Task 01:

Violation of Liskov

abstract class Bird {

    abstract void fly() ;

}

class Eagle extends Bird {

    @Override

    public void fly() {

        sout(" Eagles fly");

    }

}

class Ostrich  extends Bird {

    @Override

    public void fly() { // dummy implentation

        sout("cant fly high but It lays big egg");

    }

}

class Driverclass{

    psvm(String[] args){

    }

}

abstract class Bird {

    abstract void fly();

}

class Eagle extends Bird {

    @Override

    public void fly() {

        System.out.println("Eagles fly");

    }

}

class Ostrich extends Bird {

    @Override

    public void fly() {

        System.out.println("Can't fly high but It lays big egg");

    }

}

class task01 {

    public static void main(String[] args) {

        Bird eagle = new Eagle();

        Bird ostrich = new Ostrich();

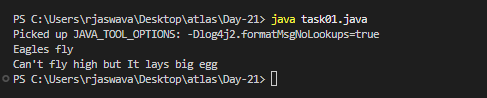
        eagle.fly();

        ostrich.fly();

    }

}

Output:



Task 02:

Implementation of Liskov

abstract class BirdsthatFly {

    abstract void fly() ;

}

abstract class BirdsthatDontFly {

    abstract void Speciality() ;

}

class Eagle extends BirdsthatFly {

    @Override

    public void fly() {

        sout(" Eagles fly");

    }

}

class Ostrich  extends BirdsthatDontFly {

    @Override

    public void Speciality() {

        sout("It lays big egg");

    }

}

class Driverclass{

    psvm(String[] args){

    }

}

abstract class BirdsThatFly {

    abstract void fly();

}

abstract class BirdsThatDontFly {

    abstract void speciality();

}

class Eagle extends BirdsThatFly {

    @Override

    public void fly() {

        System.out.println("Eagles fly");

    }

}

class Ostrich extends BirdsThatDontFly {

    @Override

    public void speciality() {

        System.out.println("It lays big egg");

    }

}

class task02 {

    public static void main(String[] args) {

        BirdsThatFly eagle = new Eagle();

        BirdsThatDontFly ostrich = new Ostrich();

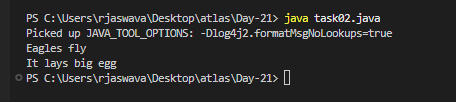
        eagle.fly();

        ostrich.speciality();

    }

}

Output:



Task 03:

Violation of Interface Segregation principle

interface ICalcShapesArea {

    calcArea();

    calcVolume();

}

class Circle implements ICalcShapesArea {

    calcArea() { sout()};

    calcVolume() { sout()}; // dummy implentation as it been forced

}

class Sphere implements ICalcShapesArea {

    calcArea() { sout()};

    calcVolume() { sout()};

}

class Driverclass {

    psvm(  ,...  ) {

    }

}

interface ICalcShapesArea {

    double calcArea();

    double calcVolume();

}

class Circle implements ICalcShapesArea {

    private double radius;

    public Circle(double radius) {

        this.radius = radius;

    }

    @Override

    public double calcArea() {

        System.out.println("Calculating Circle Area");

        return Math.PI \* radius \* radius;

    }

    @Override

    public double calcVolume() {

        System.out.println("Circle cannot have volume");

        return 0;

    }

}

class Sphere implements ICalcShapesArea {

    private double radius;

    public Sphere(double radius) {

        this.radius = radius;

    }

    @Override

    public double calcArea() {

        System.out.println("Calculating Sphere Surface Area");

        return 4 \* Math.PI \* radius \* radius;

    }

    @Override

    public double calcVolume() {

        System.out.println("Calculating Sphere Volume");

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

    }

}

class task03 {

    public static void main(String[] args) {

        ICalcShapesArea circle = new Circle(5);

        ICalcShapesArea sphere = new Sphere(5);

        circle.calcArea();

        circle.calcVolume();

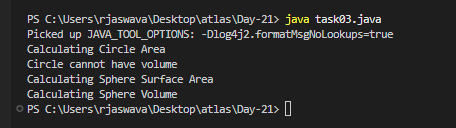
        sphere.calcArea();

        sphere.calcVolume();

    }

}

Output:



Task 04:

Implementation of Interface Segregation Principle

interface ICalcArea {

    calcArea(); calcPerimeter();

