**23CSE111**

**OBJECT ORIENTED PROGRAMMING**



**Department of computer and communication Engineering**

**Amrita School of Engineering**

**Amrita Vishwa Vidyapeetham, Amaravati Campus**

**NAME: SHAIK FEROZI BEGAM**

**ROLL NUMBER:AV.SC.U4CSE24251**

**VERFIED BY:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| WEEK | SNO. | TITLE | DATE | PAGE | SIGNATURE |
| WEEK1 |  |  |  |  |  |
|  | 1 | How to install jdk (java development kit) from Oracle. |  |  |  |
|  | 2 | Write a java program of student details. |  |  |  |
|  |  |  |  |  |  |
| WEEK2 | 1 | Write a java program to calculate the area of a rectangle. |  |  |  |
|  | 2 | Write a java program to temperature from Celsius to Fahrenheit and vica-versa. |  |  |  |
|  | 3 | Write a java program to calculate the simple interest. |  |  |  |
|  | 4 | Write a java program to find the factorial of a number. |  |  |  |
|  | 5 | Write a java program to find the fibonacci sequence of a number. |  |  |  |
|  |  |  |  |  |  |
| WEEK-3 | 1 | 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” . |  |  |  |
|  | 2 | 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 update 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 withdraw(): Whenever an amount is withdrawn, it has to be less than the current amount , else print (“Insufficient funds”) .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, 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 and display the details of two books. |  |  |  |
|  | 2 | Create a java Program with class named myclass with static variable count of int type, initialized to zero and a constant variable “pi” of type double initialized to 3.14 as attributes of the class, ow define a constructor for “myclass” that increments the count variable each time an object of my class is created (count++), finally print the final values of count and pi variables create three objects. |  |  |  |
|  |  |  |  |  |  |
| WEEK-5 | 1 | Create a calculator using the operations including addition, subtraction  Multiplication and division using multilevel inheritance and display the desired output. |  |  |  |
|  | 2 | 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.   1. Cars should have an additional property: number of doors, seating capacity. 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.   **Questions**   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 and also create an object for car and bike subclasses. Properly display its details. |  |  |  |
|  |  |  |  |  |  |
| WEEK-6 | 1 | 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. |  |  |  |
|  | 2 | 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.  (I)UG Admissions require a minimum of 60%.  (II)PG Admissions require a minimum of 70%. |  |  |  |
|  | 3 | Create a Calculator class with overloaded methos to perform addition:  i) Add two integers  ii) Add two doubles  iii) Add three integers |  |  |  |
|  | 4 | 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(). |  |  |  |
|  |  |  |  |  |  |
| Week 7 | 1 | Write a java program to create an abstract class Animal with an abstract method called sound(). Create subclasses lion and tiger that extent the Animal class and implement the sound() method to make a specific for each animal. |  |  |  |
|  | 2 | Write a java program to create an abstract class shape 3D with abstract method called sound().Create subclass sphere and cude that extend the shape 3D class and implement. |  |  |  |
|  | 3 | 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 pattern 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 patterns for a given number of rows.  Example Output for n = 5:  Star Pattern  \*  \* \*  \* \* \*  \* \* \* \*  \* \* \* \* \*  Number Pattern  1  1 2  1 2 3  1 2 3 4  1 2 3 4 5 |  |  |  |
|  |  |  |  |  |  |
| Week 8 | 1 | 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. |  |  |  |
|  | 2 | Write a Java program to create an interface Playable with a method play() that takes no arguments 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. |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**WEEK-1**

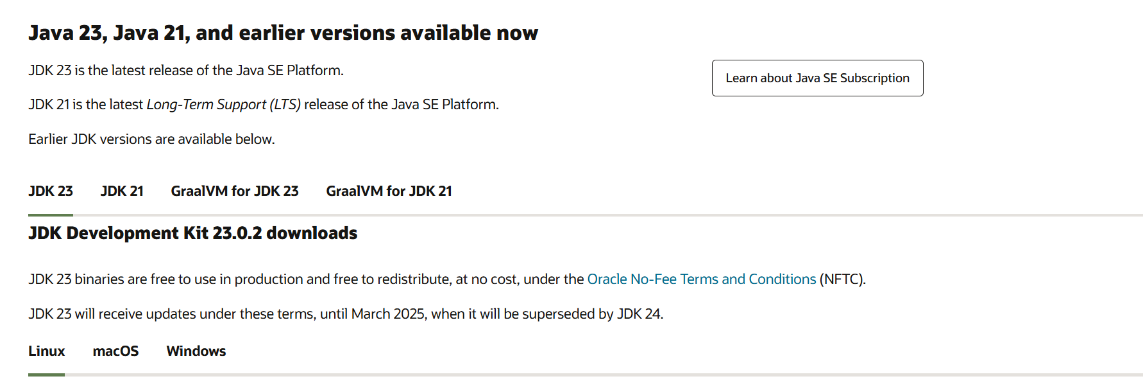
**AIM- How to install jdk(java development kit) from Oracle**.

**PROCEADURE-**

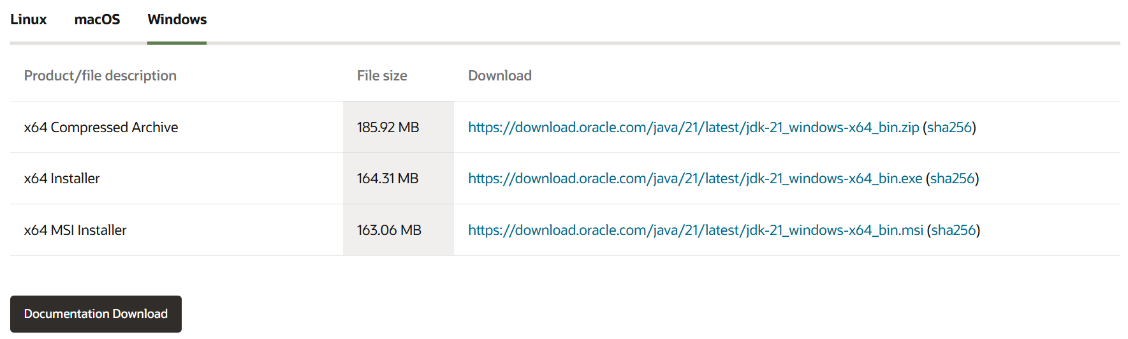
**STEP-1** To download the **Oracle Java Development Kit (JDK)** from the Official Oracle Website search ‘oracle java download’.

**STEP-2** Click on the link https://www.oracle.com>java>techonologies>downloads.

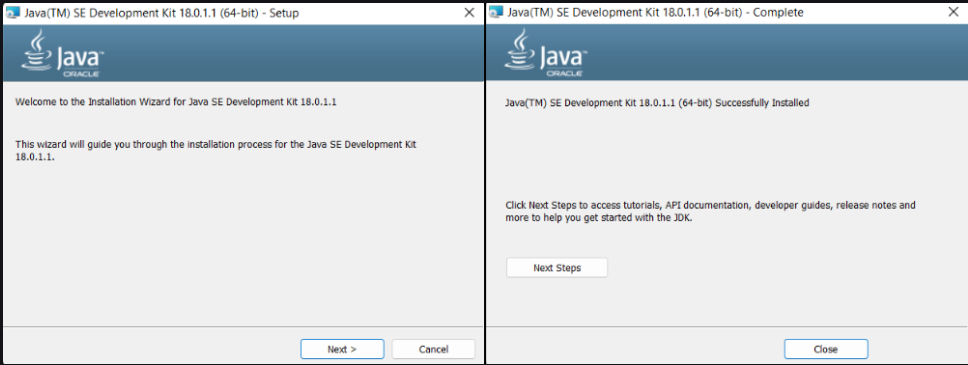
**STEP-3** Now select JDK 21(as it is stable).

