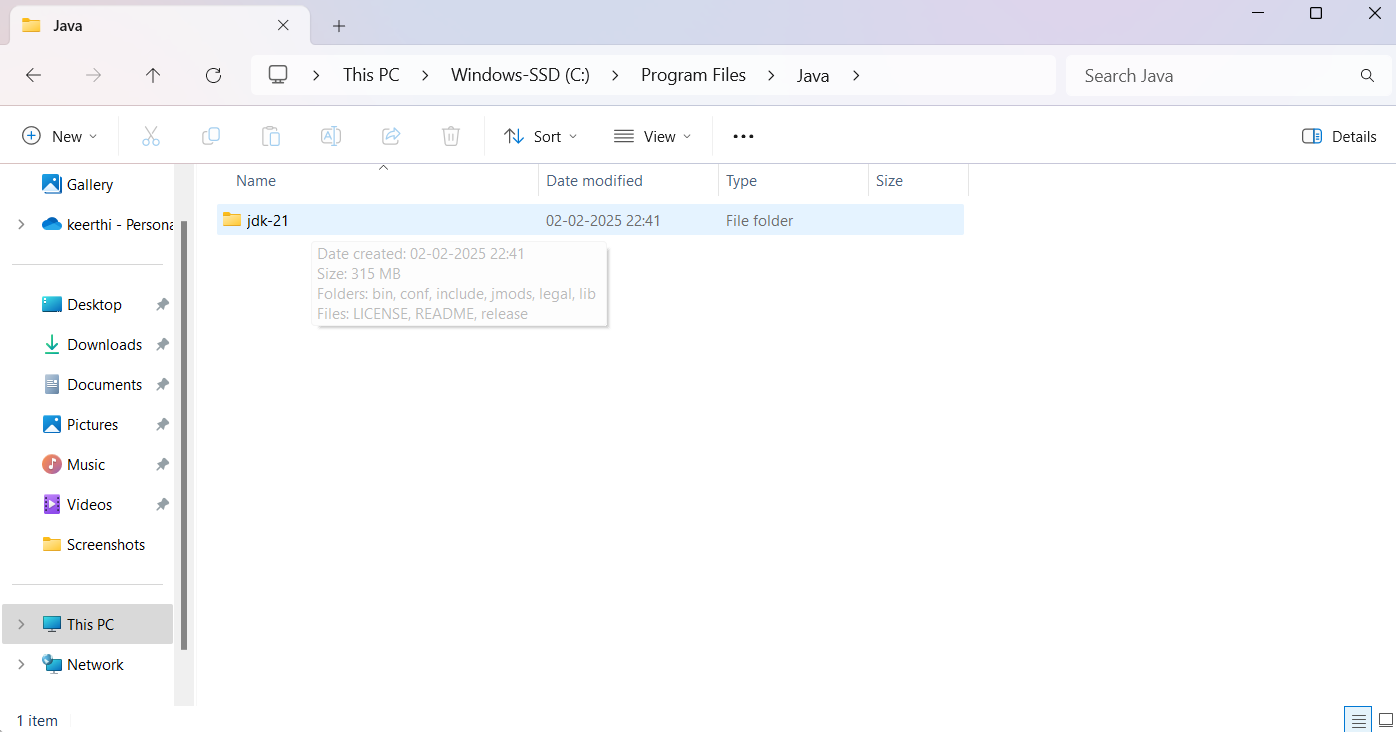
1. Locate the downloaded document jdk-21\_windows-x64\_bin.exe file.
2. Double-click on it and launch the installer.
3. Click on the next simultaneous steps.
4. Choose the installation path (C:/ProgramFiles/java/jdk-21).
5. Click on the next step and install it.
6. Wait for the process of installation to get completed.
7. Click on the close button and the whole installation is completed.

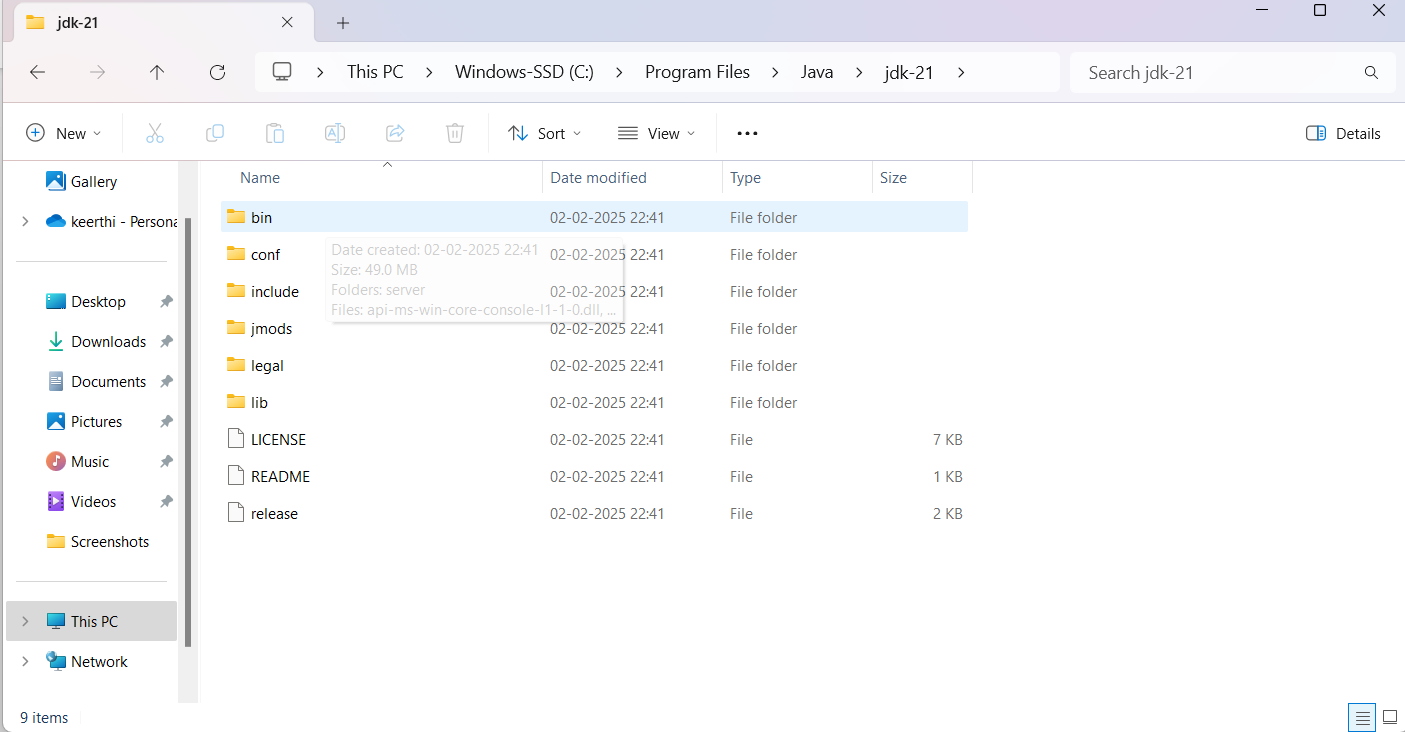
Step 3: Setting up the path of Java program

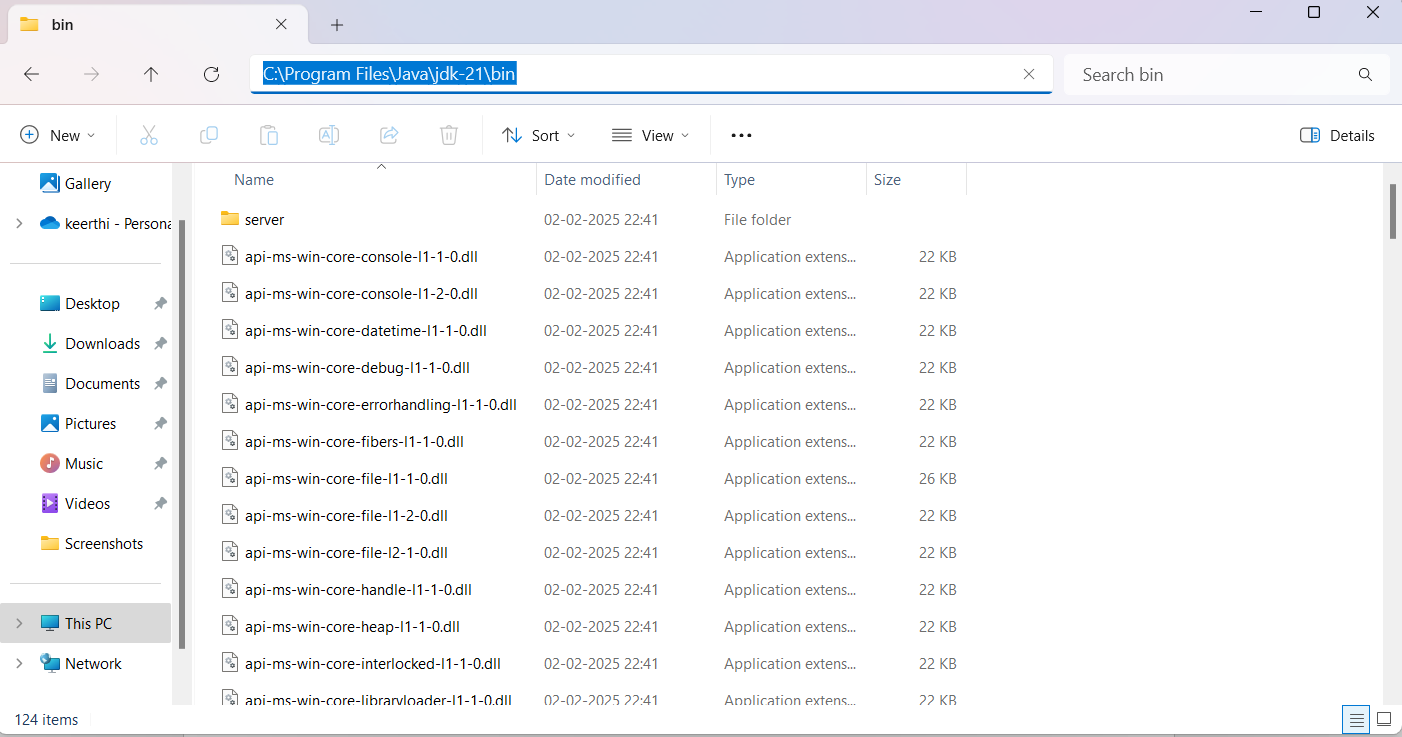
1. Access the “windows c: drive” on your laptop.
2. Choose the program files option and click on java, then click on JDK version 21, and then select the bin option.
3. Copy the path address at the top.