}

interface ICalcVolume {

    calcVolume();

}

class Circle implements ICalcArea {

    @Override

    calcArea() { sout()};

}

class Sphere implements ICalcArea, ICalcVolume {

     @Override

     calcArea() { sout()};

     calcVolume() { sout()};

}

class Driverclass {

    psvm(  ,...  ) {

    }

}

interface ICalcArea {

    double calcArea();

    double calcPerimeter();

}

interface ICalcVolume {

    double calcVolume();

}

class Circle implements ICalcArea {

    private double radius;

    public Circle(double radius) {

        this.radius = radius;

    }

    @Override

    public double calcArea() {

        System.out.println("Calculating Circle Area");

        return Math.PI \* radius \* radius;

    }

    @Override

    public double calcPerimeter() {

        System.out.println("Calculating Circle Perimeter");

        return 2 \* Math.PI \* radius;

    }

}

class Sphere implements ICalcArea, ICalcVolume {

    private double radius;

    public Sphere(double radius) {

        this.radius = radius;

    }

    @Override

    public double calcArea() {

        System.out.println("Calculating Sphere Surface Area");

        return 4 \* Math.PI \* radius \* radius;

    }

    @Override

    public double calcPerimeter() {

        System.out.println("Calculating Sphere Circumference");

        return 2 \* Math.PI \* radius;

    }

    @Override

    public double calcVolume() {

        System.out.println("Calculating Sphere Volume");

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

    }

}

class task04 {

    public static void main(String[] args) {

        Circle circle = new Circle(5);

        Sphere sphere = new Sphere(5);

        circle.calcArea();

        circle.calcPerimeter();

        sphere.calcArea();

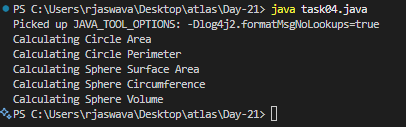
        sphere.calcPerimeter();

        sphere.calcVolume();

    }

}

Output:



Home Tasks:

Task 02:

class Animal {

void sound() {

sout(" sounds of different animals");

}

}

class Cat extends Animal{

@Override

void sound() {

sout(" Meow is the sound of cat");

}

}

class Main{

psvm(String[] args) {

Animal obj = new Cat();

obj.sound(); //Meow is the sound of cat

}

}

issue with Substitution  and Generics

Java Generics -- it has introduced  a challenge - substitution principle...

  is cat a subtype of Animal,  List<cat> is not a subtype of List<Animal>

List<Cat> Cobj = new ArrayList<>();

List<Animal> Aobj = Cobj;  ===// this will give you a wildcard ,

import java.util.ArrayList;

import java.util.List;

class Animal {

    void sound() {

        System.out.println("Sounds of different animals");

    }

}

class Cat extends Animal {

    @Override

    void sound() {

        System.out.println("Meow is the sound of cat");

    }

}

class task02 {

    public static void main(String[] args) {

        Animal obj = new Cat();

        obj.sound();

        List<Cat> catList = new ArrayList<>();

        List<? extends Animal> animalList = catList;

        catList.add(new Cat());

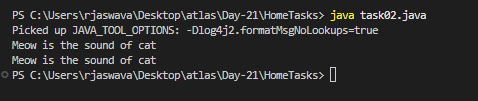
        Animal animal = animalList.get(0);

        animal.sound();

    }

}

Output:



Unbounded wildcards:

Task 03:

they are useful when the code does not depends on the actual type parmeter

void printList(List<?>  list) {

for(Object element: list) {

sout (element);

}

}

List<Cat> clist = new ArrayList<>();

clist.add(new Cat());

printList(clist); //

import java.util.ArrayList;

import java.util.List;

class Animal {

    @Override

    public String toString() {

        return "This is an Animal";

    }

}

class Cat extends Animal {

    @Override

    public String toString() {

        return "This is a Cat";

    }

}

class task03 {

    static void printList(List<?> list) {

        for(Object element: list) {

            System.out.println(element);

        }

    }

    public static void main(String[] args) {

        List<Cat> catList = new ArrayList<>();

        catList.add(new Cat());

        printList(catList);

        List<Animal> animalList = new ArrayList<>();

        animalList.add(new Animal());

        printList(animalList);

        List<String> stringList = new ArrayList<>();

        stringList.add("Hello");

        stringList.add("Hi");

        printList(stringList);

    }

}

Output:

