**Week – 5**

* **Aim:** To create a Java Program of a calculator using the operations including addition Subtraction, multiplication and division using multilevel inheritance & dis Play the desired out Put

**CODE:**

class SimpleCalculator{

//attributes, objects

int a;

int b;

//initialization

public static void add(int a, int b) {

System.out.println(a+ " + " + b + " = " + (a+b));

}

public static void diff(int a, int b) {

System.out.println(a+ " - " + b +" = " + (a-b));

}

}

class AdvCalculator extends SimpleCalculator {

// Initialization

public static void mul(int a, int b) {

System.out.println(a + " \* " + b + " = " + (a \* b));

}

}

class UltiCalculator extends AdvCalculator{

//initialization

public static void div(int a, int b) {

if(b == 0) {

System.out.println("Denominator should not be a zero ");

}

else{

System.out.println(a+ "/" + b +"=" + (a/b));

}

}

}

class Calc{

public static void main(String[] args) {

UltiCalculator u = new UltiCalculator();

System.out.println("The calculated values are: ");

u.add(5,5);

u.diff(6,5);

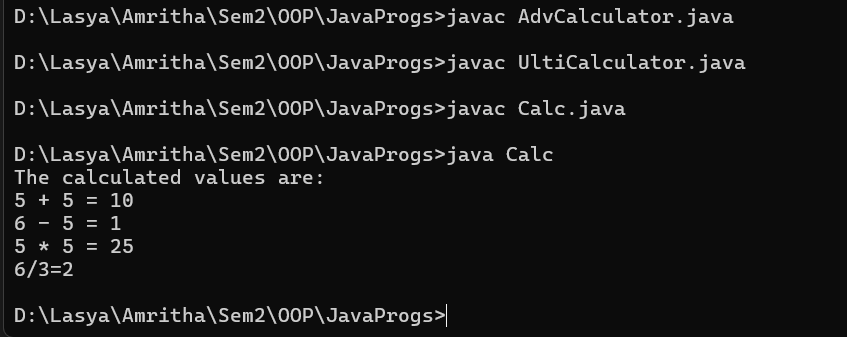
u.mul(5,5);

u.div(6,3);

} // end of the main function

} // end of the class

**OUTPUT:**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **Sl. no** | **Error** | **Error rectification** |
| 1. | error: unclosed string literal System.out.println(a+ " - " + b +" = + (a-b)); | Add a '”' after the ‘=’. |
| 2. | error: '(' or '[' expected UltiCalculator u = new UltiCalculator; | Add a '()' after class name. |

**Class Diagram:**

|  |
| --- |
| **SimpleCalculator** |
| a : double  b : double |
| + add (a,b) : void  + diff (a,b) : void |

|  |
| --- |
| **AdvCalculator** |
| + mul (a,b) : void |

|  |
| --- |
| **UltiCalculator** |
| + div (a,b) : void |

|  |
| --- |
| **Calc** |
| + main(String[]) |

**Concepts to be known:**

1. We must declare the initial value of the variable before declaring the final one.

2. here, the main objective is to increase the count according to the number of objects we make, i.e the count increases when the no.of objects are increasing.

* **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.

**CODE:**

class Vehicle {

    String brand;

    double speed;

    public Vehicle(String brand, double speed) {

        this.brand = brand;

        this.speed = speed;

    }

    public void displayDetails() {

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

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

    }

    public void start() {

        System.out.println(brand + " vehicle is starting...");

}

}

class Car extends Vehicle {

    int numberOfDoors;

    int seatingCapacity;

    public Car(String brand, double speed, int numberOfDoors, int seatingCapacity) {

        super(brand, speed);

        this.numberOfDoors = numberOfDoors;

        this.seatingCapacity = seatingCapacity;

    }

    public void displayCarDetails() {

        super.displayDetails();

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

        System.out.println("Seating capacity: " + seatingCapacity);

    }

    public void startCar() {

        super.start();

        System.out.println("Car is ready to go!");

    }

}

class Bike extends Vehicle {

    boolean hasGears;

    public Bike(String brand, double speed, boolean hasGears) {

        super(brand, speed);

        this.hasGears = hasGears;

    }

    public void displayBikeDetails() {

        super.displayDetails();

        System.out.println("Has gears: " + (hasGears ? "Yes" : "No"));

    }

    public void startBike() {

        super.start();

        System.out.println("Bike is ready to go!");

    }

}

class Truck extends Vehicle {

    double cargoCapacity;

    public Truck(String brand, double speed, double cargoCapacity) {

        super(brand, speed);

        this.cargoCapacity = cargoCapacity;