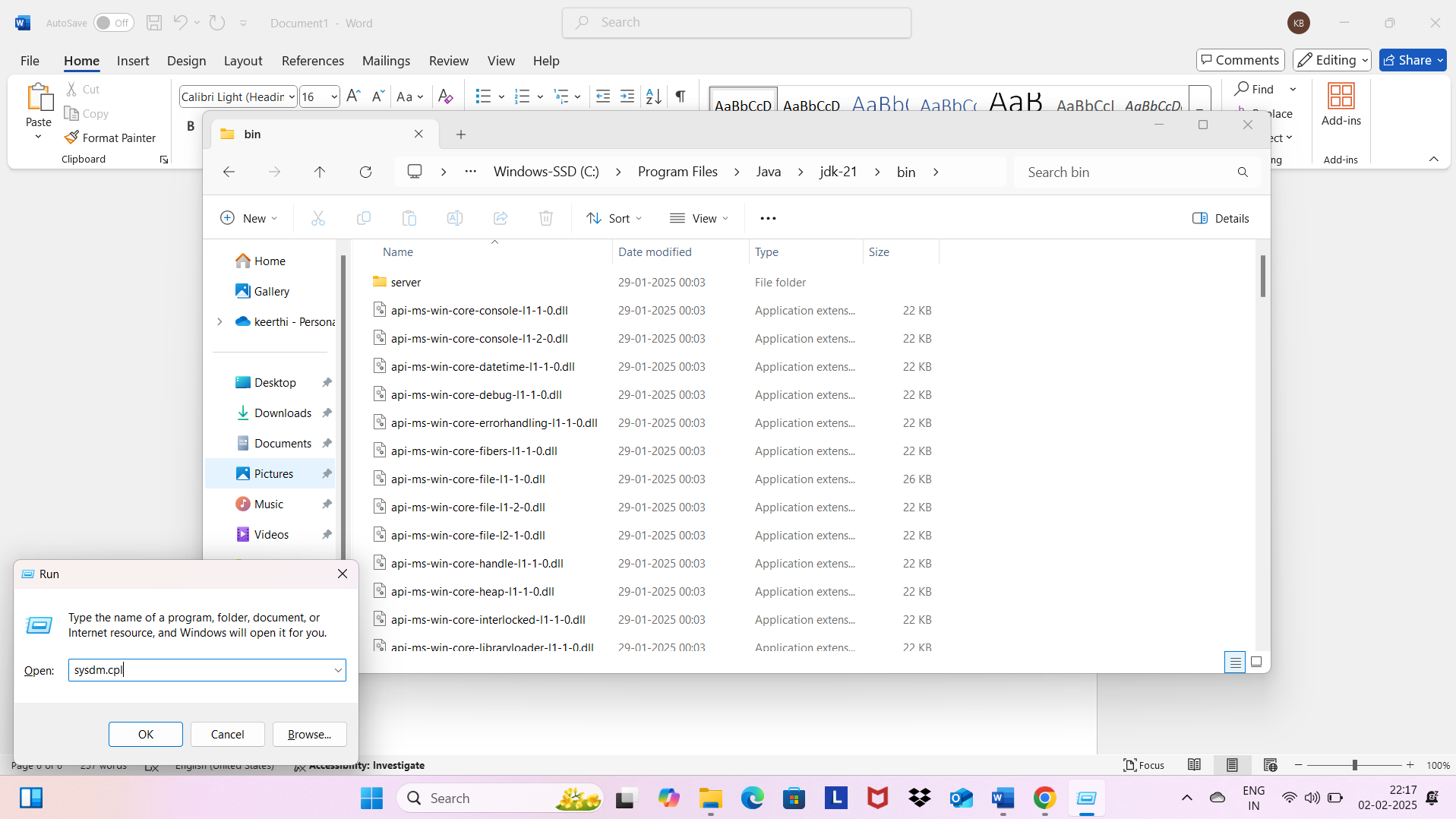


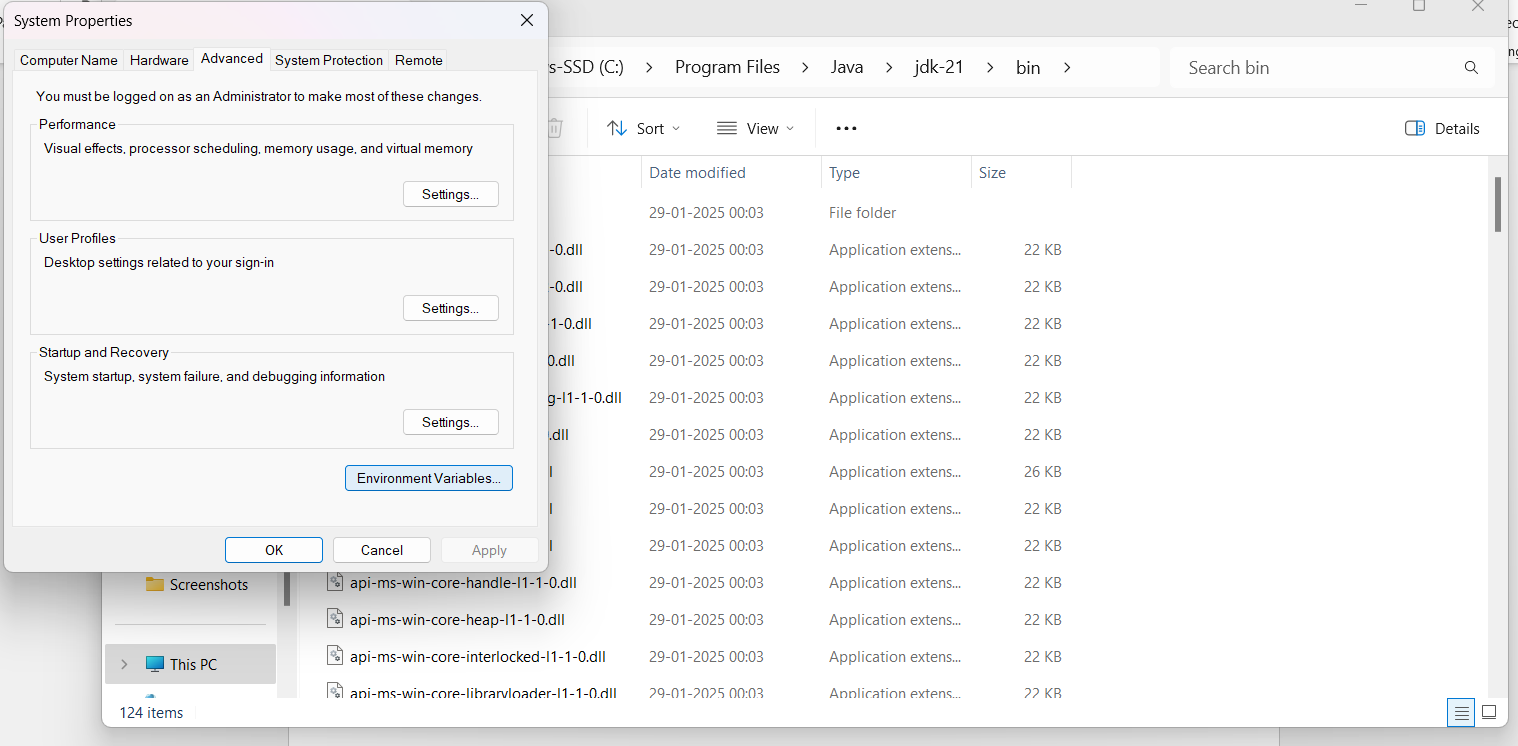




Step 4: Accessing the system properties.

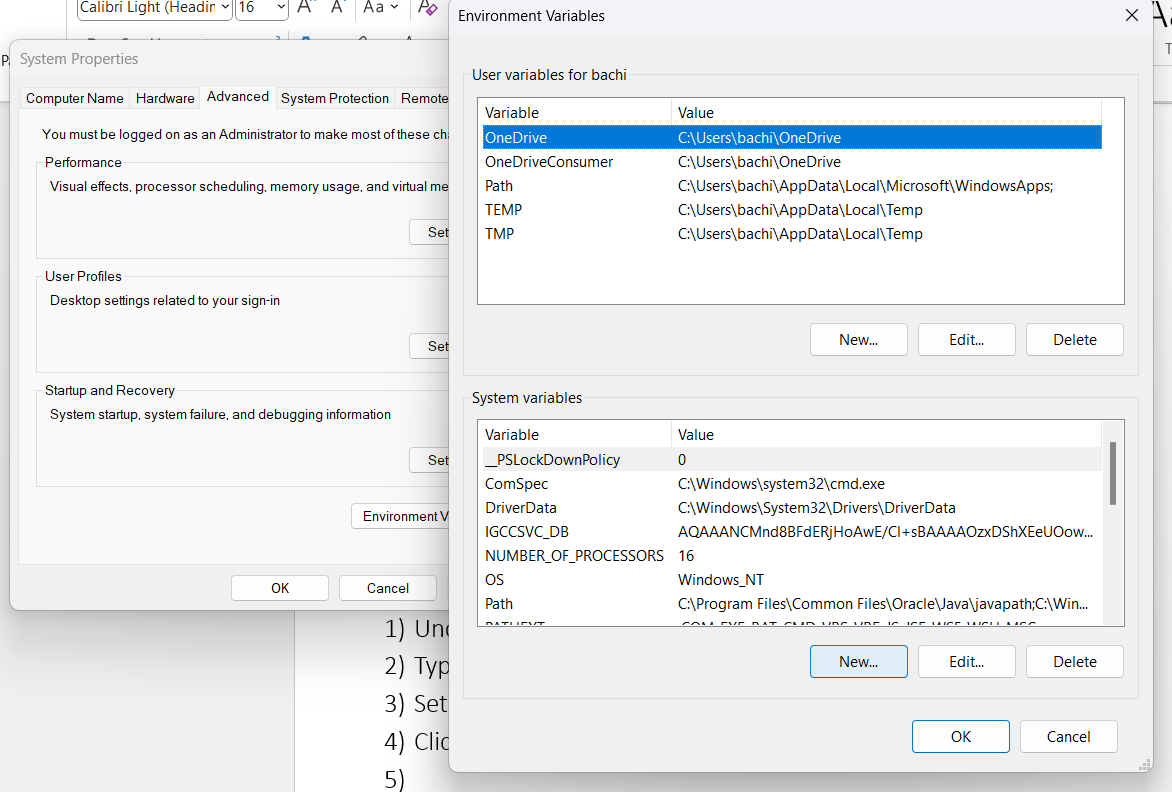
1. Press the windows + R option and type sysdm.cpl, and click on the option “ok”.
2. The system properties will open.
3. Then click on the Advanced tab.
4. Go to the environment variables present at the bottom.

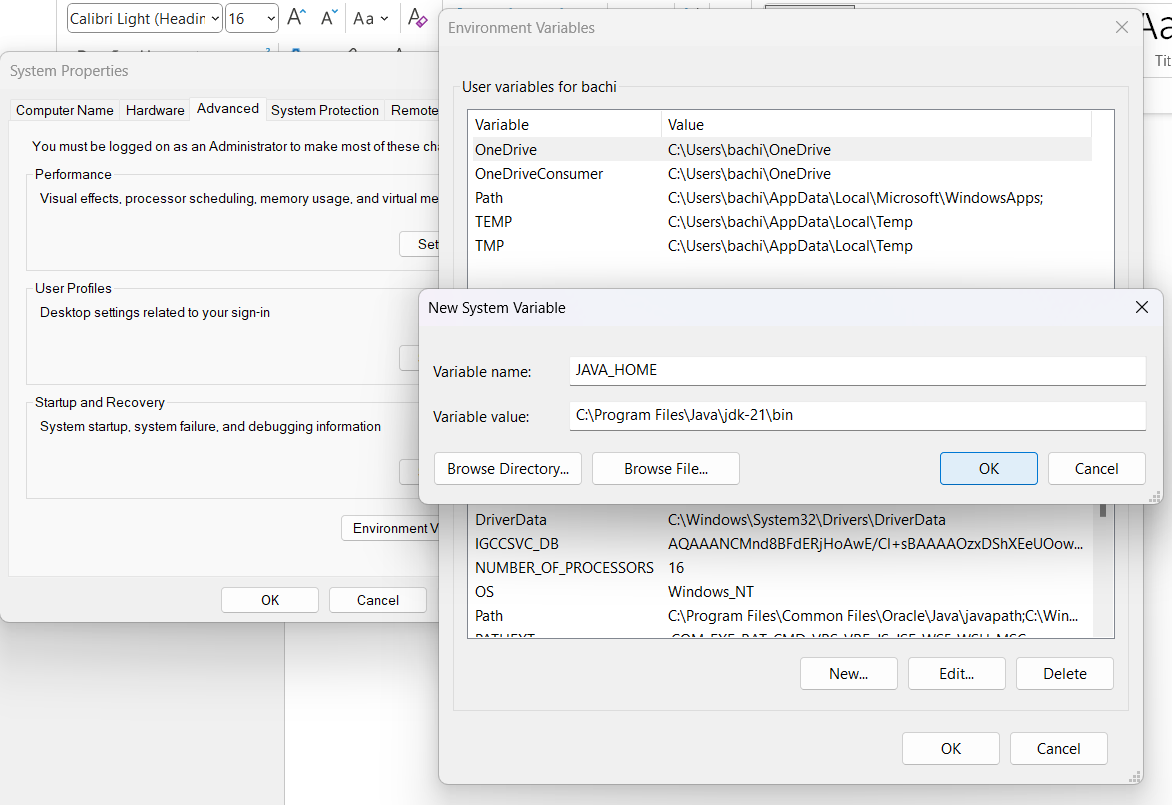




Step 5: Setting up JAVA\_HOME

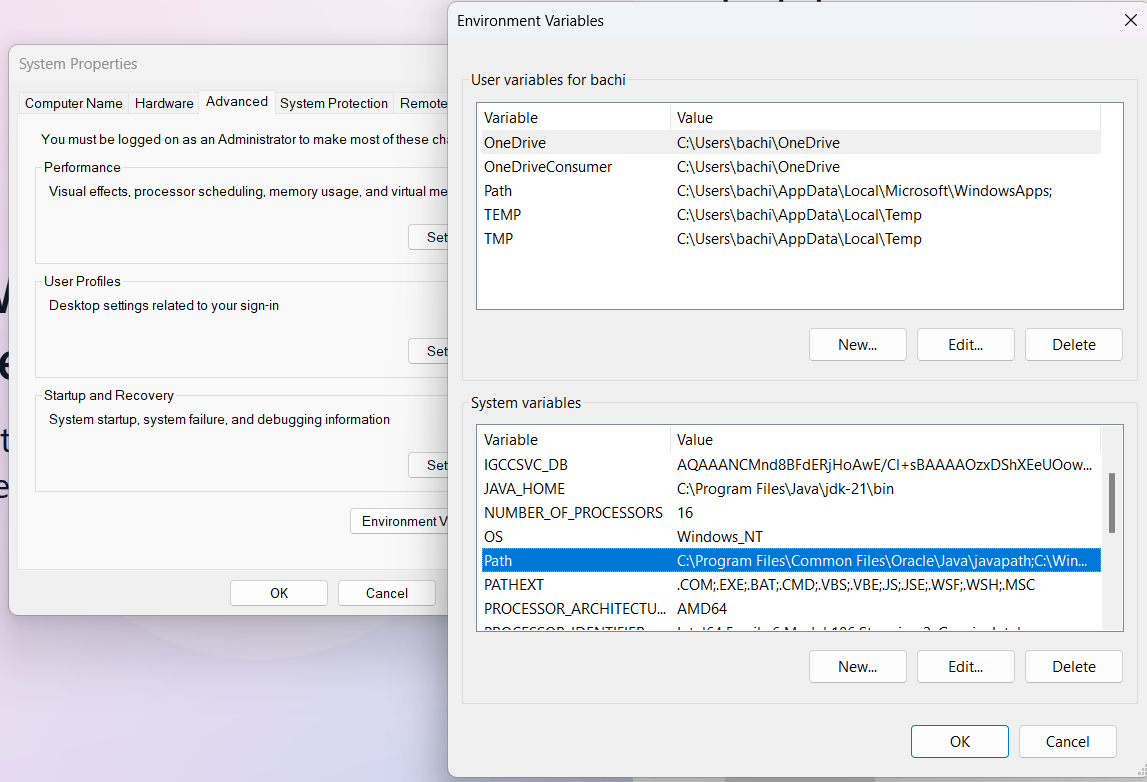
1. Under the system variables option click on the option “new”.
2. Type out the name as JAVA.HOME.
3. Set the variable value as the installation path.
4. Click on the function “ok”.