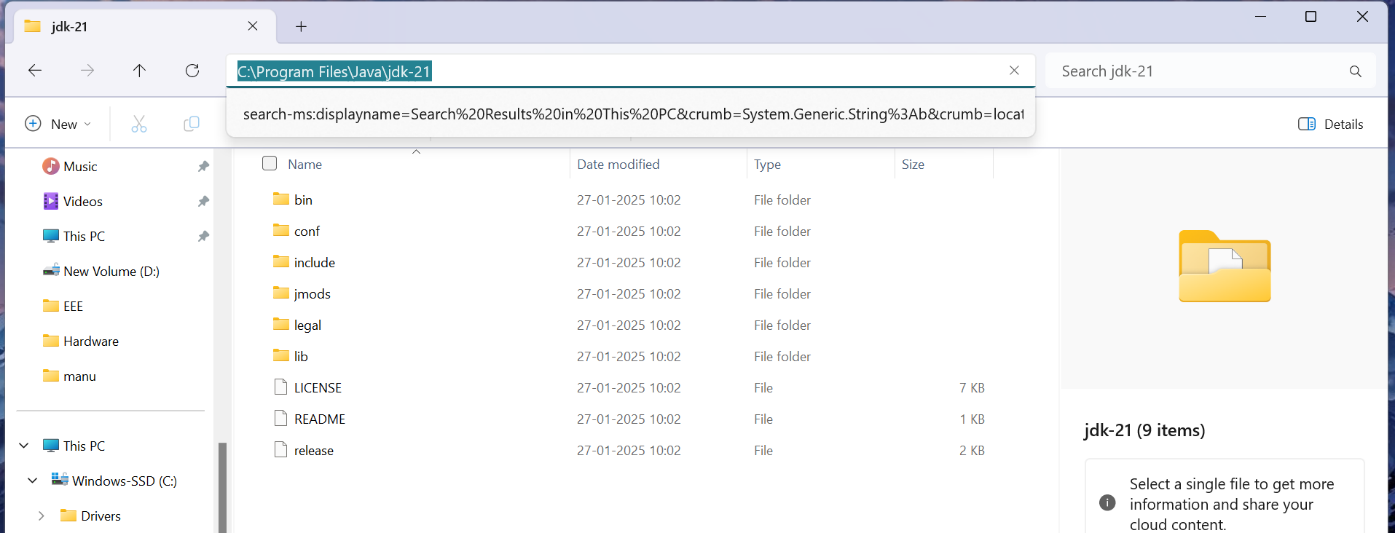
****

**STEP-4** To install JDK in windows – we’ll be downloading the latest **x64 Installer**

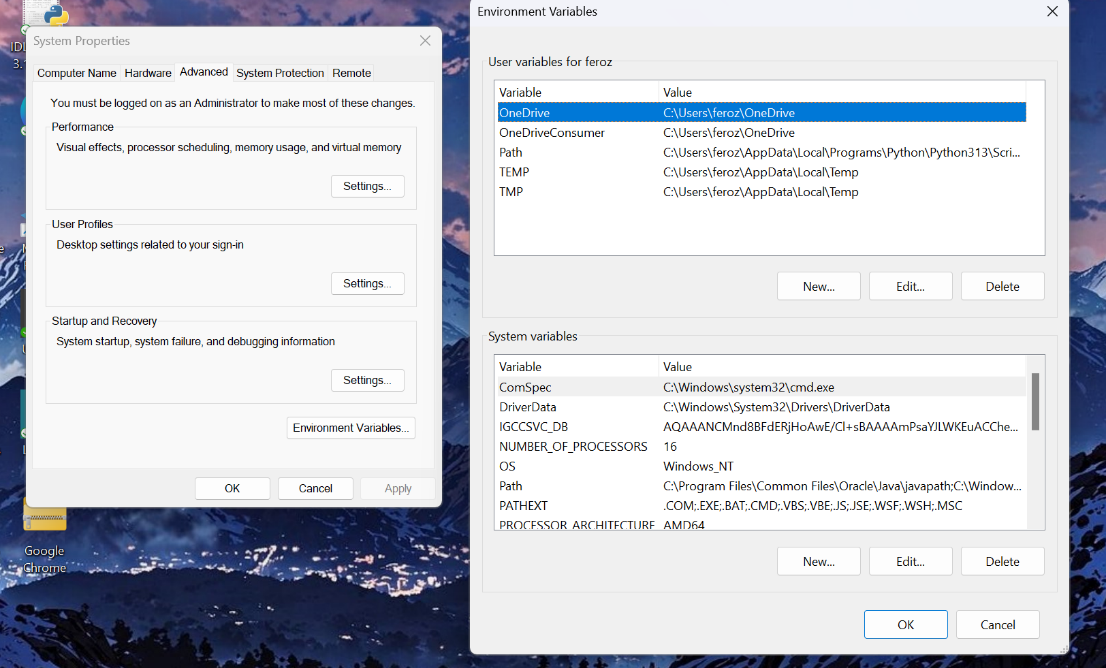


**STEP-5** After the download is complete, proceed to install the JDK by following the bootstrapped steps.

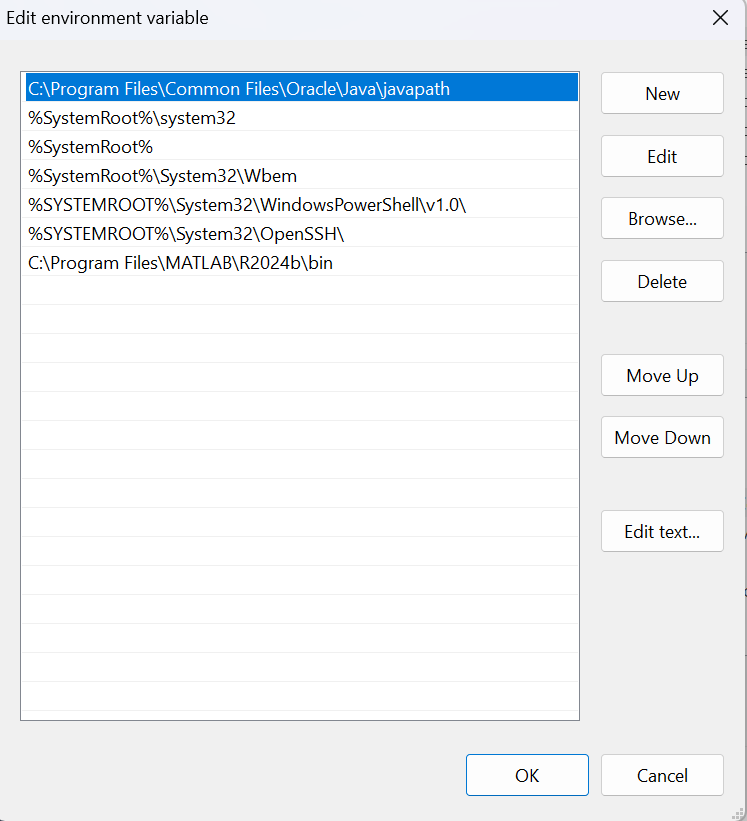


**STEP-6** We have to configure environment variables to notify the system about the directory in which JDK files are located. GO to the jdk21 location i.e C:\Program Files\Java\jdk-21.

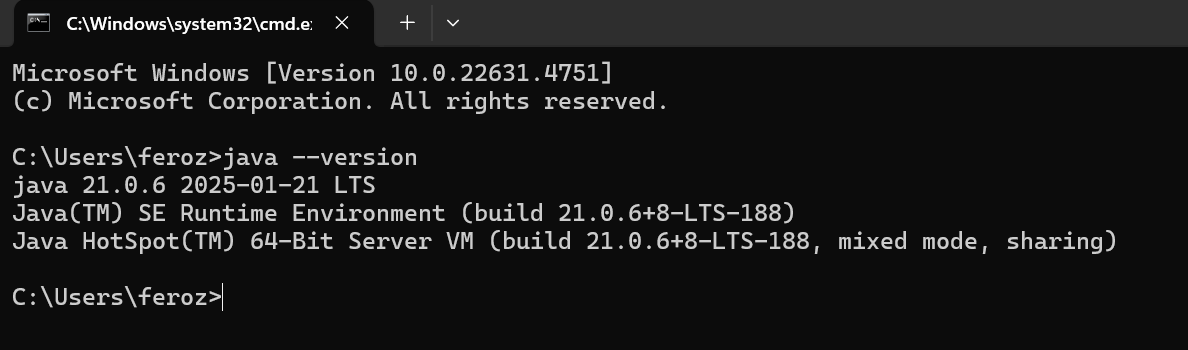
**STEP-7** Search Environment Variables in the Task Bar and click on **“Edit the system environment variables”.** Click on **“Environment Variables”**. Double click on **“Path”** in System variable.



**STEP-8** Click on **“New”** then paste the Path Address i.e. C:\Program Files\Java\jdk-21Click on **“OK”** and click on **“OK”** again and the path will be set.



**STEP-9** Now to verify if it got set or not we Open **Command Prompt**and enter the following commands **“java --version”.**



Now java is successfully installed and a path is successfully set.

**Program-1**

**AIM- Write a java program of student details.**

class studentdetail

{

public static void main(String[] args)

{

System.out.println("Name-SHAIK FEROZI BEGAM");

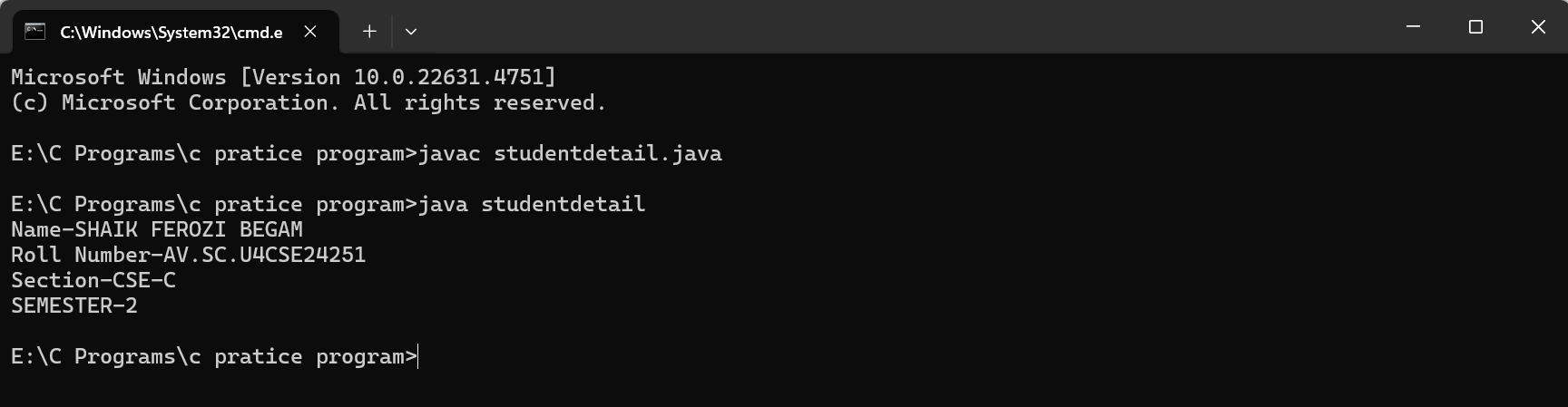
System.out.println("Roll Number-AV.SC.U4CSE24251");