    }

    public void displayTruckDetails() {

        super.displayDetails();

        System.out.println("Cargo capacity: " + cargoCapacity + " tons");

    }

    public void startTruck() {

        super.start();

        System.out.println("Truck is ready to go!");

    }

}

public class VehicleRentalSystem {

    public static void main(String[] args) {

        Car car = new Car("Toyota", 150, 4, 5);

        Bike bike = new Bike("Yamaha", 120, true);

        Truck truck = new Truck("Volvo", 90, 10);

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

        car.displayCarDetails();

        car.startCar();

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

        bike.displayBikeDetails();

        bike.startBike();

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

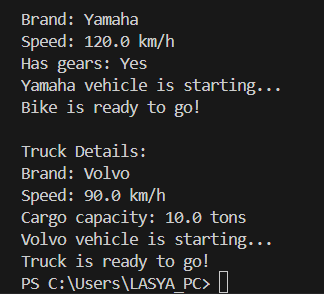
        truck.displayTruckDetails();

        truck.startTruck();

    }

}

**OUTPUT:**

****

**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **Sl. no** | **Error** | **Error rectification** |
| 1. | error: Declaring two super classes inside the same file. | Make two separate files to save the two super classes |
| 2. | error: Not declaring the variable using ‘this’ keyword inside the constructor. | Declare the variable using this keyword to run the program. |

**Concepts to be known:**

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.

**Class Diagram:**

|  |
| --- |
| **Vehicle** |
| - brand: string  - speed: double |
| +Vehicle()  + displayDetails() : void  + start() : void |

|  |
| --- |
| **Car** |
| - noOfDoors: int  - seatingCapacity: int |
| +Car()  + displayCarDetails() : void  + startCar() : void |

|  |
| --- |
| Bike |
| - hasGears: Boolean |
| +Bike()  + displayBikeDetails() : void  + startBike() : void |

|  |
| --- |
| **Truck** |
| - cargoCapacity: Boolean |
| + Truck()  +displayTruckDetails() : void  + startTruck() : void |

**Answer:**

The oops concepts used in the above program are:

Inheritance, encapsulation, polymorphism, abstraction.

To add a new vehicle type truck we need to create a truck class that will:

* Include an additional property capacity (in tons).
* Implement a showTruckdetials() method to display the truck's capacity.
* Implement a constructor for the truck class to initialize all its properties.

**Week – 6**

* **Aim:** Write a Java Program to create a Vehicle class with a method display(). Override this method in the Car subclass. Print car model, brand, petrol type, car color and provide the information about the car.

**Code :**

class Vehicle {

String brand;

String petrolType;

String color;

public void display() {System.out.println("This is a vehicle.");

}

}

class Car extends Vehicle {

Car(String brand, String petrolType, String color) {

this.brand = brand;

this.petrolType = petrolType;

this.color = color;

}

public void display() {

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

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

System.out.println("Petrol Type: " + petrolType);

System.out.println("Color: " + color);

}

}

public class Info {

public static void main(String[] args) {

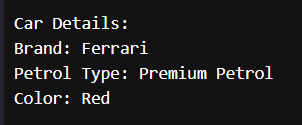
Vehicle myCar = new Car("Ferrari", "Premium Petrol", "Red");

myCar.display();

}

}

**Output :**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **Sl. no** | **Error** | **Error rectification** |
| 1. | error: class Main is public, should be declared in a file named Main.java  public class Main { | Save the file name as the name of the main class |

**Class Diagram:**

|  |
| --- |
| **Vehicle** |
| + brand : String  + petrolType : String  + color : String |
| + display() : void |

|  |
| --- |
| **Car** |
| Car(String brand, String petrolType, String color) |

**Concepts to be known:**

1. We use the concept of method overriding where the names of the methods in the different classes. The method of the parent class is overridden by the method of the child class

* **Aim:** A college is developing an automated admission system that verifies students’ eligibility for under-graduation and post-graduation. Each program has different eligibility criteria base on the percentage of students in their provided qualifications
* Ug requires 60%
* PG requires 70%

**Code :**

class Student{

String name;

double percentage;

Student(String name, double percentage){

this.name = name;

this.percentage = percentage;

}

public void Eligibility(){

System.out.println(name + " must meet the general admission criteria");

}

}

class UG extends Student{

UG(String name, double percentage){

super(name, percentage);

}

public void Eligibility(){

if (percentage>59){

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

}

else {

System.out.println(name + " is Not elligible for UG admission");

}

}

}

class PG extends Student{

PG(String name, double percentage){

super(name, percentage);