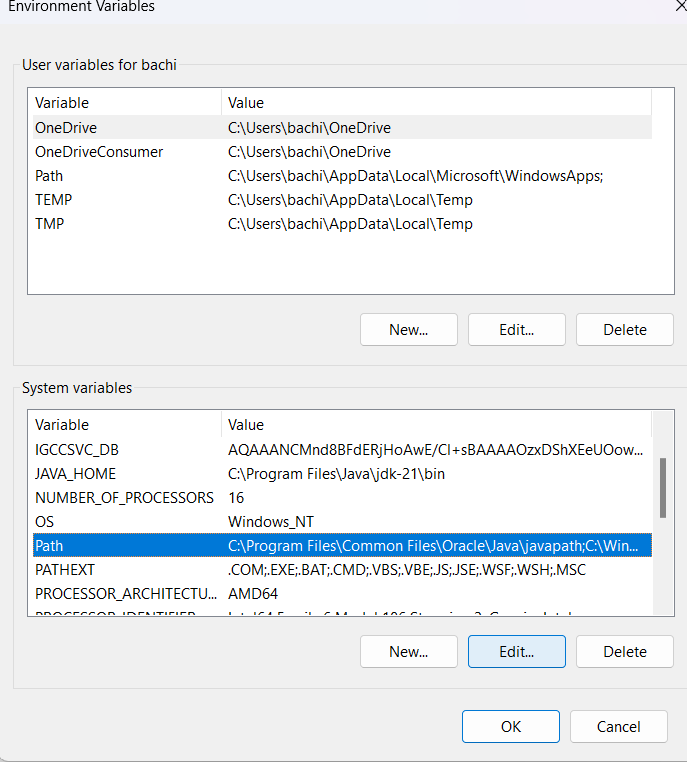
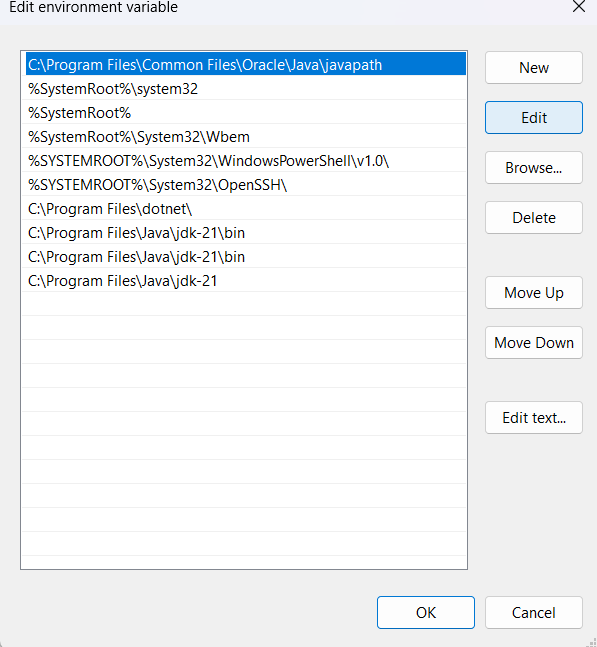


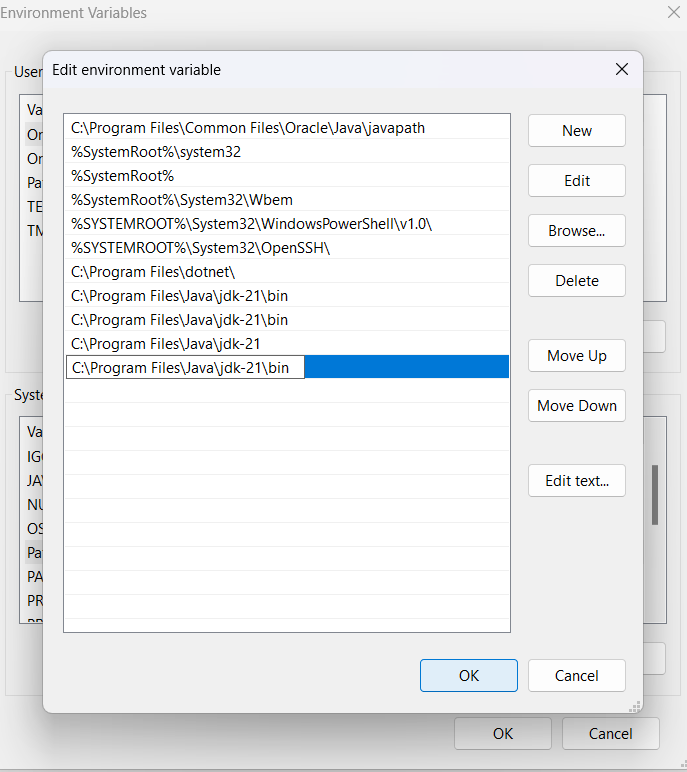


Step 6: Updating the PATH variable.

1. In the system variable option, find the path and click on edit option.
2. Click on the new option and add C:\Program Files\Java\jdk-21\bin.
3. Click on OK to save.

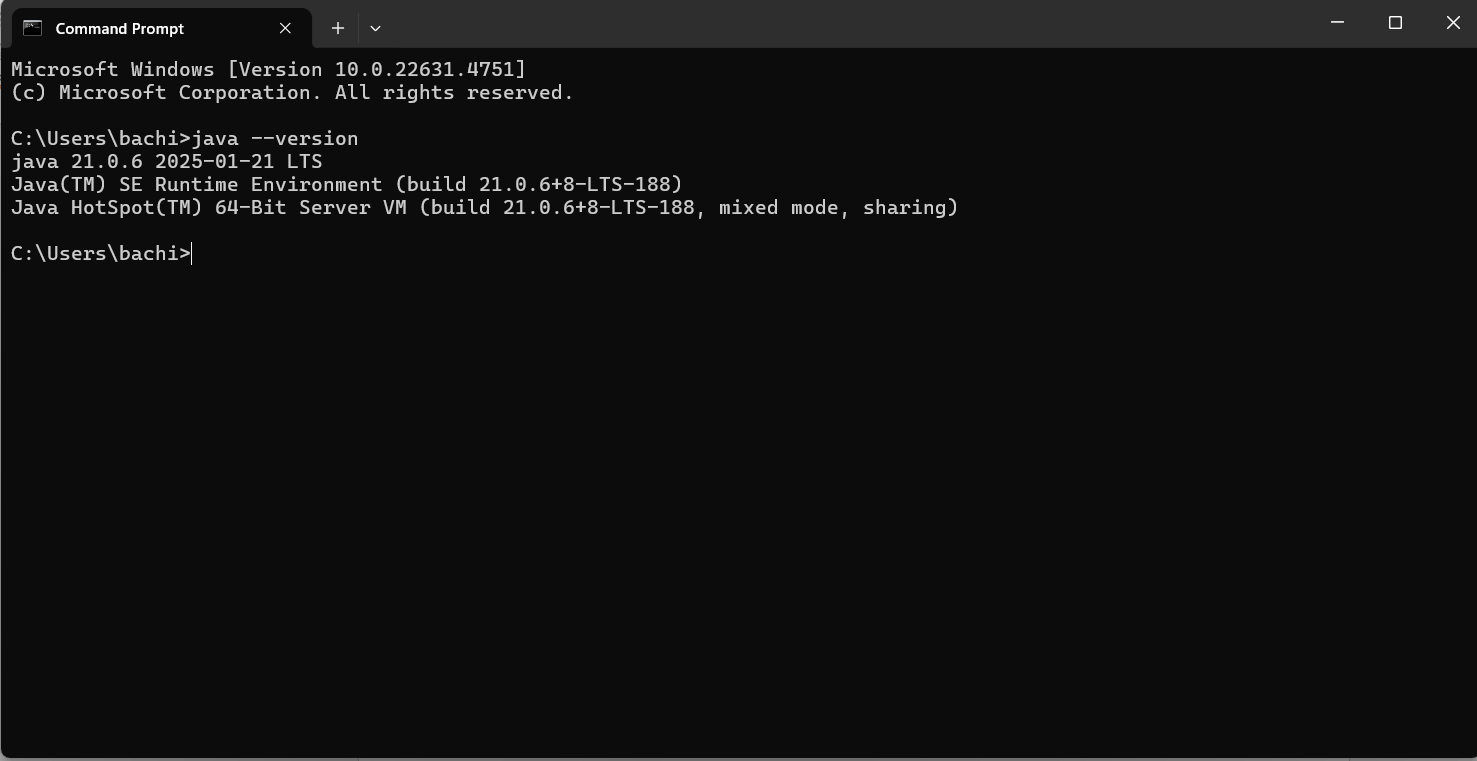


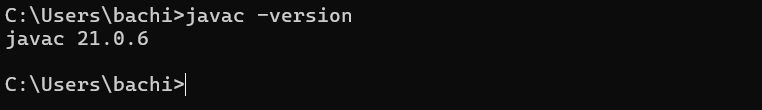


Step 7: verifying the process of installation.

1. Click on the command prompt in your system.
2. Type out the command, **java --version** and click on enter.



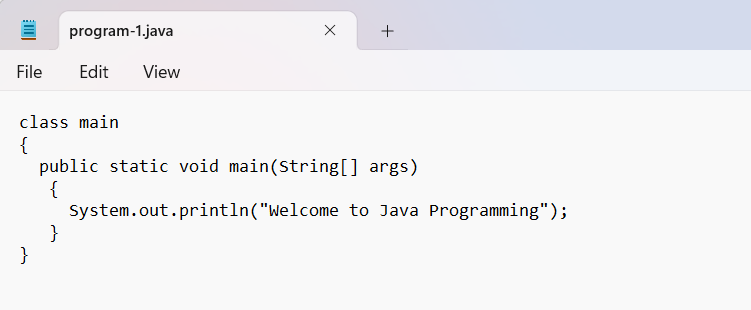
1. We can also check the compiler version by typing out the following command.



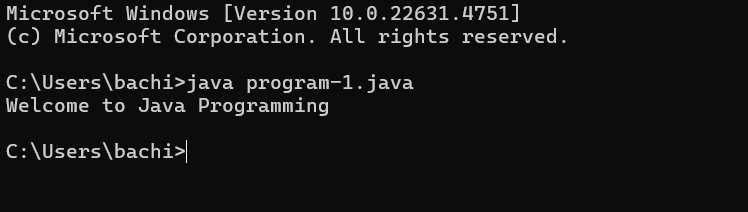
PROGRAM 2:

AIM: To write a java program and print the message “Welcome to Java Programming”.

INPUT:



OUTPUT:

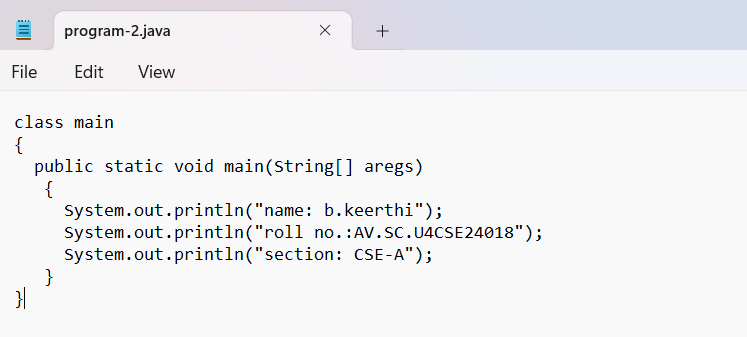


ERRORS: No errors were to be found.

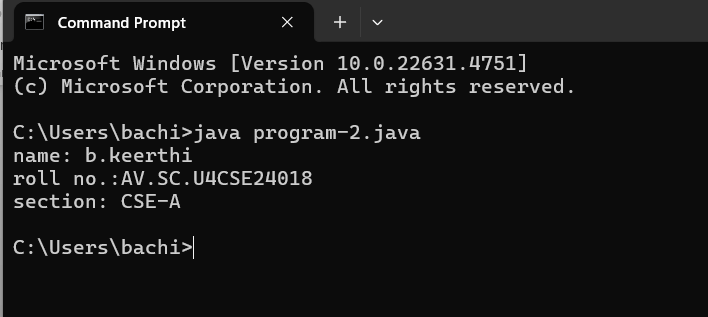
PROGRAM 3:

AIM: Writing of a java program to print out name, roll no., and section of a student.

INPUT:



OUTPUT:



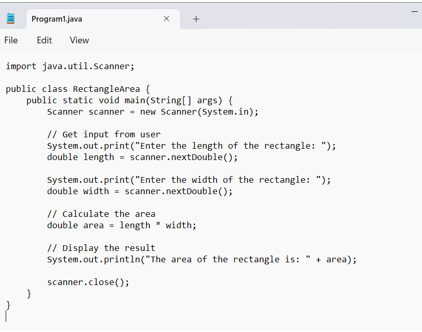
ERRORS: No errors were found as of the program.

WEEK 2

PROGRAM – 1:

AIM: write a java program to calculate the area of the rectangle.

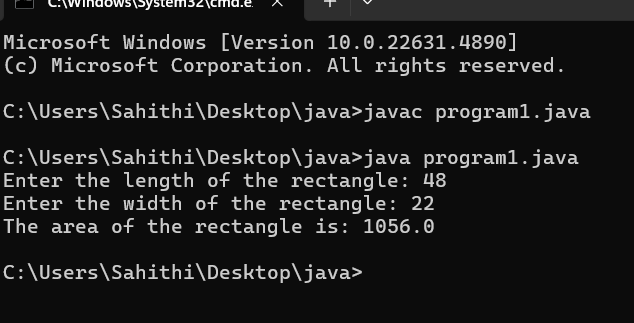
INPUT:



ERRORS:

|  |  |  |
| --- | --- | --- |
| **Sno.** | **Error message** | **Error rectification** |
| 1. | Error: <identifier> expected. Public static void main. Compilation failed | Public Static void main (String [ ] args) |

OUTPUT:



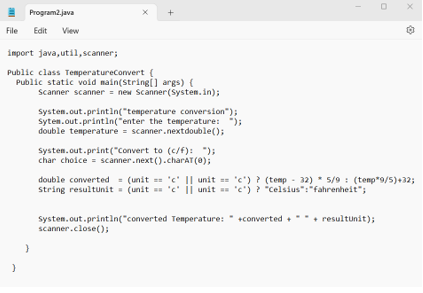
CONCEPTS KNOWN:

* Import java.util.Scanner – used to accept inputs from the user, under the util package has to be imported.
* Scanner input=new Scanner(System.in); - Used to create a Scanner object
* int ln=input.nextInt(); - Used to read the integer data type stored under the object created
* System.out.println(“ “); - It is used to print string inside the quotes. After printing, the cursor moves to the beginning of the next line.

PROGRAM 2

AIM: Write a java program to convert temperature from Celsius to Fahrenheit and vice-versa.