System.out.println("Section-CSE-C");

System.out.println("SEMESTER-2");

}

}



NO ERROR FOUND

**WEEK-2**

**PROGRAM-1**

**AIM-** **Write a java program to calculate the area of a rectangle**.

import java.util.Scanner;

public class Area {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in)

// Calculate area of triangle

        System.out.println("Enter base of the triangle:");

        double baseTriangle = scanner.nextDouble();

        System.out.println("Enter height of the triangle:");

        double heightTriangle = scanner.nextDouble();

double areaTriangle = TriangleArea(baseTriangle, heightTriangle);

        System.out.println("Area of the triangle: " + areaTriangle);

        // Calculate area of rectangle

        System.out.println("Enter length of the rectangle:");

        double lengthRectangle = scanner.nextDouble();

        System.out.println("Enter width of the rectangle:");

        double widthRectangle = scanner.nextDouble();

        double areaRectangle = RectangleArea(lengthRectangle, widthRectangle);

        System.out.println("Area of the rectangle: " + areaRectangle);

scanner.close();

    }

public static double TriangleArea(double base, double height) {

    return 0.5 \* base \* height;

    }

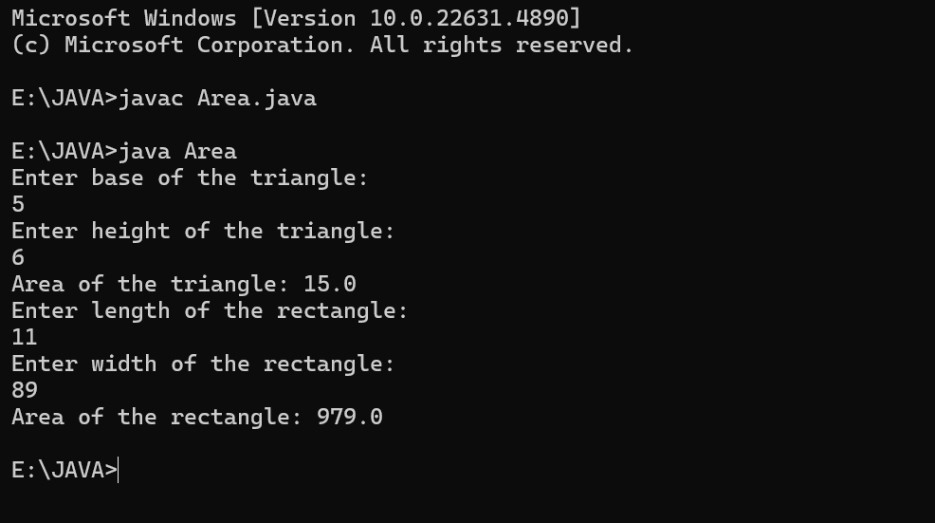
    public static double RectangleArea(double length, double width) {

        return length \* width;

    }

}

**OUTPUT-**



**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| **S.No:** | **Errors** | **Error rectification** |
| **1)** | cannot find symbol: class Scanner | Import java.util.Scanner at the beginning of the code: import java.util.Scanner; |
| **2)** | |  | | --- | | scannernextDouble(); |  |  | | --- | |  | | Scanner.nextDouble();  Should keep a dot. |

**IMPORTANT POINTS –**

1. Call nextDouble(): Once the Scanner object is created, you can call nextDouble() to read the next token as a double. This method will block until it receives a valid input.
2. getDoubleInput() -to handle invalid inputs. This method ensures that the user is prompted again if they enter something that is not a valid number.

**PROGRAM-2**

**AIM-** **Write a java program to temperature from Celsius to Fahrenheit and vica-versa.**

class Temperature{

public static double CelsiustoFahrenheit(double celsius){

  return(celsius\*1.8)+32;

}

public static double FahrenheittoCelsius(double fahrenheit) {

 return (fahrenheit-32)\*0.5;

}

public static void main(String[] args){

double celsius=100;

double fahrenheit=100;

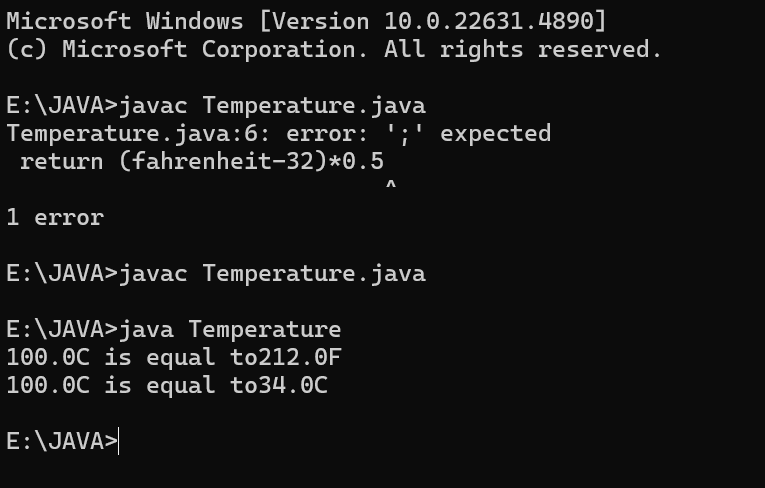
System.out.println(celsius+"C is equal to"+CelsiustoFahrenheit(celsius)+"F");

System.out.println(fahrenheit+"C is equal to"+FahrenheittoCelsius(fahrenheit)+"C");

}

}

**OUTPUT-**

****

**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| **S.No:** | **Errors** | **Error rectification** |
| **1)** | Return (Fahrenheit-32)\*0.5 | After a return statement we should keep “;”. |
| **2)** | Class name-Temperature  File name-temperature   |  | | --- | |  | | Class name and file name should be same.  Otherwise it wont be complied. |

**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 .

**PROGRAM-3**

**AIM- Write a java program to calculate the simple interest.**

import java.util.Scanner;

public class SI {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the principal amount: ");

        double principal = scanner.nextDouble();

        System.out.print("Enter the rate of interest (in %): ");

        double rate = scanner.nextDouble();

        System.out.print("Enter the time period (in years): ");

        double time = scanner.nextDouble();

        double simpleInterest = calculateSimpleInterest(principal, rate, time);

        System.out.printf("Simple Interest: %.2f%n", simpleInterest);

        System.out.printf("Total Amount: %.2f%n", principal + simpleInterest);

        scanner.close();

    }

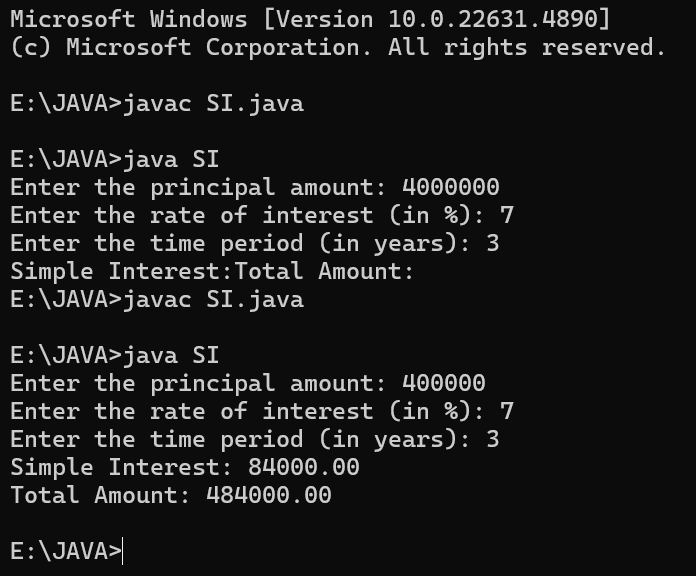
    public static double calculateSimpleInterest(double principal, double rate, double time) {

        return (principal \* rate \* time) / 100;

    }

}

**OUTPUT-**



**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| **S.No:** | **Errors** | **Error rectification** |
| **1)** | error: ';' expected  return (principal \* rate \* time) / 100 | Add a semicolon at the end of the statement  return (principal \* rate \* time) / 100; |

**IMPORTANT POINTS –**

%.2f:

%: Indicates the start of a format specifier.

.2: Specifies that the number should be rounded to two decimal places.

f: Indicates that the number is a floating-point number.

%n:

%n: Inserts a platform-independent line separator.

This means it will use the correct newline character(s) for the operating system it's running on .

**PROGRAM-4**

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

public class Factorial {

    public static void main(String[] args) {

        int num = 5;

        long factorial = 1;

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

            factorial \*= i;

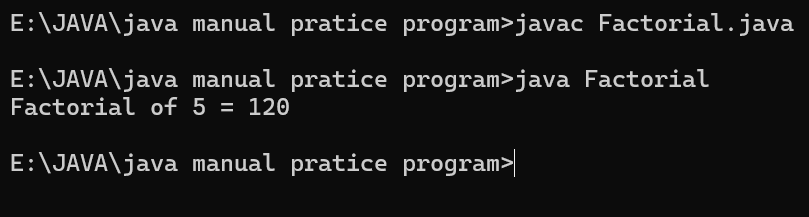
        }

System.out.println("Factorial of " + num + " = " + factorial);

    }

}

**OUTPUT-**



**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| **S.No:** | **Errors** | **Error rectification** |
| **1)** | Error- incantation needed .  for (int i = 1; i <= num ) | for (int i = 1; i <= num; i++) |

**IMPORTANT POINTS –**

1. While the for loop the data inside the parenthesis indicates the

Initial expression ,Test expression and Update expression.

1. Here “factorial\*=I” means factorial = factorial\*I.

**PROGRAM-5**

**AIM- Write a java program to find the fibonacci sequence of a number.**

public class FibonacciSeries {

    public static void main(String[] args) {

        int n = 10; // Number of terms

        int firstTerm = 0;

        int secondTerm = 1;

System.out.println("Fibonacci Series till " + n + " terms:");

        // Print the first two terms

        System.out.print(firstTerm + " " + secondTerm);

        for (int i = 3; i <= n; i++) {

            int nextTerm = firstTerm + secondTerm;

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

            firstTerm = secondTerm;

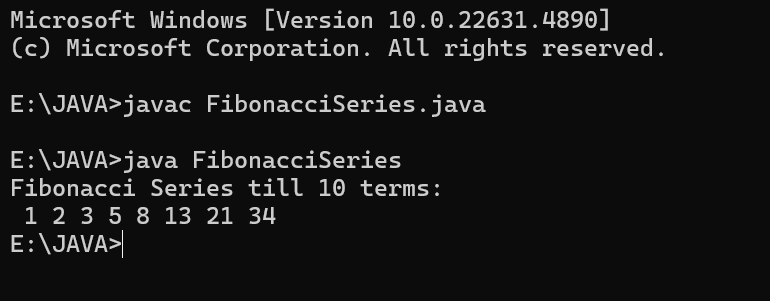
            secondTerm = nextTerm;

        }

    }

}

**OUTPUT-**



**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| **S.No:** | **Errors** | **Error rectification** |
| **1)** | Out put was 3 5 8 13 21 34, 1st and 2nd term did not get printed. | To print 1st and 2nd term = System.out.println("Fibonacci Series till " + n + " terms:"); |
| **2)** | Not giving parenthesis after closing the input.   |  | | --- | |  | | We must put parenthesis after closing the input. |

**IMPORTANT POINTS –**

1. "next" can refer to the next term to be generated or displayed based on the user's request for a certain number of terms.
2. **Starts at i = 3**: Since the first two terms are printed before the loop.
3. **Ends at i <= n**: To ensure n terms are generated.

**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”.**

public class Car {

// Attributes

private String car\_color;

private String car\_brand;

private String fuel\_type;

private double mileage;

// Default Constructor

public Car() {

System.out.println("Welcome to Meg Car Showroom");

}

public Car(String car\_color, String car\_brand, String fuel\_type, double mileage) {

this.car\_color = car\_color;

this.car\_brand = car\_brand;

this.fuel\_type = fuel\_type;

this.mileage = mileage;

}

public void setCar\_color(String car\_color) {

this.car\_color = car\_color;

}

public void setCar\_brand(String car\_brand) {

this.car\_brand = car\_brand;

}

public void setFuel\_type(String fuel\_type) {

this.fuel\_type = fuel\_type;

}

public void setMileage(double mileage) {

this.mileage = mileage;

}

public String getCar\_color() {

return car\_color;

}

public String getCar\_brand() {

return car\_brand;

}

public String getFuel\_type() {

return fuel\_type;

}

public double getMileage() {

return mileage;

}

// Methods

public void start() {

System.out.println(car\_brand + " is starting.");

}

public void service() {

System.out.println(car\_brand + " is being serviced.");

}

public void stop() {

System.out.println(car\_brand + " has stopped.");

}

public static void main(String[] args) {

// Creating objects using parameterized constructor

Car car1 = new Car("Red", "Mercedez", "Electric", 18.5);

car1.start();

System.out.println("Car 1 - Color: " + car1.getCar\_color() + ", Brand: " + car1.getCar\_brand() + ", Fuel: " + car1.getFuel\_type() + ", Mileage: " + car1.getMileage());

Car car2 = new Car("Blue", "RR", "Diesel", 20.0);

car2.service();

System.out.println("Car 2 - Color: " + car2.getCar\_color() + ", Brand: " + car2.getCar\_brand() + ", Fuel: " + car2.getFuel\_type() + ", Mileage: " + car2.getMileage());

Car car3 = new Car("Black", "Ford", "Electric", 0);

car3.stop();

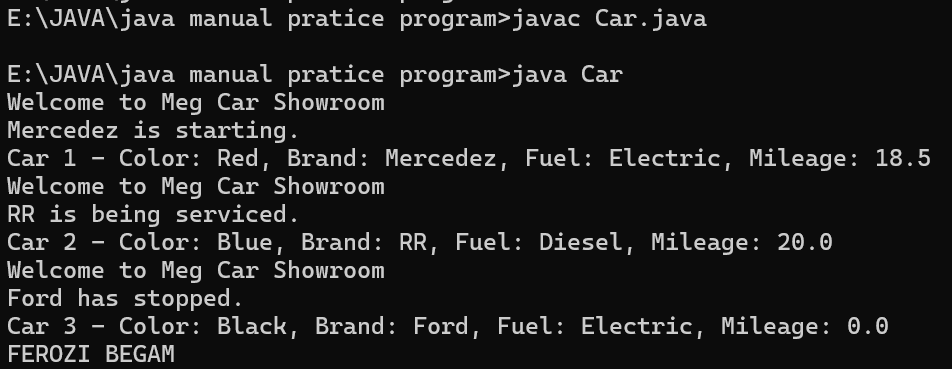
System.out.println("Car 3 - Color: " + car3.getCar\_color() + ", Brand: " + car3.getCar\_brand() + ", Fuel: " + car3.getFuel\_type() + ", Mileage: " + car3.getMileage());

System.out.println("FEROZI BEGAM");

}

}

**OUTPUT-**

****

**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Error** | **Error rectification** |
| **1)** | Class name-Car  File name-car | Class name and file name should be same.  Otherwise it wont be complied. |
| **2)** | After Start, Stop, Service not giving the parenthesis ( ). | After every method, put the parenthesis ( ). |
| **3)** | System.out.println("Car3Color:"+car3.getCar\_color() + ", Brand: " + car3.getCar\_brand() + ", Fuel: " + car3.getFuel\_type()+",Mileage:"+car3.getMileage()) | Check for missing or extra quotes, plus signs, or misplaced semicolons. |

**IMPORTANT POINTS –**

1. Here, the “public void start( )” indicates that we are writing a method to call the function.
2. this.car\_color = car\_color; is used inside a constructor to assign the passed parameter to the instance variable of the same name.
3. Car() is a constructor that gets executed when an object of the Car class is created, printing "Welcome to Carshowdown".
4. Here, the “public void start( )” indicates that we are writing a method to call the function.

**CLASS DIAGRAM-**

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

**PROGRAM-2**

**AIM- 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 update the current amount.**
2. **withdraw(): Whenever an amount is withdrawn, it has to be less than the current amount , else print (“Insufficient funds”) .**

public class BankAccount {

    private String name;

    private int AccountNum, CurrBal;

    // Constructor

    public BankAccount(String name, int AccountNum, int CurrBal) {

        this.name = name;

        this.AccountNum = AccountNum;

        this.CurrBal = CurrBal;

        System.out.println("The customer's details are: " + name + " " + AccountNum + " " + CurrBal);

    }

    // Method for withdrawal

    public void withdraw(int WAmt) {

        if (WAmt < CurrBal) {

            CurrBal = CurrBal - WAmt;

            System.out.println("After withdrawal, the current balance is: " + CurrBal);

        } else {

            System.out.println("Insufficient Funds");

        }

    } // Withdraw method ends

    // Method for deposit

    public int deposit (int DAmt) {

        CurrBal = CurrBal + DAmt;

        return CurrBal;

    } // Deposit method ends

    public static void main(String[] args) {

        // Object

        BankAccount cust1 = new BankAccount("FEROZI", 45988, 123698);

        cust1.withdraw(20000);

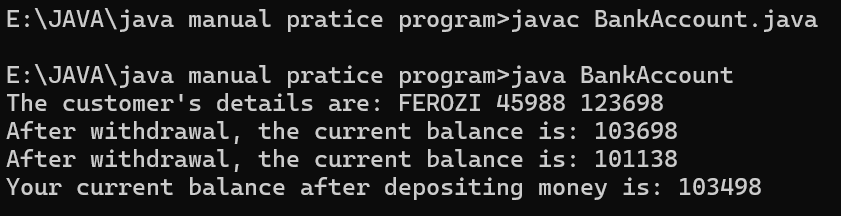
        cust1.withdraw(2560);

        System.out.println("Your current balance after depositing money is: " + cust1.deposit(2360));

    }

}

**OUTPUT-**



**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | 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; |

**IMPORTANT POINTS –**

1. Classes and Objects – The program defines a BankAccount class and creates an object (cust1) to access methods and store account details.
2. Constructors – The constructor BankAccount(String name, int AccNo, int CurrBal) initializes the object with values when a new bank account is created.
3. Instance Variables – The program uses name, AccNo, and CurrBal as private instance variables to store customer details and account balance.
4. Access Modifiers – The private keyword ensures that instance variables cannot be accessed directly from outside the class, maintaining encapsulation.
5. Methods (Functions) – The withdraw(int WAmt) method deducts money from the balance, and deposit(int DAmt) adds money and returns the updated balance.
6. Conditional Statements – The if-else condition in withdraw checks if the withdrawal amount is less than the current balance before proceeding.
7. Return Statements – The deposit method returns the updated balance after adding the deposited amount.
8. Printing Output (System.out.println) – The program prints account details, withdrawal status, and the new balance after deposits.
9. main Method – The program starts execution from the main method, where an object is created, and methods are called.
10. Basic Error Debugging – Understanding common Java errors like misspelled method names (depost → deposit), incorrect keywords (retirn → return), and missing braces helps in fixing compilation issues.

**CLASS DIAGRAM-**

|  |
| --- |
| BankAccount |
| - name: String  - Accno: int  - CurrBal: int |
| BankAccount: void  + withdraw(int WAmt): void  + deposit(int DAmt): int |

**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 and display the details of two books.**

class Book {

    // Declare attributes

    String titleOfTheBook;

    String author;

    int yearOfPublication;

    // Constructor to initialize values

    public Book(String titleOfTheBook, String author, int yearOfPublication) {

        this.titleOfTheBook = titleOfTheBook;

        this.author = author;

        this.yearOfPublication = yearOfPublication;

    }

    // Create a method to display book details

    public void getBook() {

        System.out.println("The title of the book: " + titleOfTheBook);

        System.out.println("Author: " + author);

        System.out.println("Year of publication: " + yearOfPublication);

    }

    public static void main(String[] args) {

        Book book1 = new Book("Treasure Island", "Robert Louis Stevenson", 1883);

        book1.getBook();

        Book book2 = new Book("The Lord of the Rings: The Fellowship of the Ring", "J.R.R. Tolkien", 1954);

        book2.getBook();

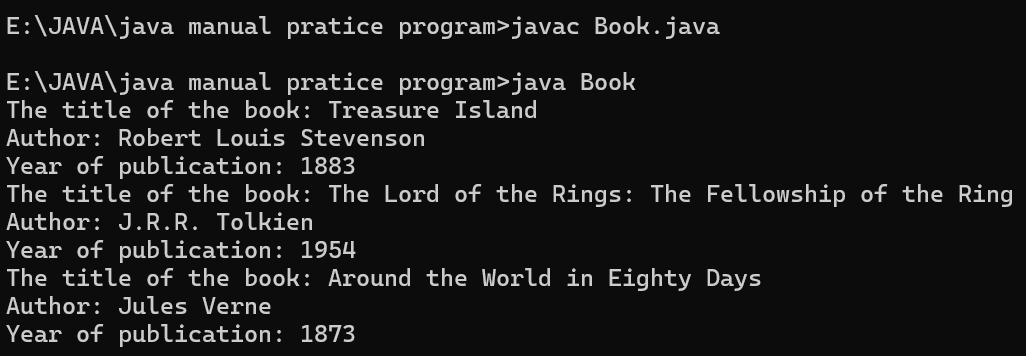
        Book book3 = new Book("Around the World in Eighty Days", "Jules Verne", 1873);

        book3.getBook();

    }

}

**OUTPUT-**



**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | Error rectification |
| 1. | Forgot to add the details  Book book3 = new Book("-", "-", -); | Book book3 = new Book("Around the World in Eighty Days", "Jules Verne", 1873); |
| 2. | Did not give data type to yearOfPublication; | int yearOfPublication; |

**IMPORTANT POINTS –**

1. (public Book(String titleOfTheBook, String author, int yearOfPublication)):

* This method initializes the attributes of the Book class when an object is created.
* It ensures that all necessary details (title, author, year of publication) are provided when creating a new book object.

1. Public String getTitle():Return the title of the book.
2. Public String get()Author(): the author of the book.
3. Public String getYearOfPublication():Return the year of publication of the book.
4. public void getBook(): This method prints out the details of the book, including its title, author, and year of publication.

**CLASS DIAGRAM-**

|  |
| --- |
| Book   * Title: String * Author: String * Year of publication: int   + Book(title: String,  Author: String;  Year of publication: int  + displayDetails( ): void |

**PROGRAM-2**

**AIM- Create a java Program with class named myclass with static variable count of int type, initialized to zero and a constant variable “pi” of type double initialized to 3.14 as attributes of the class, ow define a constructor for “myclass” that increments the count variable each time an object of my class is created (count++), finally print the final values of count and pi variables create three objects.**

public class MyClass {

    static int count = 0;

    final double pi = 3.14;

    // Constructor with proper declaration

    public MyClass() {

        count = count + 1;

    }

    public void display() {

        System.out.println("Count is: " + count);

        System.out.println("Double is: " + pi);

        System.out.println();

    }

    public static void main(String[] args) {

        MyClass Asec = new MyClass();

        Asec.display();

        MyClass Bsec = new MyClass();

        Bsec.display();

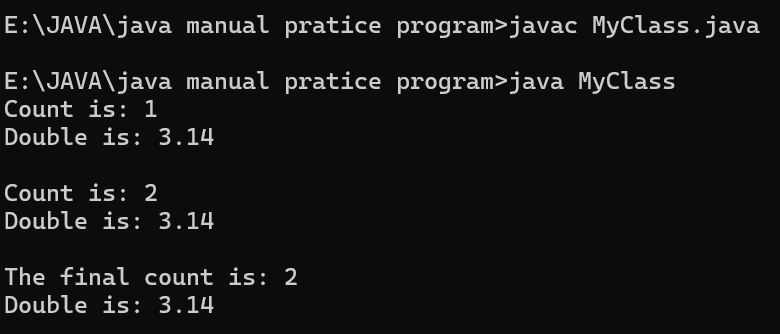
        System.out.println("The final count is: " + count);

        System.out.println("Double is: " + Bsec.pi);

    }

}

**OUTPUT-**



**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | Error rectification |
| 1. | Class name=MyClass  File name=Myclass | Class name and file name should be same.  Otherwise it wont be complied. |

**IMPORTANT POINTS –**

* **Asec.display()** and **Bsec.display()** access the instance methods and variables through their respective object references.
* **System.out.println(“Double is :”+Bsec.pi);** accesses that pi variable of the Bsec object.
* new keyword followed by the class constructor. This allocates memory for the object and initializes its attributes.
* **new** is necessary for creating objects and invoking constructors.
* **Object References** are needed to access instance variables and methods.
* final double pi means that once pi is initialized with the value 3.14, it cannot be changed.

**CLASS DIAGRAM-**

|  |
| --- |
| Myclass   * Count: int * Pi: double   + myclass( )  + main(args: String[]): void |

**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 addition{

public int add(int a, int b)

{

int addition = a+b;

return addition;

}}

class subtraction extends addition

{

public int sub(int a, int b)

{

int subtraction = a-b;

return subtraction;

}

}

class multiplication extends subtraction

{

public int mult(int a, int b)

{

int multiplication = a\*b;

return multiplication;

}

}

class division extends multiplication

{

public int div(int a,int b)

{

int division = a/b;

return division;

}

}

class calculator

{

public static void main(String[] args)

{

division obj = new division();

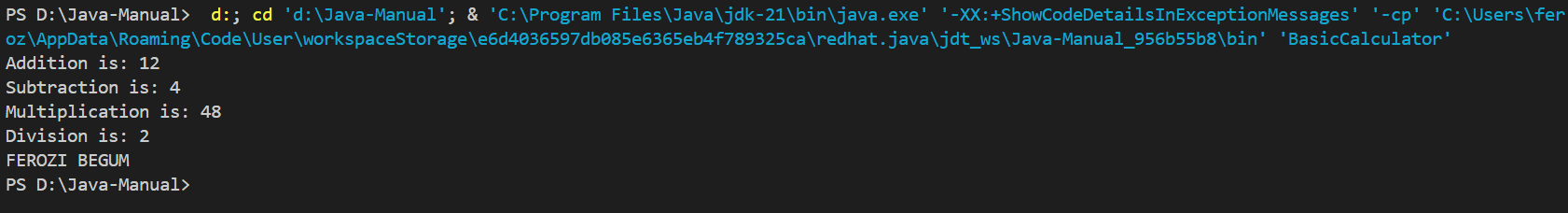
System.out.println("Addition is:"+ obj.add(10,2));

System.out.println ("Subtraction is:"+obj.sub(8,4));

System.out.println("Multiplication is:"+obj.mult(12,4));

System.out.println("Division is:"+obj.div(8,4));

} }

**OUTPUT- **

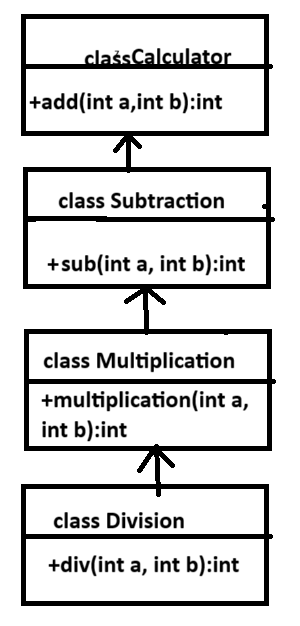
**ERROR RECTIFICATION-**

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Error Type | Cause | Rectification |
| 1 | Constructor error | Invalid name to method | Defined class name |
| 2 | Syntax error | Expected ‘()’ | Added parenthesis |
| 3 | Logical error | Incorrect arithmetic  operation | Correct operation  rectified |

**IMPORTANT POINTS –**

* The code demonstrates multilevel inheritance where division  extend Multiplication , Multiplication extends Subtraction, and Subtraction extends Calculator.
* This means division inherits methods from all its parent classes (Calculator, Subtraction, Multiplication).

**CLASS DIAGRAM-**

****

**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 Vehicle {

    private String brand;

    private int speed;

    Vehicle(String brand, int speed) {

        this.brand = brand;

        this.speed = speed;

    }

    void details() {

        System.out.println("Brand: " + brand);

        System.out.println("Speed: " + speed + " km/h");

    }

}

class Car extends Vehicle {

    private int doors;

    private int capacity;

    Car(String brand, int speed, int doors, int capacity) {

        super(brand, speed);

        this.doors = doors;

        this.capacity = capacity;

    }

    void carDetails() {

        System.out.println("Number of doors: " + doors);

        System.out.println("Capacity: " + capacity + " persons");

    }

    @Override

    void details() {

        super.details();

        carDetails();

    }

}

class Bike extends Vehicle {

    private boolean gears;

    Bike(String brand, int speed, boolean gears) {

        super(brand, speed);

        this.gears = gears;

    }

    void bikeDetails() {

        System.out.println(gears ? "This bike has gears." : "This bike does not have a gear system.");

    }

    @Override

    void details() {

        super.details();

        bikeDetails();

    }

}

class Truck extends Vehicle {

    private int tons;

    Truck(String brand, int speed, int tons) {

        super(brand, speed);

        this.tons = tons;

    }

    void truckDetails() {

        System.out.println("The capacity of truck is: " + tons + " tons.");

    }

    @Override

    void details() {

        super.details();

        truckDetails();

    }

}

public class Rent {

    public static void main(String[] args) {

        System.out.println("Car Details:");

        Car c = new Car("Ferrari", 120, 2, 2);

        c.details();

        System.out.println("\nBike Details:");

        Bike b = new Bike("KTM", 80, true);

        b.details();

        System.out.println("\nTruck Details:");

        Truck t = new Truck("TATA", 100, 10);

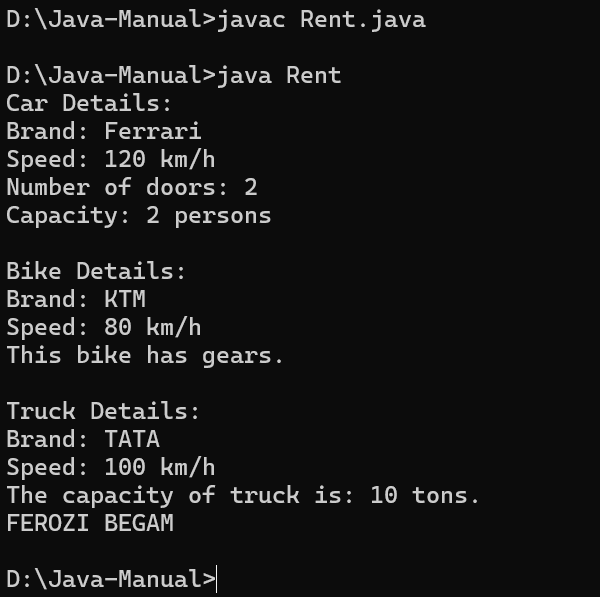
        t.details();

        System.out.println("FEROZI BEGAM");

    }

}

**OUTPUT-**



**ERROR RECTIFICATION-**



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

**IMPORTANT POINTS –**

* Inheritance is used to derive Car, Bike, and Truck from Vehicle.
* Encapsulation is maintained by keeping variables private and using methods for access.
* Method Overriding allows child classes to add their own details after the base details.

**CLASS DIAGRAM-**

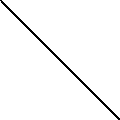
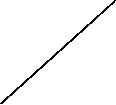


|  |
| --- |
| Vehicle |
| -Brand:String  -Speed:int |
| + Vehicle()  + details(): void |

|  |
| --- |
| Car |
| -doors:int  -capacity:int |
| + carDetails()  +details() |

|  |
| --- |
| Truck |
| -tons:int |
| -truckDetails()  +details() |

|  |
| --- |
| Bikes |
| -gears:bool |
| + bikeDetails()  + details() |



**WEEK-6**

**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 a car.**

class Vehicle {

    String car\_company;

    String car\_model;

    long car\_price;

    int seating\_capacity;

    boolean petrol;

    Vehicle(String car\_company, String car\_model, long car\_price, int seating\_capacity, boolean petrol) {

        this.car\_company = car\_company;

        this.car\_model = car\_model;

        this.car\_price = car\_price;

        this.seating\_capacity = seating\_capacity;

        this.petrol = petrol;

    }

    void displayInfo() {

        System.out.println("Car company: " + car\_company);

        System.out.println("Car model: " + car\_model);

        System.out.println("Car price: " + car\_price);

        System.out.println("Car seating capacity: " + seating\_capacity);

        System.out.println("Car uses petrol: " + petrol);

    }

}

class Car extends Vehicle {

    Car(String car\_company, String car\_model, long car\_price, int seating\_capacity, boolean petrol) {

        super(car\_company, car\_model, car\_price, seating\_capacity, petrol);

    }

}

public class Main {

    public static void main(String[] args) {

        // Creating a Car object with correct arguments

        Car c1 = new Car("Toyota", "Camry", 3000000, 5, true);

        c1.displayInfo();

        Car c2 = new Car("Mercedes", "Benz", 5000000, 2, false);

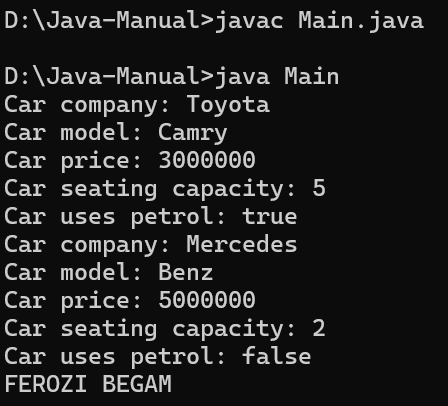
        c2.displayInfo();

        System.out.println("FEROZI BEGAM");

    }

}

**OUTPUT-**



**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | Error rectification |
| 1. | Incorrect class name for main method(Truck). | Rename Truck to Main or place main inside car or vehicle. |
| 2. | Inconsistent car model output in displayinfo(). | Ensure Car correctly passes Toyota” to super(car\_model,color,fueltype) |

**IMPORTANT POINTS –**

* Constructor Chaining:The Car class calls the parent constructor using super(car\_model, color, fuel\_type); to initialize inherited attributes.
* 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.
* 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 for undergraduate(UG) and postgraduate(PG) programs. Each program has different eligibility criteria based on the student's percentage in their previous qualification.**

**(I)UG Admissions require a minimum of 60%.**

**(II)PG Admissions require a minimum of 70%.**

class Student {

    String name;

    double percentage;

    Student(String name, double percentage) {

        this.name = name;

        this.percentage = percentage;

    }

    void studentsInfo() {

        System.out.println("Student Name: " + name);

        System.out.println("Percentage: " + percentage);

    }

}

class UG extends Student {

    UG(String name, double percentage) {

        super(name, percentage);

    }

    void checkEligibility() {

        if (percentage >= 60) {

            System.out.println(name + " is eligible for admission in UG.");

        } else {

            System.out.println(name + " is not eligible for admission in UG.");

        }

    }

}

class PG extends Student {

    PG(String name, double percentage) {

        super(name, percentage);

    }

    void checkEligibility() {

        if (percentage >= 70) {

            System.out.println(name + " is eligible for admission in PG.");

        } else {

            System.out.println(name + " is not eligible for admission in PG.");

        }

    }

}

public class AutomatedAdmission {

    public static void main(String[] args) {

        UG ug = new UG("Ferozi", 85);

        ug.studentsInfo();

        ug.checkEligibility();

        PG pg = new PG("Madhav", 90);

        pg.studentsInfo();

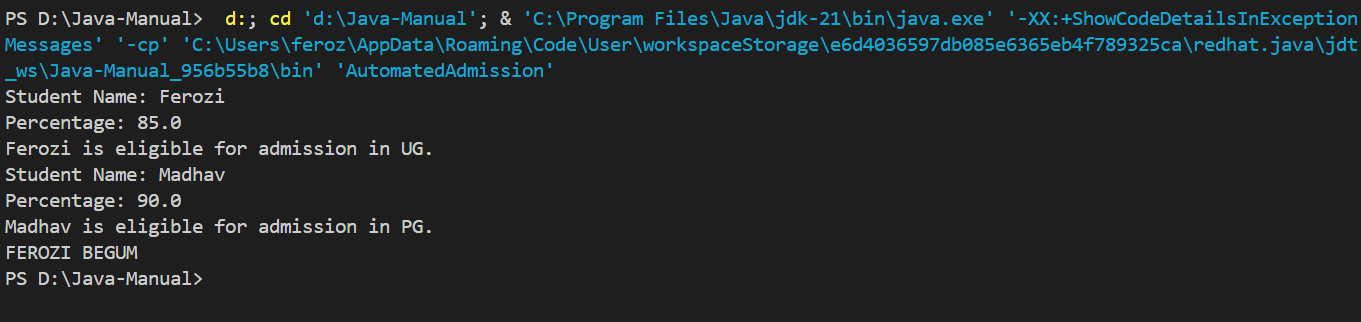
        pg.checkEligibility();

        System.out.println("FEROZI BEGUM");

    }

}

**OUTPUT-**

****

**ERROR RECTIFICATION-**

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

**IMPORTANT POINTS –**

* User Input Handling: Uses Scanner to take user input for name, percentage, and program type.
* Decision Making with Conditions: Uses if-else statements to check eligibility criteria
* String Handling: Converts program input to uppercase (toUpperCase()) to handle case variations.

**CLASS DIAGRAM-**



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

**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 Addition {

    public int add(int a, int b) {

        return a + b;

    }

    public double add(double a, double b) {

        return a + b;

    }

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

        return a + b + c;

    }

}

class Calculator {

    public static void main(String[] args) {

        Addition add = new Addition();

        System.out.println("add 2 int = " + add.add(10, 20));

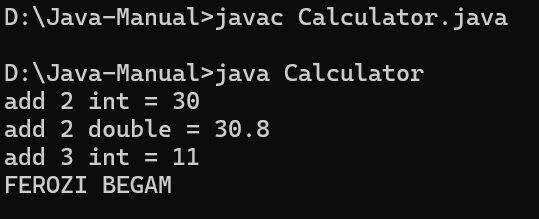
        System.out.println("add 2 double = " + add.add(10.5, 20.3));

        System.out.println("add 3 int = " + add.add(1, 3, 7));

System.out.println(“FEROZI BEGAM”);

    } }

**OUTPUT-**

****

**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | Error rectification |
| 1. | Method parameters missing spaces. E.g.,”inta, intb”should be “int a, int b”. | Add proper spacing between parameters: (int a, int b) |
| 2. | Class name =addition  Entered wrong class name . | Class name=Calculator |

**IMPORTANT POINTS –**

**CLASS DIAGRAM-**

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

**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 Shape {

    public double calculateArea(double side) {

        return side \* side; // Square area

    }

    public double calculateArea(double length, double breadth) {

        return length \* breadth; // Rectangle area

    }

}

class Circle extends Shape {

    @Override

    public double calculateArea(double radius) {

        return 3.14 \* radius \* radius; // Circle area

    }

}

public class findAreaShape {

    public static void main(String[] args) {

        Shape s = new Shape();

        Circle c = new Circle();

        System.out.println("Area of square is: " + s.calculateArea(5.0));

        System.out.println("Area of the rectangle is: " + s.calculateArea(3.5, 7.8));

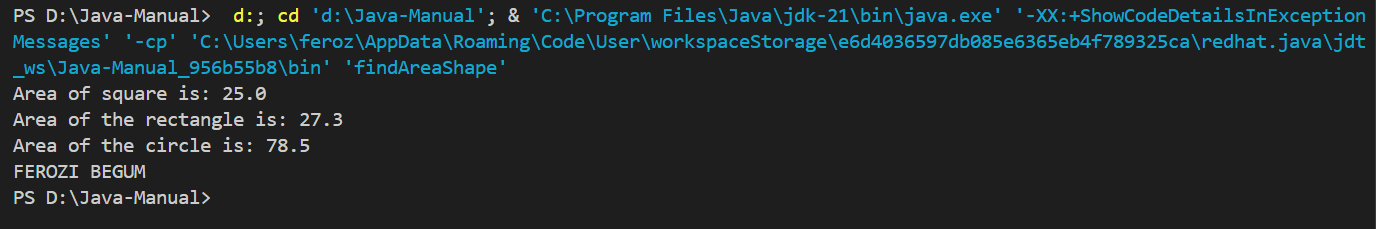
        System.out.println("Area of the circle is: " + c.calculateArea(5.0));

        System.out.println("FEROZI BEGUM");

    }

}

**OUTPUT-**

****

**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | Error rectification |
| 1. | Circle class method does not override theparent class method properly. | Ensure @Override is used, and the method signature should match correctly. |
| 2. | Only created 1 obj  Shape s = new Shape();  Output= Area of square is: 25.0  Area of the rectangle is: 27.3  Circle is not there | We should create 2 obj  Shape s = new Shape();  Circle c = new Circle();  Output= Area of square is: 25.0  Area of the rectangle is: 27.3  Area of the circle is: 78.5 |

**IMPORTANT POINTS –**

* Inheritance: Circle class extends Shape, inheriting its methods**.**
* Method Overloading: Shape has multiple calculateArea methods with different parameters**.**
* Method Overriding: Circle overrides calculateArea from Shape to implement its own formula.

**CLASS DIAGRAM-**



|  |
| --- |
| **Shape** |
| +calculateArea(double side) : double  +calculateArea(double ,double b) : double |



|  |
| --- |
| **Circle** |
| + calculateArea(double radius) : double |

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

**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 extent the Animal class and implement the sound() method to make a specific for each animal.**

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



|  |
| --- |
| Tiger |
| +sound() |

|  |
| --- |
| Lion |
| +sound() |



**CODE-**

abstract class Animal{

    abstract void sound();

}

class lion extends Animal{

    @Override

    public void sound(){

        System.out.println("the lion is roaring!!");

    }

}

class tiger extends Animal{

    @Override

    public void sound(){

     System.out.println("the tiger is rearing!!");

    }

}

class landtSounds{

    public static void main(String[] args) {

        lion l = new lion();

        l.sound();

        tiger t =new tiger();

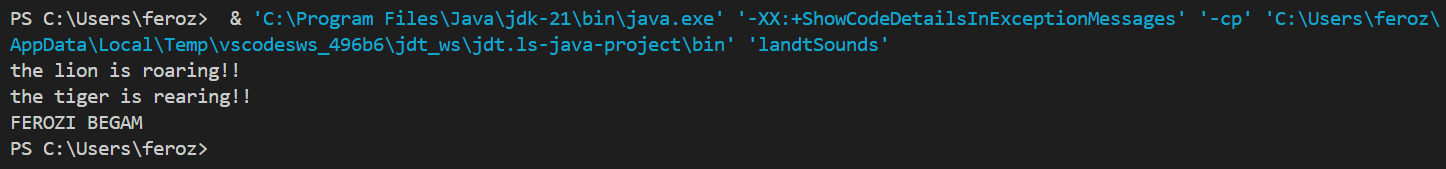
        t.sound();

        System.out.println("FEROZI BEGAM");

    }

}

**OUTPUT-**



**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | Error rectification |
| 1. | Not using polymorphism to handle multiple animals. | Use Animal a = new Lion(); a.sound(); and Animal b = new Tiger(); b.sound(); to demonstrate polymorphism. |
| 2. | Not using proper access modifiers (e.g., classes and methods default to package-private). | Use public for classes and methods if they need to be accessed from other packages or classes. |
| 3. | Forgetting to use @Override annotation when overriding abstract method sound(). | Always use @Override to ensure the method properly overrides the abstract method in the superclass. |

**IMPORTANT POINTS –**

**PROGRAM-2**

**AIM- Write a java program to create an abstract class shape 3D with abstract method called sound().Create subclass sphere and cude that extend the shape 3D class and implement.**

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



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

|  |
| --- |
| Cube |
| - side: int |
| + calculateVolume(): double + calculateSurfaceArea(): double |



**CODE-**

abstract class Shape3D{

abstract double calculateVolume();

abstract double calculateSurfaceArea();

}

class Sphere extends Shape3D{

int radius;

public Sphere(int radius){

this.radius = radius;

}

@Override

public double calculateVolume(){

return (4.0/3.0) \* Math.PI \* Math.pow(radius, 3);

}

@Override

public double calculateSurfaceArea(){

return 4 \* Math.PI \* Math.pow(radius, 2);

}

}

class cube extends Shape3D{

int side;

public cube(int side){

this.side = side;

}

@Override

public double calculateVolume(){

return Math.pow(side,3);

}

@Override

public double calculateSurfaceArea(){

return 6 \* Math.pow(side, 2);

}

}

class Shapein3D{

public static void main(String[] args) {

cube c = new cube(5);

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

System.err.println("Surface Area="+ c.calculateSurfaceArea());

Sphere s = new Sphere(5);

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

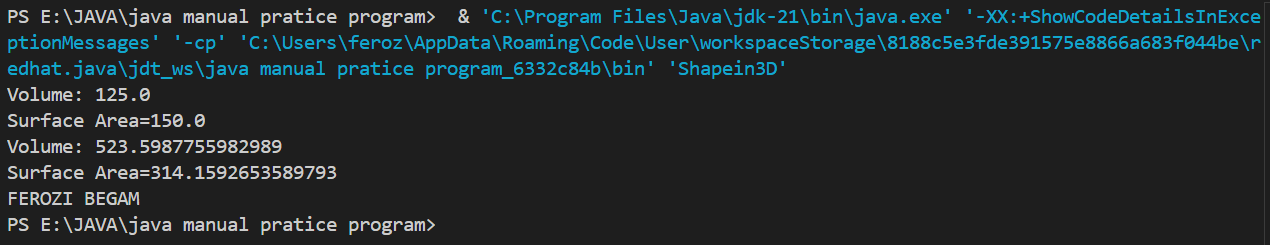
System.err.println("Surface Area="+ s.calculateSurfaceArea());

System.out.println("FEROZI BEGAM");

}

}

**OUTPUT-**

****

**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | Error rectification |
| 1. | Class name cube starts with lowercase letter, violating Java naming conventions. | Rename class to Cube (capitalize first letter) to follow Java conventions. |
| 2. | No access modifiers specified for classes and fields | Use public for classes and fields if they need to be accessed outside their package. |

**IMPORTANT POINTS –**

* Abstract Class Usage:Shape3D is an abstract class with two abstract methods: calculateVolume() and calculateSurfaceArea(). This enforces that all subclasses provide their own implementation.
* You can create references of type Shape3D and assign objects of Sphere or Cube to them, allowing polymorphic 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 pattern 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 patterns for a given number of rows.**