}

public void Eligibility(){

if (percentage>69){

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

}

else {

System.out.println(name + " is Not elligible for PG admission");

}

}

}

public class Admission {

public static void main (String[] args){

UG ug = new UG("Laila", 69);

PG pg = new PG("Majnu", 59);

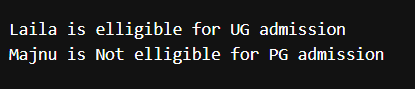
ug.Eligibility();

pg.Eligibility();

}

}

**Output :**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **Sl. no** | **Error** | **Error rectification** |
| 1. | error: ';' expected System.out.println(name + " is elligible for UG admission") | Add a ‘;’ after the print statement. |

**Class Diagram:**

|  |
| --- |
| **UG** |
| UG(String name, double percentage) |

|  |
| --- |
| **Student** |
| + name : String  + percentage : double |
| + Student(String name, double percentage): void  + Eligibility(): void |

|  |
| --- |
| **PG** |
| PG(String name, double percentage) |

**Concepts to be known:**

1. The variables once declared in the super class need not be declared twice in any of the sub classes.

2. super keyword is used in sub classes to access the methods of super classes, they are basically the reverse of overriding.

* **Aim:** To create a Java Program with class named “my class” with a Static Variable Count int type and initialize to 0 and A Constant Variable "pi" of type double initialized to 3.1415 has attributes of that class. Now defi a Constructor for my class that increments the Count Variable each time an object of my class is created. Finaly Print the final values of count.

**CODE:**

class AddCalculator{

    AddCalculator(){

        System.out.println("This is a calculator");

    }

    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;

    }

}

public class AddCalc{

    public static void main (String [] args){

    AddCalculator calc = new AddCalculator();

    System.out.println("The sum of the 2 numbers: 7 and 18 is "+ calc.add(7, 18));

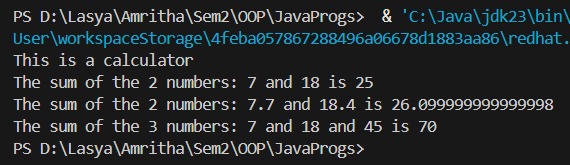
    System.out.println("The sum of the 2 numbers: 7.7 and 18.4 is "+ calc.add(7.7, 18.4));

    System.out.println("The sum of the 3 numbers: 7 and 18 and 45 is "+ calc.add(7, 18, 45));

    }

}

**OUTPUT:**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **Sl. no** | **Error** | **Error rectification** |
| 1. | error: Main method not defined | Add public static void main (String [] args){ |
| 2. | error: ';' return type is not expected | Change return type from int to double in its case. |

**Class Diagram:**

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

**Concepts to be known:**

1. We use the concept of method overloading where the names of the methods in the same class are same but the parameters are given different.

* **Aim:** Write a Java Program and create a Shape class with a method calcArea(). That is overloaded for different shapes like square and rectangle. Create a sub class circle that overrides the calcArea() for a circle.

**Code :**

class Shape{

int calcArea(int a){

return a\*a;

}

int calcArea(int b, int h){

return b\*h;

}

}

class Circle extends Shape{

double r;

double pi = 3.141592653589793;

Circle(double r){

this.r = r;

}

double calcArea(double r){

return pi\*r\*r;

}

}

public class AreaCalc {

public static void main(String[] args) {

Circle c = new Circle(7);

System.out.println("The area of circle is " + c.calcArea(7.7));

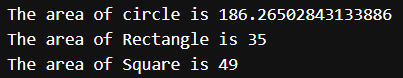
System.out.println("The area of Rectangle is " + c.calcArea(5, 7));

System.out.println("The area of Square is " + c.calcArea(7));

}

}

**Output :**



**ERRORS:**

|  |  |  |
| --- | --- | --- |
| **Sl. no** | **Error** | **Error rectification** |
| 1. | error: invalid method declaration; return type required  calcArea(int a){ | Enter the return type as per required. Here it is int |
| 2. | error: incompatible types: possible lossy conversion from double to int  return pi\*r\*r; | For calculating area of circle, we need to give return type double. |

**Class Diagram:**

|  |
| --- |
| **Circle** |
| + r : double  + pi : double |
| + calcArea(int r) : double |

|  |
| --- |
| **Shape** |
| + calcArea(int a) : int  + calcArea(int b, int h) : int |

**Concepts to be known:**

1. We use the concept of method overloading to calculate the area of square and rectangle in the parent class Shape.

2 we use method overriding in the child class Circle to calculate it’s area.