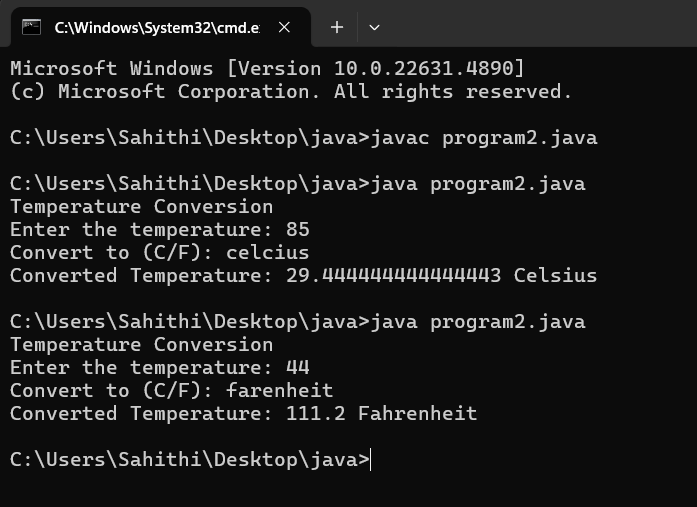
INPUT:



ERRORS:

|  |  |  |
| --- | --- | --- |
| **SNO.** | **Error message** | **Error rectification** |
| 1. | Error: ‘. ‘expected | Remove the ‘ , ‘ symbol and replace with the correct symbol |
| 2. | Error: class, interface, enum, or record expected | Static is written correctly |
| 3. | Error: <identifier> expected | Static is written correctly |

OUTPUT:



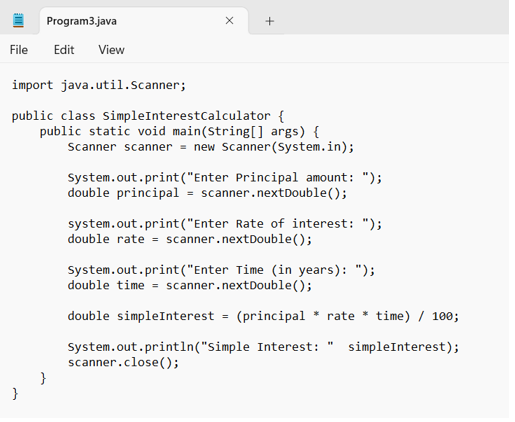
CONCEPTS KNOWN:

* import java. util. Scanner; - To accept input from user, Scanner class under util package has to be imported.
* Scanner input=new Scanner (System.in); - Used to create a Scanner object
* double fh=input. next Double (); - Used to read double data type stored under the object created
* System.out.println(““); - It is used to print string inside the quotes. After printing, the cursor moves to the beginning of the next line.

PROGRAM-3

AIM: Write a java program to calculate the simple interest.

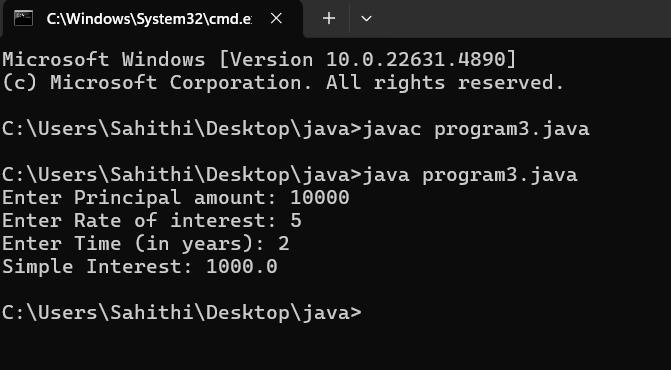
INPUT:



ERRORS:

|  |  |  |
| --- | --- | --- |
| **SNO** | **Error message** | **Error rectification** |
| 1. | Error: ‘) ‘or ‘, ‘is expected | Error rectified at line 5 |
| 2. | Error: not a statement | Statement has been verified at line 5 |
| 3. | Error: ‘; ‘is expected | Error rectified at line 6 |

OUTPUT:



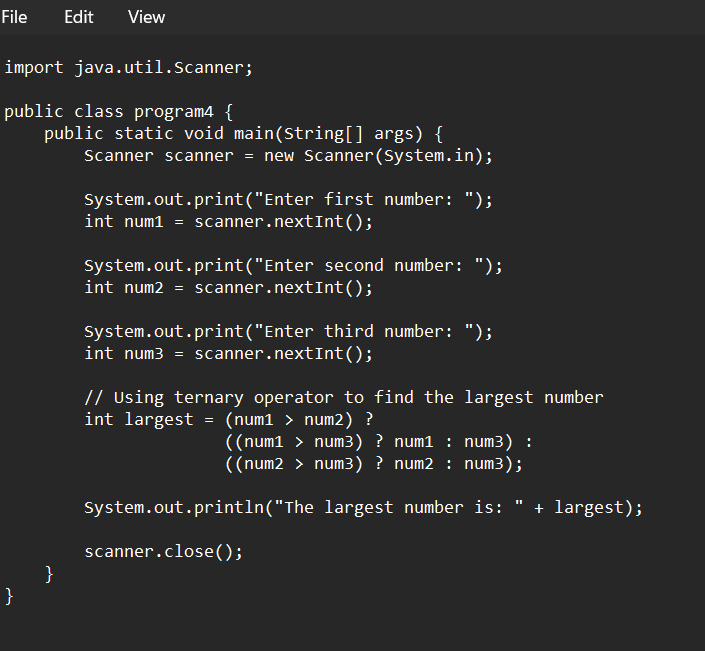
CONCEPTS TO KNOW:

* import java. util.Scanner; - To accept input from user, Scanner class under util package has to be imported.
* Scanner input=new Scanner (System.in); - Used to create a Scanner object
* double p=input. Next Double (); - Used to read double data type stored under the object created
* System.out.println (““); - It is used to print string inside the quotes. After printing, the cursor moves to the beginning of the next line.

PROGRAM 4:

AIM: Write a java program to find the largest of three numbers, using ternary operator

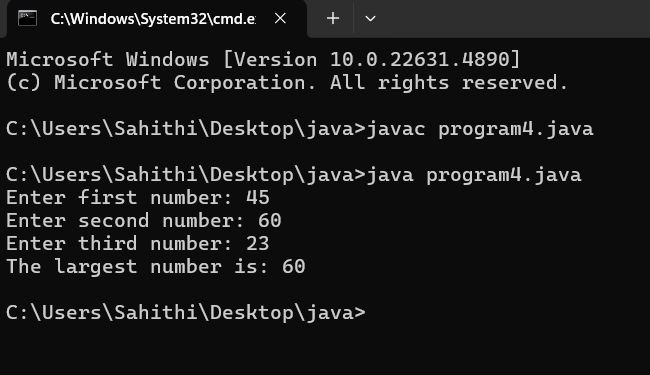
INPUT:



ERRORS:

|  |  |  |
| --- | --- | --- |
| **SNO** | **Error message** | **Error rectification** |
| 1. | Error: package ava.util does not exist | Error rectified and changed to java. util |
| 2. | Error: cannot find the symbol | A symbol “= “which is missed is written again. |
| 3. | Error: cannot find the symbol | Missed symbol is replaced. |
| 4. | Error: package system does not exist | A symbol ‘. ‘Which has been missed is written. |
| 5. | Error: cannot find symbol | The symbol ‘; ‘is written |

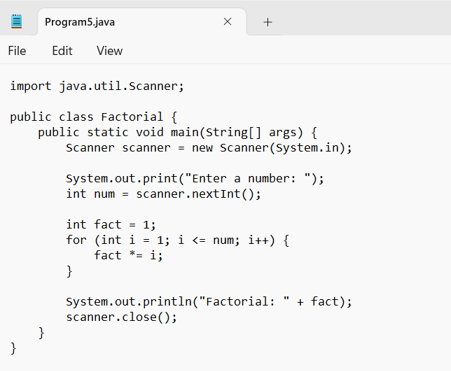
OUTPUT:



CONCEPTS KNOWN:

* import java.util.Scanner; - To accept input from user, Scanner class under util package has to be imported.
* Scanner input=new Scanner(System.in); - Used to create a Scanner object
* int a=input.nextInt (); - Used to read integer data type stored under the object created
* int result=(a>b)? ((a>c)? a:c): ((b>c)? b:c); - Nested Ternary operator is used here.
* Syntax for ternary operator is- condition? expression 1: expression 2; whose answer is stored in a variable and then used.

PROGRAM 5:



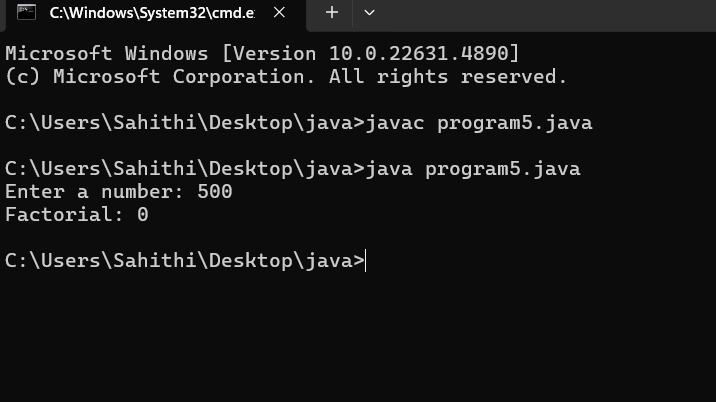
AIM: Write a java program to find the factorial of a number.

INPUT:

ERRORS:

|  |  |  |
| --- | --- | --- |
| **SNO.** | **Error message** | **Error rectification.** |
| 1. | Error: class, interface, enum, or record is expected. | The word factorial has been replaced |
| 2. | Error: unnamed classes are a preview feature and are disabled by default. | The word “static” has been replaced with “Static” |
| 3. | Error: class, interface, enum, or record expected | The symbol ‘} ‘has been removed. |

OUTPUT:



CONCEPTS KNOWN:

* for (int i=1; n>=i; --n) {} - For loop syntax: for (initial expression; test expression; update expression {} The loop is executed, until the test expression evaluates to be false

WEEK – 3

PROGRAM 1:

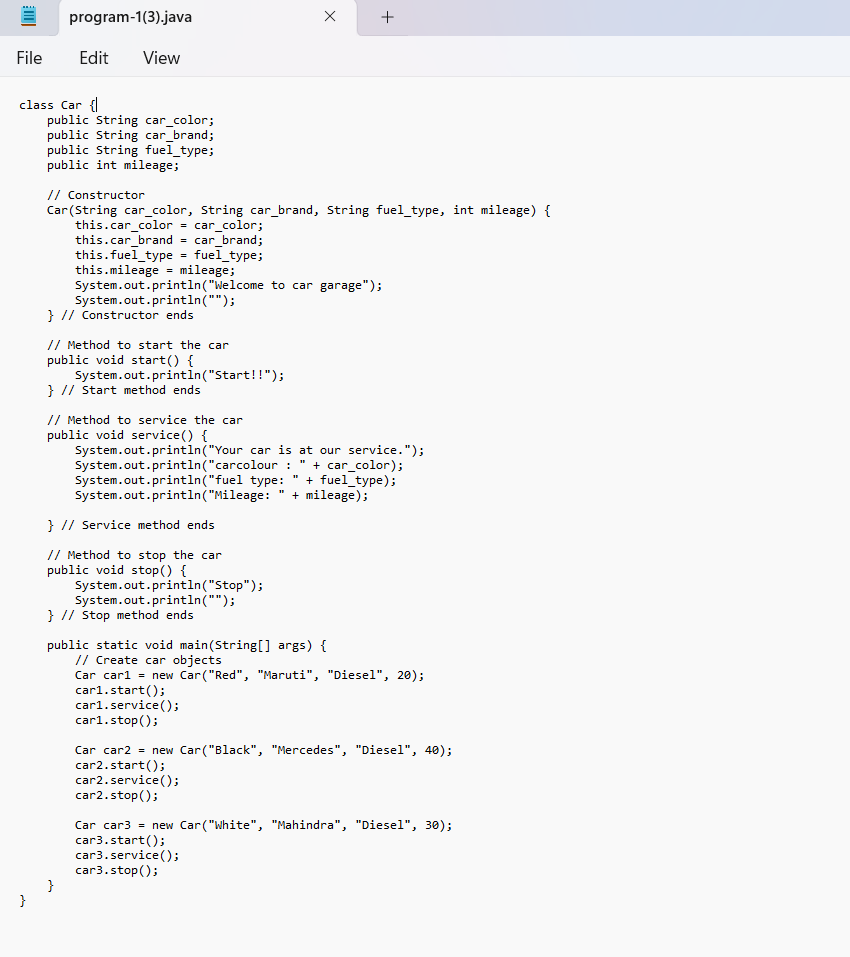
AIM: To create a java program with the following instructions:

1. Create a class with name “Car”
2. Create 4 attributes, named: car\_color, car\_brand, fuel\_type, mileage
3. Create 3 methods, named: start (), service (), stop ()
4. Create 3 objects, named: car1, car2, car3
5. Create a constructor, which should print, “Welcome to car garage”.

CLASS DIAGRAM:

|  |
| --- |
| Car |
| + car\_color: String  + car\_brand: String  + fuel\_type: String  + mileage: int |
| + Car (): void  + start (): void  + service (): void  + stop (): void |

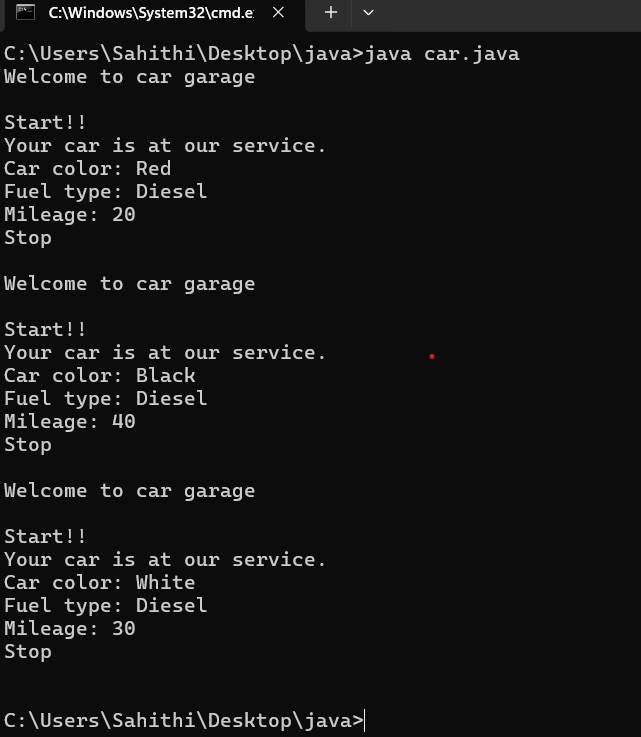
INPUT:



ERRORS:

|  |  |  |
| --- | --- | --- |
| Sno. | Error message | Error rectification |
| 1. | error: ';' expected car1.start() | Add a “;”  car1.start(); |
| 2. | error: illegal start of type  public void stop () | Add a “)”  public void stop () { |
| 3. | error: cannot find symbol thiscar\_brand=car\_brand; | Add a “.”  this.car\_brand=car\_brand; |

OUTPUT:



CONCEPTS KNOWN:

* public String car\_color; - Used to declare a variable named car\_color, with data type as String with public accessibility.
* Car (String car\_color, String car\_brand, String fuel\_type, int mileage) { } – It is a constructor (method with name same as class), which requires parameters such as car\_color (String data-type) and so on.
* this.car\_color=car\_color; - “this” is a default method, which is used to point to the instance variables.
* public void start () {} – used to declare a method, which will return nothing(void) in public accessibility.