**Example Output for n = 5:**

**Star Pattern Number Pattern**

**1 \***

**1 2 \* \***

**1 2 3 \* \* \***

**1 2 3 4 \* \* \* \***

**1 2 3 4 5 \* \* \* \* \***

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



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

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



CODE-

abstract class patternPrinting{

public abstract void printPattern(int n);

public void displayTitle(String title){

System.out.println("The pattern name ="+title);

}

}

class StarPattern extends patternPrinting{

public 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 patternPrinting{

public 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 patterns{

public static void main(String[] args) {

patternPrinting p = new StarPattern();

p.displayTitle("Star Pattern");

p.printPattern(5);

numberPattern k=new numberPattern();

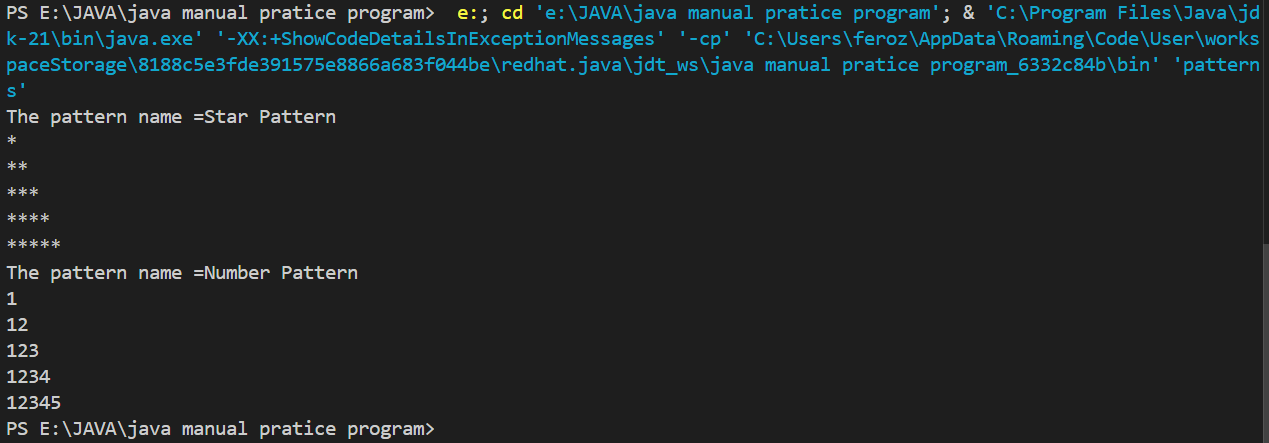
k.displayTitle("Number Pattern");

k.printPattern(5);

}

}

**OUTPUT-**



**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | Error rectification |
| 1. | Missing @Override annotation on overridden printPattern methods. | Add @Override annotation above printPattern methods in subclasses to ensure proper overriding. |
| 2. | Subclass doesn’t override abstract method. | implement printpattern()in all subclasses. |

**IMPORTANT POINTS –**

* patternPrinting is an abstract class with an abstract method printPattern(int n). This enforces subclasses to provide their own pattern printing logic.
* StarPattern prints a right-angled triangle of stars, and numberPattern prints a right-angled triangle of numbers.

**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.**

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



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

|  |
| --- |
| Triangle |
| - side1: double  - side2: double  - side3: double |
| + getPerimeter(): double |

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

**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 side1, side2, side3;

Triangle(double side1, double side2, double side3) {

this.side1 = side1;

this.side2 = side2;

this.side3 = side3;

}

public double getPerimeter() {

return side1 + side2 + side3;

}

}

public class perimeter {

public static void main(String[] args) {

Shape s;

s = new Rectangle(4, 6);

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

s = new Circle(5);

s.getPerimeter();

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

s = new Triangle(3, 4, 5);

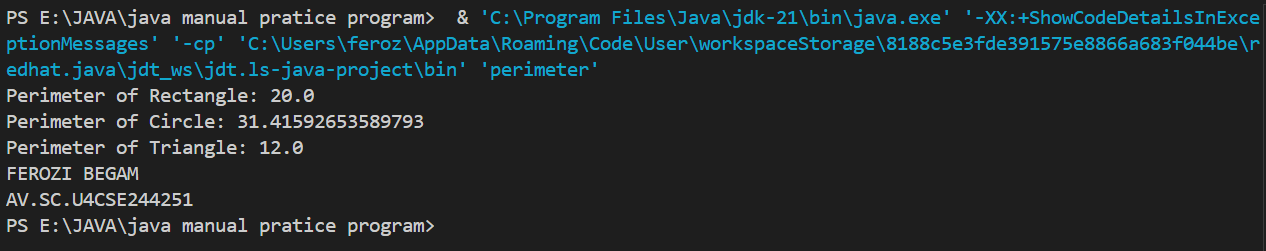
System.out.println("Perimeter of Triangle: " + s.getPerimeter());

System.out.println("FEROZI BEGAM\nAV.SC.U4CSE244251");

}

}

**OUTPUT-**



**ERROR RECTIFICATION-**

|  |  |  |
| --- | --- | --- |
| Sno. | Error | Error rectification |
| 1. | Missing @Override annotation on getPerimeter() methods. | Add @Override to all getPerimeter() implementations for clarity. |
| 2. | Using s as a generic variable name for polymorphic Shape. | Use descriptive names like rectangle, circle, or triangle where possible. |

**IMPORTANT POINTS –**

* The Shape interface enforces the implementation of getPerimeter(), ensuring all shapes adhere to a common contract.
* The Shape s reference demonstrates polymorphism by dynamically binding to Rectangle, Circle, and Triangle objects.
* Use Math.PI for accurate circle perimeter calculations. Avoid hardcoding π

**PROGRAM-2**

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

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



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

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

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



**CODE-**

interface Playable {

void play();

}

class Football implements Playable {

@Override

public void play() {

System.out.println("Running with the ball...");

System.out.println("Scoring a goal!");

}

}

class Volleyball implements Playable {

@Override

public void play() {

System.out.println("Serving the ball...");

System.out.println("Blocking the opponent's attack!");

}

}

class Basketball implements Playable {

@Override

public void play() {

System.out.println("Dribbling the ball...");

System.out.println("Shooting a three-pointer...");

System.out.println("Slam dunking!");

}

}

public class OutdoorGame {

public static void main(String[] args) {

Football football = new Football();

Volleyball volleyball = new Volleyball();

Basketball basketball = new Basketball();

System.out.println("\n--- Playing Sports ---\n");

football.play();

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

volleyball.play();

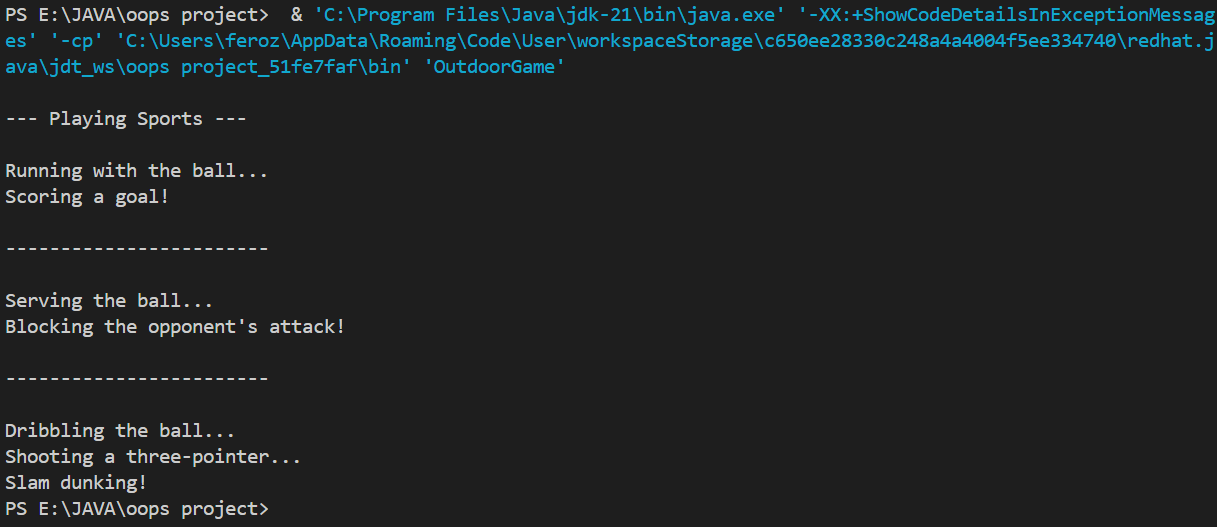
System.out.println("\n------------------------\n");

basketball.play();

}

}

**OUTPUT-**



**ERROR RECTIFICATION-**

|  |  |
| --- | --- |
| Error | Error Rectification |
| Forgetting to Use @Override | Use @Override |
| // Filename should be: OutdoorGame.java  public class Outdoorgame { | Corrected code // Filename should be: OutdoorGame.java  public class OutdoorGame { |
| Football football = new Playable(); didn’t create the obj correctly. | Playable football = new Football(); |

**IMPORTANT POINTS –**

* Interface implementation: All classes override the play() method from the Playable interface.
* Playable game = new Football();
  + game.play();
  + this setup supports **runtime polymorphism**:
* You **cannot instantiate** an interface:Playable p = new Playable();
* We can write the code this way also

Playable[] games = { new Football(), new Volleyball(), new Basketball() };

for (Playable game : games) {

game.play();

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

}