PROGRAM 2:

AIM: To write a java program to create a class named Bank Account, with 2 methods deposit () and withdraw ().

1. Deposit (): Whenever an amount is deposited, it has to be updated with the current amount.
2. Withdraw (): Whenever an amount is withdrawn, it has to be less than the current amount, else print (“Insufficient funds”)

CLASS DIAGRAM:

|  |
| --- |
| BankAccount |
| - name: String  - Accno: int  - CurrBal: int |
|  |

INPUT:

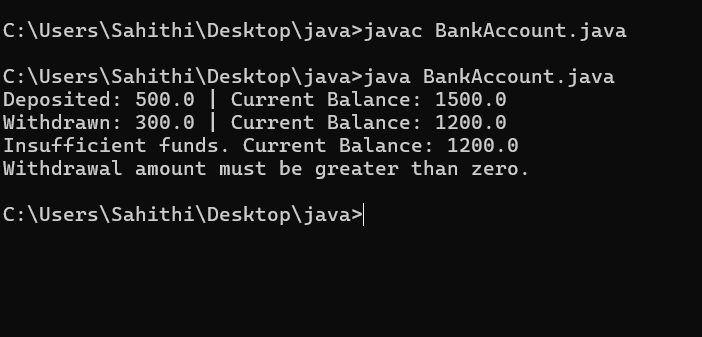
A screenshot of a computer program

AI-generated content may be incorrect.

ERRORS:

|  |  |  |
| --- | --- | --- |
| **Sno.** | **Error message** | **Error rectification** |
| 1. | error: ';' expected cust1.withdraw(3050) | Add a “;”  cust1.withdraw(3050); |
| 2. | error: cannot find symbol thisCurrBal=CurrBal; | Add a “.”  this.CurrBal=CurrBal; |

OUTPUT:



CONCEPTS KNOWN:

* private String name; - Used to declare a variable named name, with data type as String with private accessibility.
* BankAccount (String name, int Accno, int CurrBal) {} – It is a constructor (method with name same as class), which requires parameters such as name (String data-type) and so on.
* this.CurrBal=CurrBal; - “this” is a default method, which is used to point to the instance variables.
* public void withdraws (int WAmt) {} – used to declare a method, which will return nothing(void) in public accessibility, which requires a parameter WAmt (integer data type).
* public int deposit (int DAmt) {} - used to declare a method, which will return integer data type in public accessibility, which requires a parameter DAmt (integer data type).
* BankAccount cust1=new BankAccount("Ram",5587,20000); - used to create an object in class BankAccount, with object name as cust1.
* cust1.withdraw(50000); - Calling a method, under object cust1, by passing a parameter.
* System.out.println("Your current balance after depositing money is:"+cust1.deposit(25000)); - Deposit method will return the value, which will be directly printed

WEEK-4

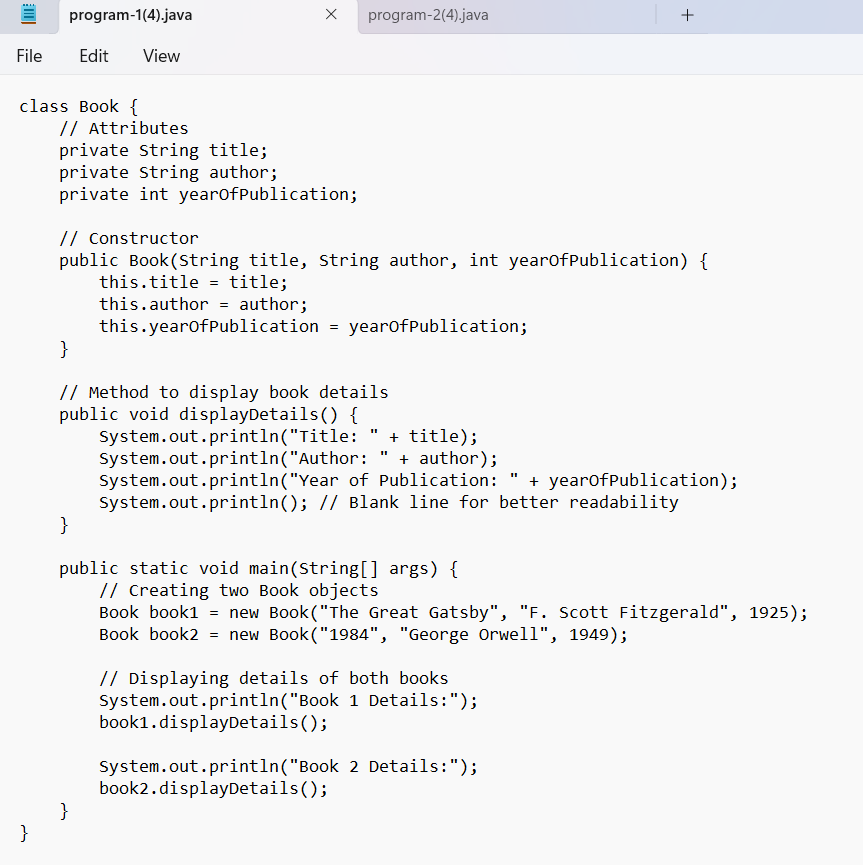
PROGRAM-1:

AIM: Write a java program with class named “book”. The class should contain various attributes such as “title\_of\_the\_book”, “author”, “year\_of\_publication”. It should also contain a constructor with the parameters which initializes “title\_of\_the\_book”, “author”, “year\_of\_publication”. Create a method which displays the details of the book, that is “author”, “title\_of\_the\_book”, “year\_of\_publication”. Display the details of two books by creating two objects.

CLASS DIAGRAM:

|  |
| --- |
| Book |
| - title: String  - author: String  - yearOfPublication: int |
| + Book(title, author, yearOfPublication)  + displayDetails(): void |

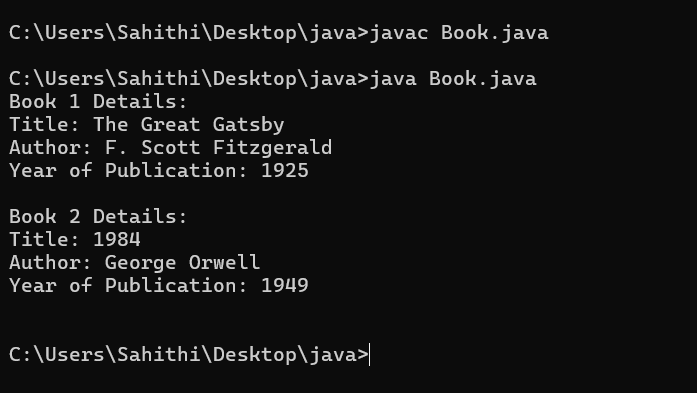
INPUT:

****

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **Sno.** | **Error message** | **Error rectification** |
| 1. | Error: ‘;’ is expected | In line no.16 the required symbol is added. |
| 2. | Error: cannot find the symbol | Line 5 string changed to int and S is capital is string. |
| 3. | Error: ‘)’ and ‘;’ is expected | The space at the last line is removed. |
| 4. | Error: class, interface, enum or record expected | Additional symbol ‘{‘ is removed at the end. |

OUTPUT:



CONCEPTS KNOWN:

* The Book class defines the attributes and behavior of a book.
* Objects (book1 and book2) are created from the Book class.
* The attributes (title, author, yearOfPublication) are declared private to restrict direct access.
* Getter and setter methods could be added to allow controlled access.
* Two objects, book1 and book2, are created using the new keyword.
* The System.out.println() function is used to display book details.

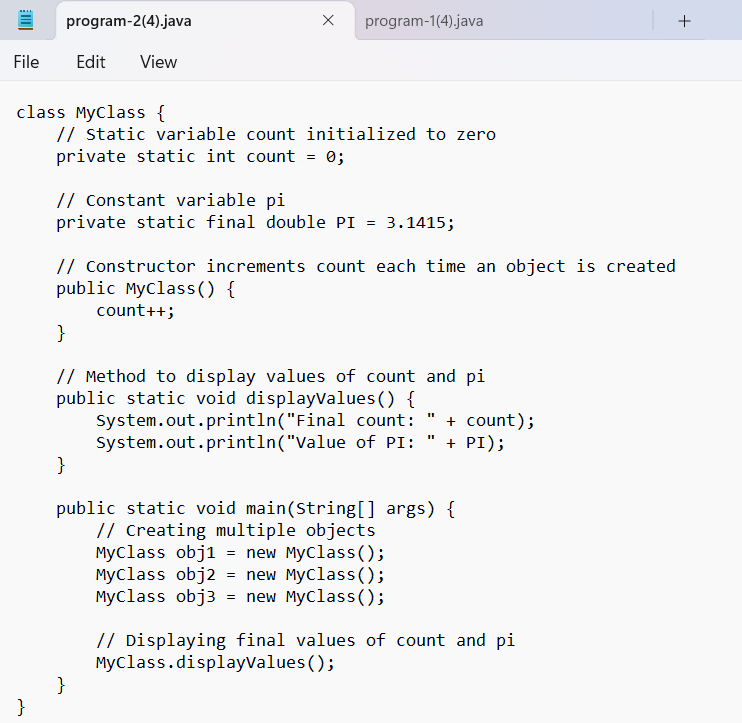
PROGRAM-2

AIM: To create a java program with class name “my\_class” with a static variable “count” of “int” type, initialized to zero and a constant variable “pi” of type double initialized to “3.1415” as attributes of that class. Define a constructor for “my\_class” that increments the count variable each time an object of my\_class is created. Finally print the values of “count” and “pi”.

CLASS DIAGRAM:

|  |
| --- |
| Myclass |
| - count: int  - pi: double |
| + Myclass()  + display(): void |

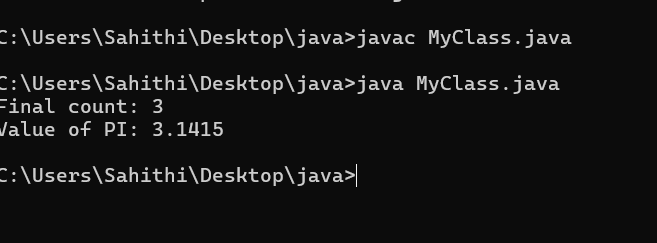
INPUT:



ERRORS:

|  |  |  |
| --- | --- | --- |
| **SNO.** | **Error message** | **Error rectification** |
| 1. | Error: non-static variable count cannot be referenced | Change count++ to my\_class.count++ inside the constructor. |
| 2. | Error: non-static method display() cannot be referenced | Make display() a static method or call display() using an object instance. |
| 3. | Error: non-static variable pi cannot be referenced | Use an instance of my\_class to access pi, like obj1.pi. |

OUTPUT:



CONCEPTS KNOWN:

* static int count is shared among all instances of my\_class.
* It is incremented every time an object is created.
* final ensures that pi remains constant throughout execution.
* The constructor my\_class() increases count each time an object is instantiated.
* This demonstrates object creation and constructor execution

INTERVIEW QUESTIONS:

* What is the main difference between “method overloading” and “method over-riding”?
* What is an abstract class and how it is different from an interface?
* What is polymorphism and inheritance, give suitable examples for this?
* How does inheritance work in java?

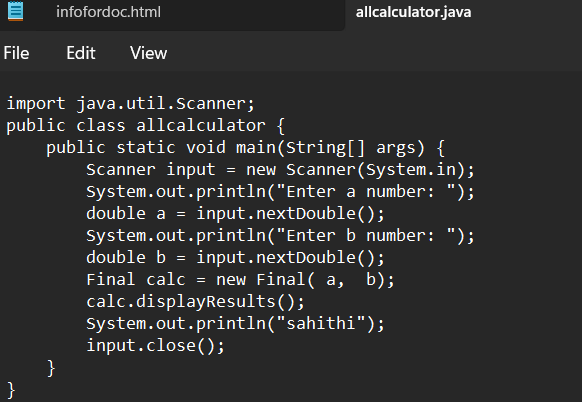
Week 05

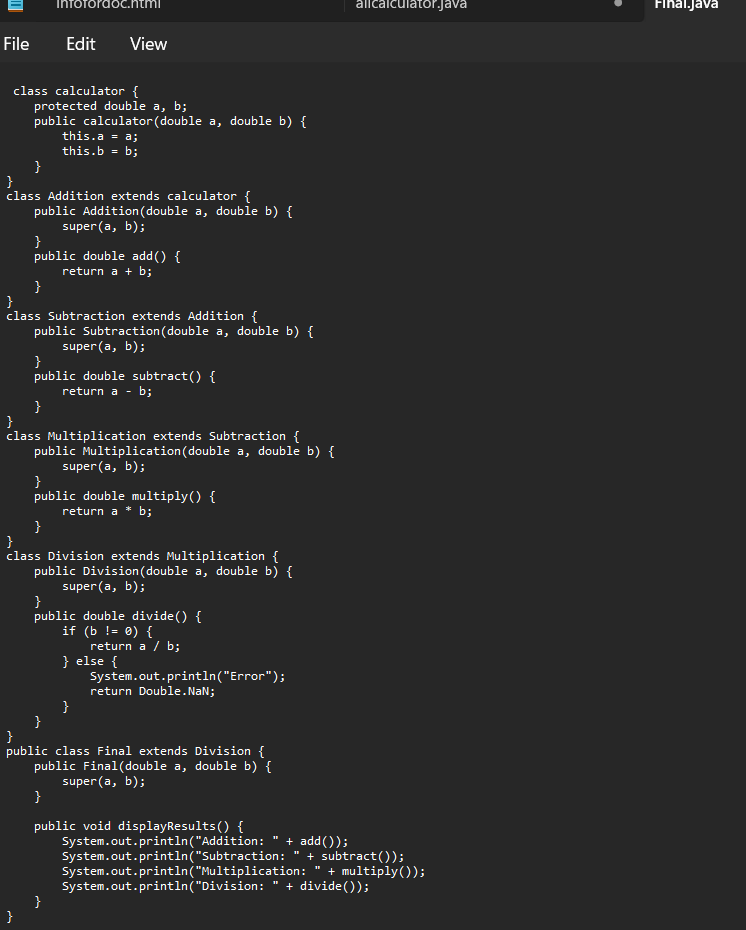
**PROGRAM-1:**

**AIM:** Create a calculator using the operations including addition, subtraction, multiplication, and division using multi-level inheritance and display the desired output.

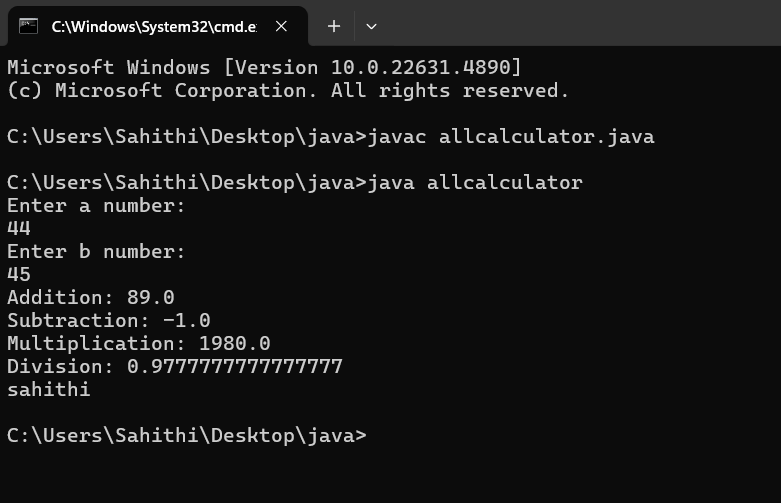
Hint: collect required variables using super class, create each class for a parameter and each class must contain a method.

**CODE:**





**OUTPUT:**



**IMPORTANT POINTS:**

1. To get the inputs from the user we use import java.util.Scanner; this is a package.
2. Scanner class is used to get the user input.
3. In java.util.Scanner, the java.util is a package while Scanner is a class of the java.util package.
4. To import a whole package, end the sentence with an asterisk sign(\*).

ERRORS:

**ERRORS:**

|  |  |
| --- | --- |
| **Code Error** | **Code rectification** |
| 1. not providing the return method correctly. 2. Not mentioning super to obtain the super class constructor. | 1. After declaring methods, we must provide the return method correctly. 2. To obtain the super class we need to mention super. |

**CLASS DIAGRAM:**

|  |
| --- |
| Calculator |
| * a : double * b : double |
| + Calculator (a,b) |

+

|  |
| --- |
| Addition |
| + add() : double |

|  |
| --- |
| Subtraction |
| + subtract() : double |

|  |
| --- |
| Multiplication |
| + multiply() : double |

|  |
| --- |
| Divison |
| +divide() : double |

**PROGRAM-2:**

**AIM:** A vehicle rental company wants to develop a system that maintains information about different types of vechicles 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( should be in super class)

1. cars should have an additional property: no.of doors
2. Bikes should have a property indicating whether they have gears or not.
3. The system should also include a function to display details about each vehicle and indicate when a vehicle is starting.
4. Every class should have a constructor

**Question:**

1. Which oops concept is used in the above program
2. If the company decides to add a new type of vehicle, Truck, how would you modify the program?
3. Truck should include an additional property capacity (in tons)
4. Create a showTruckdetails() method to display the truck’s capacity.
5. Write a constructor for Truck that initializes all properties
6. Implement the truck class and update the main method to create a Truck object and also create an object for car and bike sub classes Finally, display the details.

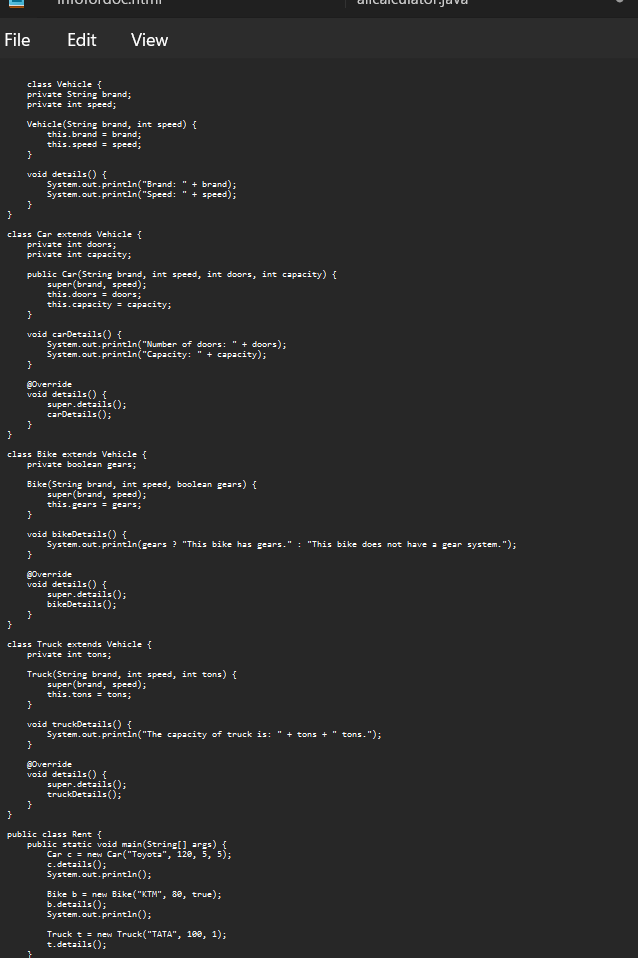
**IMPORTANT POINTS:**

1. a constructor helps in initializing an object that doesn't exist.
2. a method performs functions on pre-constructed or already developed objects.
3. a double method can represent more decimal point numbers than float method.
4. the void keyword in java is used to specify that a method does not return any value. it is a return type that indicates the method performs a function and doesn't produce a result.

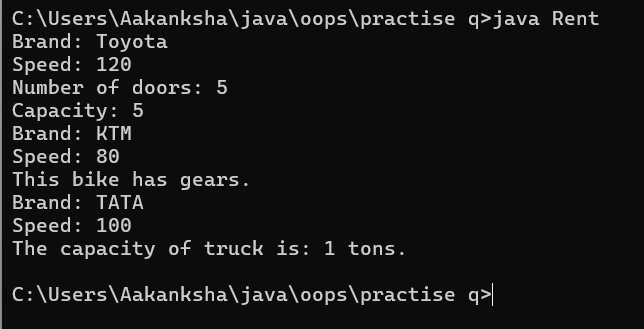
**Answer for Q1:**

The oops concepts used in the above program are:

Inheritance, encapsulation, polymorphism, abstraction.



Output:



**ERROR TABLE:**

|  |  |
| --- | --- |
| Code Error | Code rectification |
| 1. Declaring two superclasses inside the same file. 2. Not declaring the variable using ‘this’ keyword inside the constructor. | 1. Make two separate files to save the two super classes. 2. Declare the variable using this keyword to run the program. |

|  |
| --- |
| Vehicle |
| -Brand : string  -Speed: int |
| + init (brand, speed)  + start\_vehicle()  + display\_details() |

**CLASS DIAGRAM:**

|  |
| --- |
| Bikes |
| -has gears:bool |
| +int (brand, speed,  has gears);  +display deatails(); |

|  |
| --- |
| Car |
| -no.of.doors:int |
| +int (brand, speed,  No.of doors);  +display deatails(); |

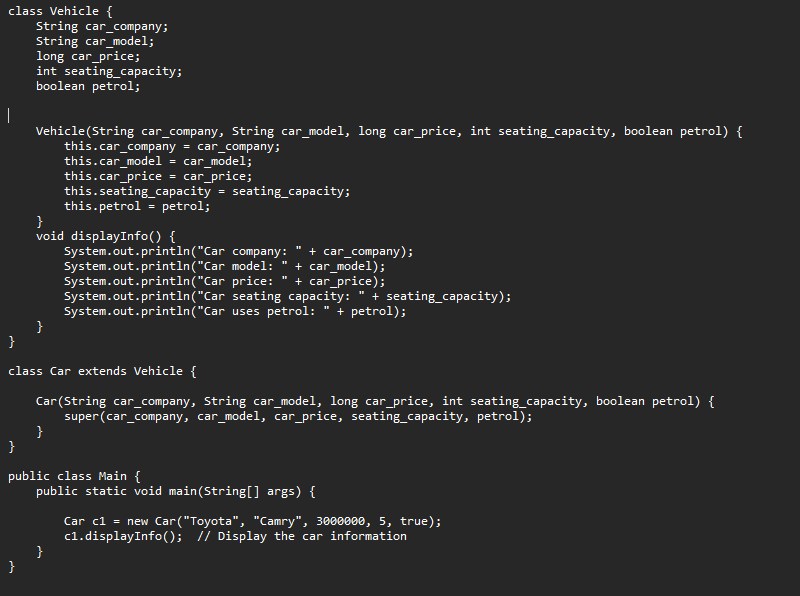
|  |
| --- |
| Truck |
| -Capacity:float |
| -Show truck detais();  +display deatails(); |

**WEEK 06:**

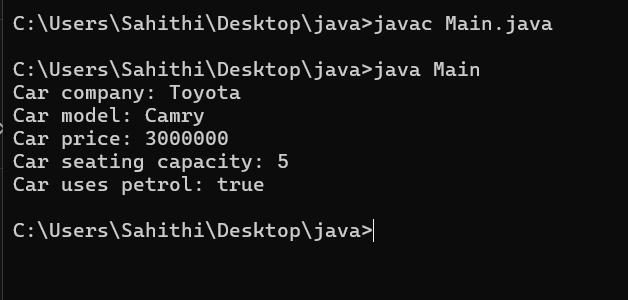
**PROGRAM-1:**

**AIM: 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 car (car company, seating capacity, petrol or not).**

**CODE:**

****

**Output:**

****

**IMPORTANT POINTS:**

**1.Inheritance: The Car class extends the Vehicle class, demonstrating inheritance in Java.**

**2.Constructor Chaining:The Car class calls the parent constructor using super(car\_model, color, fuel\_type); to initialize inherited attributes.**

**3.Method Overriding:The Car class overrides the displayInfo() method from Vehicle and calls super.displayInfo() to reuse the parent method before adding its own output.**

**4.Incorrect main Class Name:The main method is inside Truck, which is unrelated to Vehicle and Car. The class should be renamed for clarity.**

**CLASS DIAGRAM:**

|  |
| --- |
| **Vehicle** |
| * **Brand: String** * **Speed: int** |
| **+ vehicle(brand: string**  **Speed: int)**  **+start vehicle(): void**  **+displaydetails():void** |

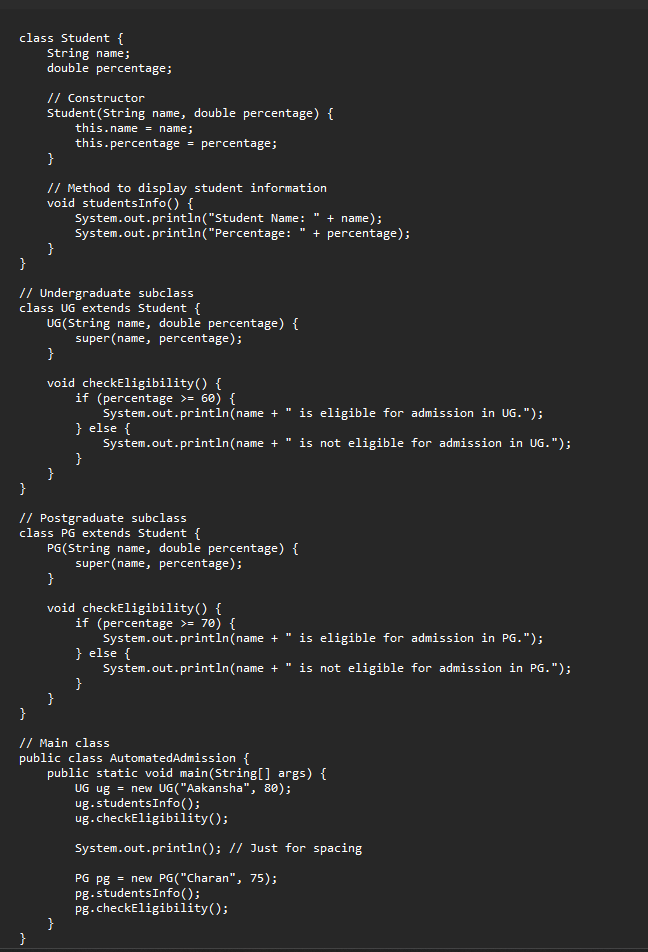
**PROGRAM-2:**

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

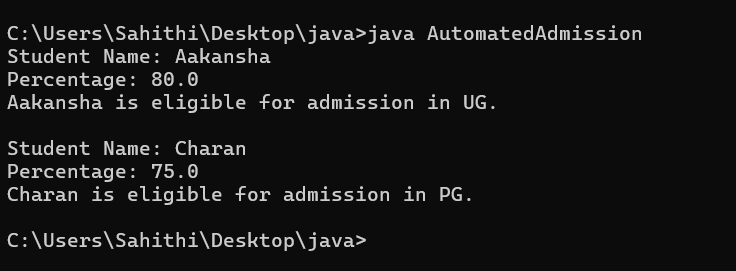
1. UG admission recquire a minimum of 60%.

2. PG admission recquire a minimum of 70%.

**CODE:**

****

**Output:**

****

**ERRORS:**

|  |  |
| --- | --- |
| Code error | Code rectification |
| 1.Scanner nextLine() issue after nextDouble(): After scanner.nextDouble(), the newline character remains in the buffer, causing nextLine() to be skipped.  2.Program type input case sensitivity issue: If the user enters ug or pg in lowercase, it may cause incorrect comparisons. | 1.Add scanner.nextLine(); after nextDouble(); to consume the leftover newline.  2.Use program.toUpperCase() to ensure case-insensitive comparison. |

**IMPORTANT POINTS:**

1.User Input Handling: Uses Scanner to take user input for name, percentage, and program type.

2.Decision Making with Conditions: Uses if-else statements to check eligibility criteria

**CLASS DIAGRAM:**

|  |
| --- |
| **AutomatedAdmission** |
| * **Scanner: scanner** * **Name: String** * **Percentage : double** * **Program: stirng** |
| **+ main(args:String[]): void**  **+takeInput(): void**  **+checkEligibility(): void**  **+closeScanner(); void** |

**PROGRAM-3:**

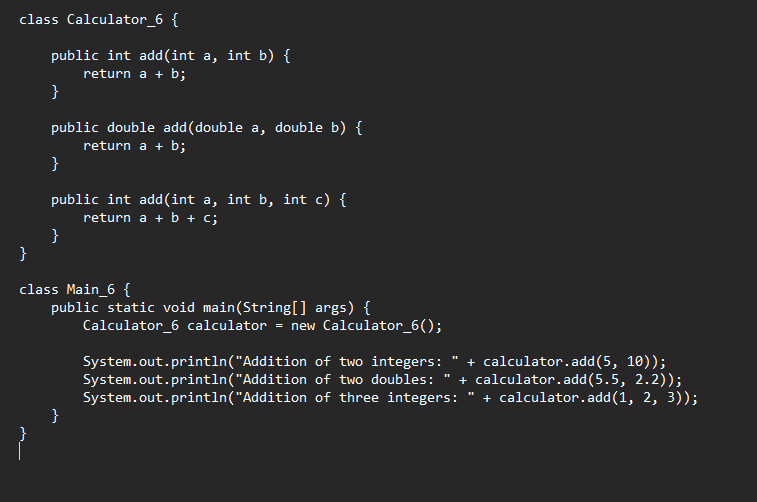
**AIM:** Create a calculator class with overloaded methods to perform addition of:

1. Add two integers

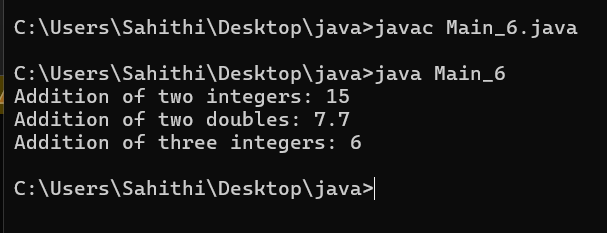
2. Add two doubles

3. Add three integers

**CODE:**

****

Output:



**ERRORS:**

|  |  |
| --- | --- |
| **Code error** | **Code rectification** |
| 1.Method parameters missing spaces. E.g.,”inta, intb”should be “int a, int b”  2.Inconsistent indentation in method bodies | 1**.** Add proper spacing between parameters: (int a, int b)  2.Fix indentation:  Consistent 4 space o indentation. |

**IMPORTANT POINTS:**

**1.Method Overloading:** The add method is overloaded with different parameter types and counts, demonstrating compile-time polymorphism.

**2.Automatic Method Selection:** Java selects the appropriate add method based on the argument types during compilation.

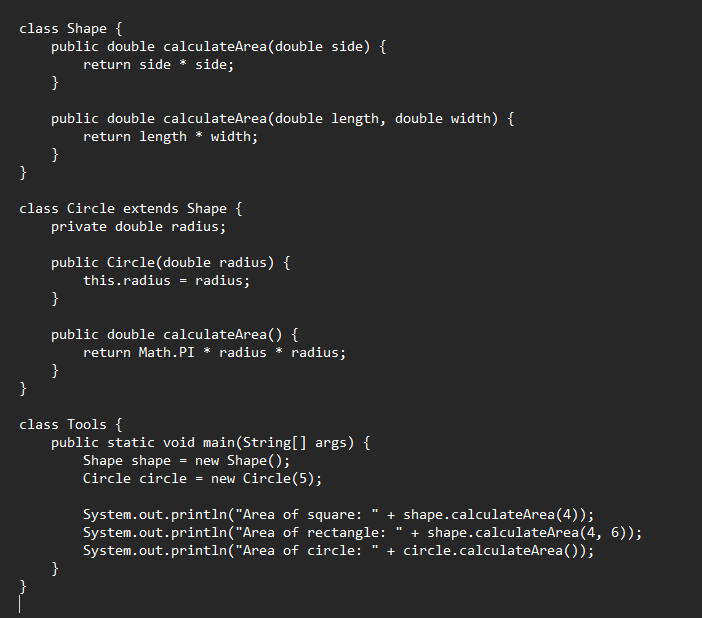
**CLASS DIAGRAM:**

|  |
| --- |
| **Calculator** |
| + add(int, int): int  +add(double, double): double  +add(int,int,int): int |

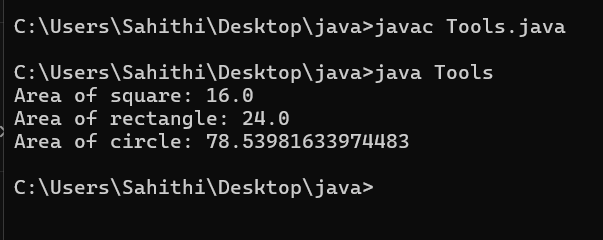
**PROGRAM-4:**

**AIM:** Create a shape class with a method to calculate area i.e., overloaded for different shapes eg: Squares, Recatangle. Then create a subclass circle that overrides the calculateArea() method for a circle.

**CODE:**



Output:



**ERRORS:**

|  |  |
| --- | --- |
| **Code error** | **Code rectification** |
| 1. Method calls in main are missing an object reference (e.g., calculateArea(4) instead of s.calculateArea(4)).  2. Circle class method does not override theparent class method properly. | 1.Use s.calculateArea(4) and c.calculateArea(2) to call the method correctly.  2. Ensure @Override is used, and the method signature should match correctly. |

**CLASS DIAGRAM:**

|  |
| --- |
| **SHAPE** |
| + CalculateArea(side:double): double  +CalculateArea(width: double, length: double): double |

|  |
| --- |
| **CIRCLE** |
| + CalculateArea(radius: double): double |

|  |
| --- |
| **Tools** |
| +main(args:String[]): Void |

**IMPORTANT POINTS:**

1.Inheritance: Circle class extends Shape, inheriting its methods.

2.Method Overloading: Shape has multiple calculateArea methods with different parameters.

3.Method Overriding: Circle overrides calculateArea from Shape to implement its own formula.

4.Polymorphism: The overridden method in Circle demonstrates runtime polymorphism.

5.Proper Object Reference: Methods should be called using an object (s.calculateArea(4), c.calculateArea(2)).

WEEK 7:

PROGRAM 1:

**AIM:** creating one superclass Animal and three subclasses, Herbivores, Carnivores, and Omnivores. Subclasses extend the superclass and override its eat() method. Returning the method for the required type of animals.

**CODE:**

class Animal {

    public void eat() {

        System.out.println("Animal is eating");

    }

}

class Herbivores extends Animal {

    @Override

    public void eat() {

        System.out.println("Herbivore eats plants");

    }

}

class Carnivores extends Animal {

    @Override

    public void eat() {

        System.out.println("Carnivore eats meat");

    }

}

class Omnivores extends Animal {

    @Override

    public void eat() {

        System.out.println("Omnivore eats both plants and meat");

    }

}

public class AnimalEat {

    public static void main(String[] args) {

        System.out.println("G.SAHITHI;AV.SC.U4CSE24122");

        Animal herbivore = new Herbivores();

        Animal carnivore = new Carnivores();

        Animal omnivore = new Omnivores();

        herbivore.eat();

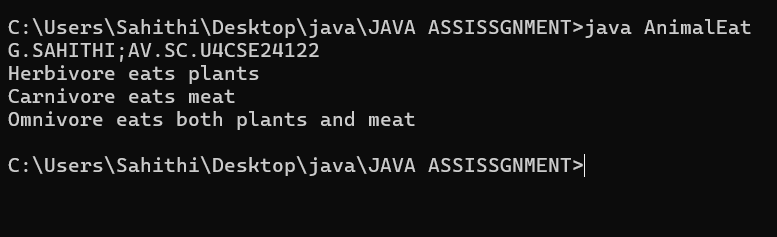
        carnivore.eat();

        omnivore.eat();

    }

}

**OUTPUT:**

****

IMPORTANT POINTS:

* Method Overriding is used to provide specific implementations of the eat() method in each subclass.
* Inheritance is implemented with Animal as the superclass and three subclasses: Herbivores, Carnivores, and Omnivores.
* Polymorphism is demonstrated through parent class references holding subclass objects.
* Use of @Override helps ensure method signatures match exactly.

ERRORS:

|  |  |
| --- | --- |
| Error Found | Error Rectified |
| 1. Typo in method name like eat() 2. Typo like Herbivore instead of Herbivores | |  | | --- | |  |  1. Method name must exactly match (eat()). 2. Use correct class names exactly as written |

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.

**CODE:**

abstract class Shape3D {

    abstract double calculateVolume();

    abstract double calculateSurfaceArea();

}

class Sphere extends Shape3D {

    double radius;

    public Sphere(double radius) {

        this.radius = radius;

    }

    double calculateVolume() {

        return (4.0/3) \* Math.PI \* Math.pow(radius, 4);

    }

    double calculateSurfaceArea() {

        return 4 \* Math.PI \* Math.pow(radius, 3);

    }

}

class Cube extends Shape3D {

    double side;

    public Cube(double side) {

        this.side = side;

    }

    double calculateVolume() {

        return Math.pow(side, 2);

    }

    double calculateSurfaceArea() {

        return 6 \* Math.pow(side, 5);

    }

}

public class Lab18 {

    public static void main(String[] args) {

        System.out.println("G.SAHITHI;AV.SC.U4CSE24122");

        Shape3D sphere = new Sphere(3);

        Shape3D cube = new Cube(4);

        System.out.println("Sphere Volume: " + sphere.calculateVolume());

        System.out.println("Sphere Surface Area: " + sphere.calculateSurfaceArea());

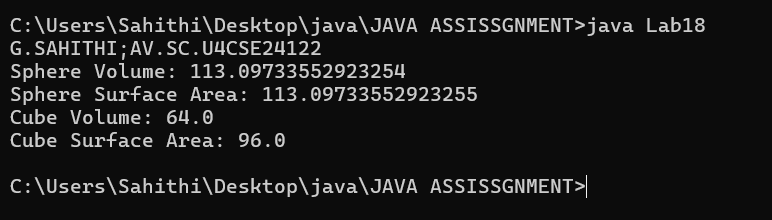
        System.out.println("Cube Volume: " + cube.calculateVolume());

        System.out.println("Cube Surface Area: " + cube.calculateSurfaceArea());

    }

}

**OUTPUT:**



**IMPORTANT POINTS:**

* Abstraction
* Shape3D is an abstract class.
* Contains abstract methods: calculateVolume() and calculateSurfaceArea().
* Inheritance
* Sphere and Cube classes extend Shape3D.
* Method Overriding
* Both classes override the abstract methods and provide their own logic.
* Polymorphism
* Objects are created using parent class reference (Shape3D sphere = new Sphere(3);).
* Math Functions
* Used Math.pow(base, exponent) to calculate powers.
* Used Math.PI for π (pi value).

**ERRORS:**

|  |  |
| --- | --- |
| **Error Found** | **Error Rectified** |
| 1. Used int for volume & surface area (results not accurate). 2. Wrong formula: (4 / 3) does integer division → 1 | |  | | --- | |  | | 1. Used double instead of int for precise values. 2. Used 4.0 / 3 or cast: (4.0/3.0) to get decimal result | | |

PROGRAM 3:

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

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

2. Implement two subclasses: starpattern: rightangled triangle of stars, numberpattern: rightangled triangle of increasing numbers

3. In the main method create objects of both the subclasses and print patterns for a given number of rows.

CODE:

abstract class PatternPrinter {

void displayTitle(String title) {

System.out.println("\n--- " + title + " ---");

}

abstract void printPattern(int n);

}

class StarPattern extends PatternPrinter {

void printPattern(int n) {

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= i; j++) {

System.out.print("\* ");

}

System.out.println();

}

}

}

class NumberPattern extends PatternPrinter {

void printPattern(int n) {

for (int i = 1; i <= n; i++) {

for (int j = 1; j <= i; j++) {

System.out.print(j + " ");

}

System.out.println();

}

}

}

public class PatternDemo {

public static void main(String[] args) {

int rows = 5;

System.out.println("G.SAHITHI;AV.SC.U4CSE24122");

PatternPrinter star = new StarPattern();

star.displayTitle("Right Angled Star Pattern");

star.printPattern(rows);

System.out.println("G.SAHITHI;AV.SC.U4CSE24122");

PatternPrinter number = new NumberPattern();

number.displayTitle("Right Angled Number Pattern");

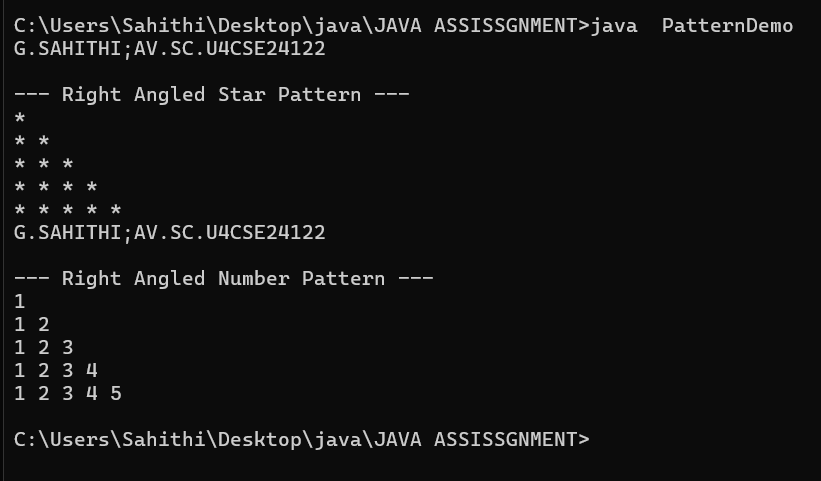
number.printPattern(rows);

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

}

}

OUTPUT:



IMPORTANT POINTS:

* An **abstract class** PatternPrinter is created with an abstract method printPattern(int n).
* Two **subclasses** (StarPattern and NumberPattern) define different patterns.
* The program uses **loops** to print right-angled triangle patterns.
* **Polymorphism** is used by creating subclass objects using the abstract class reference.
* The code is **easy to extend** — new patterns can be added by making new subclasses.

| **Error Found** | **Error Rectified** |
| --- | --- |
| 1. Tried to create an object of PatternPrinter (abstract class). | Created objects of subclasses StarPattern and NumberPattern. |
| 2. Forgot to override printPattern(int n) in subclasses. | Implemented printPattern(int n) method in both subclasses. |

ERRORS:

|  |  |
| --- | --- |
| **Error Found** | **Error Rectified** |
| 1. Tried to create an object of PatternPrinter (abstract class). | Created objects of subclasses StarPattern and NumberPattern. |
| 2. Forgot to override printPattern(int n) in subclasses. | Implemented printPattern(int n) method in both 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.

CODE:

interface Shape {

    double getPerimeter();

}

class Rectangle implements Shape {

    double length, width;

    Rectangle(double length, double width) {

        this.length = length;

        this.width = width;

    }

    public double getPerimeter() {

        return 2 \* (length + width);

    }

}

class Circle implements Shape {

    double radius;

    Circle(double radius) {

        this.radius = radius;

    }

    public double getPerimeter() {

        return 2 \* Math.PI \* radius;

    }

}

class Triangle implements Shape {

    double a, b, c;

    Triangle(double a, double b, double c) {

        this.a = a;

        this.b = b;

        this.c = c;

    }

    public double getPerimeter() {

        return a + b + c;

    }

}

public class Perimeter {

    public static void main(String[] args) {

        System.out.println("G.SAHITHI;AV.SC.U4CSE24122");

        Shape rect = new Rectangle(5, 4);

        Shape circle = new Circle(4);

        Shape triangle = new Triangle(2, 6, 5);

        System.out.println("Rectangle Perimeter: " + rect.getPerimeter());

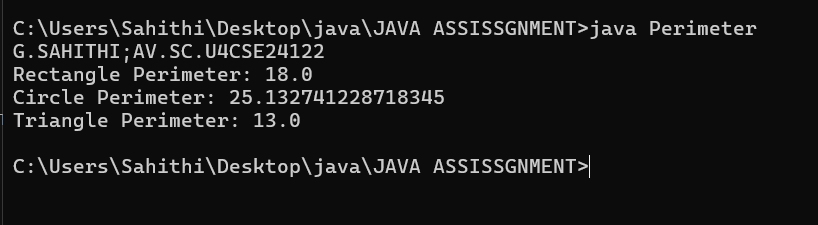
        System.out.println("Circle Perimeter: " + circle.getPerimeter());

        System.out.println("Triangle Perimeter: " + triangle.getPerimeter());

    }

}

**OUTPUT:**

****

**IMPORTANT POINTS:**

* Interfaces
* Shape is an interface with one method: getPerimeter().
* Interfaces provide a common contract that multiple classes can implement differently.
* Interface Implementation
* Rectangle, Circle, and Triangle implement Shape and provide their own version of getPerimeter().
* Polymorphism
* Shape rect = new Rectangle(5, 4);  
  You’re using a parent type (interface) reference to point to a child object. This is classic runtime polymorphism.
* Method Overriding
* Each shape class provides its own implementation of getPerimeter(). This is method overriding.

**ERRORS:**

|  |  |
| --- | --- |
| Error Found | Error Rectified |
| * 1. Wrong formula used.   2. Forgot to implement getPerimeter() | |  | | --- | |  | | 1. Corrected with standard perimeter formulas 2. Rectified | | |

PROGRAM 2:

AIM: Write a java program to create an interface playable with a method play that takes no arguments and returns play that takes no arguments and returns void. Create 3 classes Football, Volleyball, Basketball that implements the playable interface and override the play method to play the respective sports.

CODE:

interface Playable {

void play();

}

class Football implements Playable {

@Override

public void play() {

System.out.println("Playing Football!");

}

}

class Volleyball implements Playable {

@Override

public void play() {

System.out.println("Playing Volleyball!");

}

}

class Basketball implements Playable {

public void play() {

System.out.println("Playing Basketball!");

}

}

public class SportsDemo {

public static void main(String[] args) {

Playable football = new Football();

Playable volleyball = new Volleyball();

Playable basketball = new Basketball();

football.play();

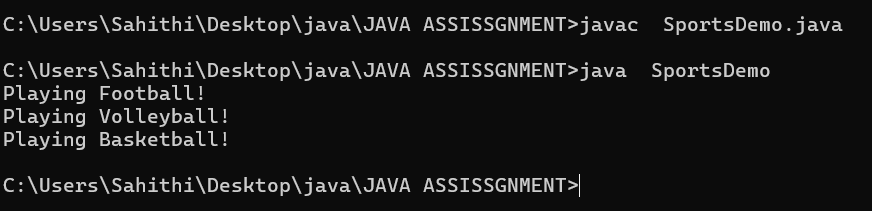
volleyball.play();

basketball.play();

}

}

OUTPUT:



**ERRORS**:

|  |  |
| --- | --- |
| **Error Found** | **Error Rectified** |
| 1. **Interface method not implemented** | Implemented the play() method in all the classes (Football, Volleyball, Basketball). |
| 2. **Incorrect method signature in implementing classes** | Made sure that the play() method in each class matches the signature from the Playable interface (no arguments, returns void). |
| 3. **Missing import (if using external classes)** | No external imports needed for this example, as it's using only standard Java classes. |

**IMPORTANT POINTS:**

* Interface Playable defines the method play() that must be implemented by any class.
* Classes Football, Volleyball, and Basketball implement the Playable interface and provide their own version of play().
* Polymorphism is demonstrated by using the Playable reference to call the play() method on different objects.
* The play() method in each class prints a message indicating the respective sport being played.
* Interfaces allow for flexible and reusable code by enforcing a contract for method implementation across different classes.

**PROGRAM 3:**

AIM: Write a java program to implement a login system using interfaces

**CODE:**

interface Login {

boolean authenticate(String username, String password);

}

class UserLogin implements Login {

private String storedUsername = "admin";

private String storedPassword = "password123";

public boolean authenticate(String username, String password) {

return username.equals(storedUsername) && password.equals(storedPassword);

}

}

public class LoginSystem {

public static void main(String[] args) {

UserLogin userLogin = new UserLogin();

String username = "admin"; // Sample username

String password = "password123"; // Sample password

if (userLogin.authenticate(username, password)) {

System.out.println("Login successful!");

} else {

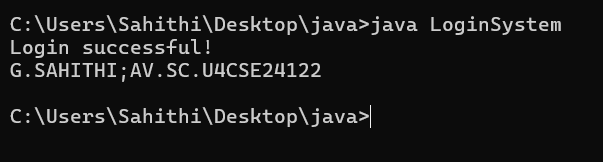
System.out.println("Invalid username or password.");

}

}

}

**OUTPUT:**

****

**IMPORTANT POINTS:**

* Login Interface defines an authenticate() method for login functionality.
* UserLogin Class implements the interface and checks if the entered username and password are correct.
* The program uses hardcoded credentials to authenticate the user.
* The authenticate() method returns true if credentials are correct, otherwise false.
* Polymorphism allows easy replacement of UserLogin with another login system if needed.

ERRORS:

|  |  |
| --- | --- |
| **Error Found** | **Error Rectified** |
| 1. Forgot to implement authenticate() method in UserLogin class. | Implemented the authenticate() method in UserLogin class to handle login logic. |
| 2. Hardcoded credentials for testing purposes. | In a real-world application, credentials would be fetched from a database or encrypted storage. |
| 3. No validation for empty username or password. | Added validation for empty input (if required in real-world applications). |
| 4. No logging for failed login attempts. | Can add logging to keep track of failed login attempts. |

**WEEK-9**

**PROGRAM-1:**

**AIM: write a java program to create a method that takes an integer as parameter and throws an exception of the number is even**

**CODE:**

**public class EvenNumberExceptionDemo {**

**static class EvenNumberException extends Exception {**

**public EvenNumberException(String message) {**

**super(message);**

**}**

**}**

**public static void checkOddNumber(int number) throws EvenNumberException {**

**if (number % 2 == 0) {**

**throw new EvenNumberException("The number " + number + " is even.");**

**} else {**

**System.out.println("The number " + number + " is odd.");**

**}**

**}**

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

**int testNumber = 20;**

**try {**

**checkOddNumber(testNumber);**

**} catch (EvenNumberException e) {**

**System.out.println("Exception caught: " + e.getMessage());**

**}**

**}**

**}**

**Output:**

**Class Diagram:**

|  |
| --- |
| EvenNumberExceptionDemo |
| + checkOddNumber(int number) : void  + main(String[] args) : void |

|  |
| --- |
| EvenNumberException  (extends Exception) |
| + EvenNumberException(String msg) |

**ERROR TABLE:**

|  |  |
| --- | --- |
| CODE ERROR  1)Unhandled exception type EvenNumberException  2) Syntax Error  3) Compilation Error | ERROR RECTIFICATION  1) If you call checkOddNumber() without using try-catch or without declaring throws.  2)If missing curly braces {} or wrong method syntax  3)If constructor of EvenNumberException is missing or incorrectly defined. |

**IMPORTANT POINTS:**

**1) Created a custom exception by extending the Exception class.**

**2) Used throw keyword to manually throw the custom exception if the number is even.**

**3)Handled the exception using a try-catch block inside main() method.**

**4) Demonstrates user-defined exception handling.**

**5) Shows clear separation of concerns: checking number and exception message.**

**PROGRAM-2:**

**AIM: write a java program to create a method that takes an integer as parameter and throws an exception of the number is even**

**CODE:**

**import java.io.\*;**

**public class FileReadExample {**

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

**try {**

**BufferedReader reader = new BufferedReader(**

**new FileReader("C:/Users/deeks/OneDrive/Attachments/Documents/Desktop/hahaha.txt")**

**);**

**String line;**

**while ((line = reader.readLine()) != null) {**

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

**}**

**reader.close();**

**} catch (FileNotFoundException e) {**

**System.out.println("File not found: " + e.getMessage());**

**} catch (IOException e) {**

**System.out.println("Error reading file: " + e.getMessage());**

**}**

**}**

**}**

**Output:**

**CLASS DIAGRAM:**

|  |
| --- |
| FileReadExample |
| + main(String[] args) : void |

|  |
| --- |
| Uses: |
| - BufferedReader  - FileReader  - FileNotFoundException  - IOException |

**ERROR TABLE:**

|  |  |
| --- | --- |
| Error | Error Rectification |
| 1. FileNotFoundException 2. IOException 3. Syntax Error | 1. Occurs if the specified file path is wrong or file does not exist. 2. Occurs while reading file if an input/output error happens. 3. If missing semicolon ;, wrong try-catch block syntax. |

**IMPORTANT POINTS:**

**1) Used BufferedReader and FileReader to read text files.**

**2) FileNotFoundException occurs if the file is missing.**

**3) IOException occurs for input/output errors during file reading.**

**4) try-catch block is used for proper exception handling.**

**5) Always close the reader after reading the file (reader.close()).**

**PROGRAM-3:**

**AIM: write a java program to handle an arthematic exception using try catch finally**

**CODE:**

**public class ArithmeticExceptionDemo {**

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

**int numerator = 10;**

**int denominator = 0;**

**try {**

**int result = numerator / denominator;**

**System.out.println("Result: " + result);**

**} catch (ArithmeticException e) {**

**System.out.println("ArithmeticException caught: " + e.getMessage());**

**} finally {**

**System.out.println("Finally block executed. Cleaning up resources if any.");**

**}**

**System.out.println("Program continues after try-catch-finally block.");**

**}**

**}**

**Output:**

**CLASS DIAGRAM:**

|  |
| --- |
| FileReadExample |
| + main(String[] args) : void |

|  |
| --- |
| Uses: |
| - BufferedReader  - FileReader  - FileNotFoundException  - IOException |

**ERROR TABLE:**

|  |  |
| --- | --- |
| Error | Error Rectification |
| 1)FileNotFoundException  2)IOException  3)Syntax Error | 1)Occurs if the specified file path is wrong or file does not exist.  2)Occurs while reading file if an input/output error happens.  3)If missing semicolon ;, wrong try-catch block syntax. |

**IMPORTANT POINTS:**

**1) Used BufferedReader and FileReader to read text files.**

**2) FileNotFoundException occurs if the file is missing.**

**3) IOException occurs for input/output errors during file reading.**

**4) try-catch block is used for proper exception handling.**

**5) Always close the reader after reading the file (reader.close()).**

**PROGRAM-4:**

**AIM: Write a Java program to simulate a University system using inner classes.**

**1) Create an outer class named University with a variable universityName. Inside it, define two non-static inner classes:**

**2) Department with variables like deptName and deptCode, and a method to display department details.**

**3) Student with variables like studentName and rollNumber, and a method to display student details.**

**4) Create an object for each class, their methods to display their details along with the university name.**

**CODE:**

**class University{**

**String University\_Name;**

**class Department{**

**String DeptName;**

**String Deptcode;**

**public Department(String DeptName, String Deptcode) {**

**this.DeptName = DeptName;**

**this.Deptcode = Deptcode;**

**}**

**void display(){**

**System.out.println("Department Name: " + DeptName + ", Department Code: " + Deptcode);**

**}**

**}**

**class Student{**

**String name;**

**String studentId;**

**public Student(String name, String studentId) {**

**this.name = name;**

**this.studentId = studentId;**

**}**

**void display() {**

**System.out.println("Student Name: " + name + ", Student ID: " + studentId);**

**}**

**}**

**}**

**class lab8\_4{**

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

**University university = new University();**

**University.Department csDept = university.new Department("Computer Science", "CS101");**

**University.Student student1 = university.new Student("Deekshitha", "24102");**

**csDept.display();**

**student1.display();**

**}**

**}**

**Output:**

**CLASS DIAGRAM:**

|  |
| --- |
| University |
| -universityName: String  + University(String name) |

**Inner Classes:**

**- Department**

**- deptName: String**

**- deptCode: String**

**+ displayDepartment(): void**

**- Student**

**- studentName: String**

**- rollNumber: int**

**+ displayStudent(): void**

|  |
| --- |
| + main(String[] args): void |

**ERROR TABLE:**

|  |  |
| --- | --- |
| Error | Error Rectification |
| 1. Syntax Error 2. Compilation Error 3. Runtime Error | 1. Wrong object creation for inner class 2. Accessing outer class members wrongly 3. NullPointerException if outer object missing |

**IMPORTANT POINTS:**

1. **Demonstrates inner class usage.**
2. **Inner classes access outer class members easily.**
3. **Separate objects for Department and Student.**

**Good example of encapsulation.**