Amadou Sarjo Jallow

619016

Lab Solutions

1.

A. Explanation of the comparison of the two person instances:

Person: equals()

```java
@Override
public boolean equals(Object aPerson) {
    if(aPerson == null) return false;
    if(!(aPerson instanceof Person)) return false;
    Person p = (Person)aPerson;
    boolean isEqual = this.name.equals(p.name);
    return isEqual;
}
```

codesnap.dev

PersonWithJob:equals()

```java
@Override
public boolean equals(Object aPerson) {
    if(aPerson == null) return false;
    if(!(aPerson instanceof PersonWithJob)) return false;
    PersonWithJob p = (PersonWithJob)aPerson;
    boolean isEqual = this.getName().equals(p.getName()) &&
            this.getSalary()==p.getSalary();
    return isEqual;
}
```

codesnap.dev

PersonWithJob

```java
public static void main(String[] args) {
    Person p1 = new PersonWithJob("Joe", 30000);
    Person p2 = new Person("Joe");
    //As PersonsWithJobs, p1 should be equal to p2
    System.out.println("p1.equals(p2)? " + p1.equals(p2));
    System.out.println("p2.equals(p1)? " + p2.equals(p1));
}
```

codesnap.dev

p1 is a **PersonWithJob** object and **p2** is a **Person** object.

- In **p1.equals(p2)**: The **equals** method of **PersonWithJob** is called. This checks if **p2** is an instance of **PersonWithJob**. Since **p2** is only an instance of **Person**, it returns **false**.
- In **p2.equals(p1)**: The **equals** method of **Person** is called. This checks if **p1** is an instance of **Person**, which it is true as a **PersonWithJob** is also a **Person**. Since the instance test is passed, this only checks the name field, which matches, so it returns **true**.

B. Solution by replacing inheritance with composition

Source:

Lab3 - code/code for prob1/lesson3/labs/prob1/composition/Person.java

Lab3 - code/code for prob1/lesson3/labs/prob1/composition/PersonWithJob.java

PersonWithJob

```java
public class PersonWithJob {

    private double salary;
    private Person person;
    public double getSalary() {
        return salary;
    }
    PersonWithJob(String n, double s) {
        salary = s;
        person = new Person(n);
    }

    @Override
    public boolean equals(Object aPerson) {
        if(aPerson == null) return false;
        if(!(aPerson instanceof PersonWithJob)) return false;
        PersonWithJob p = (PersonWithJob)aPerson;
        boolean isEqual = this.person.equals(p.person) &&
                this.getSalary()==p.getSalary();
        return isEqual;
    }
    public static void main(String[] args) {
        PersonWithJob p1 = new PersonWithJob("Joe", 30000);
        Person p2 = new Person("Joe");
        PersonWithJob p3 = new PersonWithJob("Joe", 30000);
        //Consistent comparisons
        System.out.println("p1.equals(p2)? " + p1.equals(p2)); // false
        System.out.println("p2.equals(p1)? " + p2.equals(p1)); // false
        System.out.println("p1.equals(p3)? " + p1.equals(p3)); // true
    }


}
```

codesnap.dev

Person

```java
public class Person {

    private String name;
    Person(String n) {
        name = n;
    }
    public String getName() {
        return name;
    }
    @Override
    public boolean equals(Object aPerson) {
        if(aPerson == null) return false;
        if(!(aPerson instanceof Person)) return false;
        Person p = (Person)aPerson;
        boolean isEqual = this.name.equals(p.name);
        return isEqual;
    }
    public static void main(String[] args) {

    }

}
```

codesnap.dev

2.

Source:

Lab3 - code/code for prob2/Apartment.java

Lab3 - code/code for prob2/Building.java

Lab3 - code/code for prob2/LandlordInfo.java

Lab3 - code/code for prob2/Main.java

Apartment

```java
public class Apartment {
    private double rent;
    Apartment(double rent) {
        this.rent = rent;
    }
    public double getRent() {
        return rent;
    }
}
```

codesnap.dev

LandlordInfo

```java
public class LandlordInfo {
    private List<Building> buildings;
    public LandlordInfo() {
        this.buildings = new ArrayList<>();
    }
    public void addBuilding(Building building) {
        buildings.add(building);
    }
    public double calcProfits() {
        double totalProfit = 0;
        for (Building b : buildings) {
            totalProfit += b.calcProfits();
        }
        return totalProfit; // TotalRent - TotalMaintenanceCost on Each Building
    }
}
```

codesnap.dev

Building

```java
public class Building {
    private List<Double> maintenanceCosts;
    private List<Apartment> apartments;
    public Building(double... costs){
        this.maintenanceCosts = new ArrayList<>(); // Initialize the list
        for(double cost: costs){
            this.maintenanceCosts.add(cost); // Add each cost to the list
        }
        this.apartments = new ArrayList<>();
    }
    public void addApartment(Apartment apartment){
        apartments.add(apartment);
    }
    public double getTotalMaintenanceCost(){
        double total = 0;
        for(double cost: maintenanceCosts){
            total += cost;
        }
        return total;
    }
    public double getTotalRent(){
        double total = 0;
        for(Apartment apartment: apartments){
            total += apartment.getRent();
        }
        return total;
    }
    public double calcProfits(){
        return getTotalRent() - getTotalMaintenanceCost();
    }
}
```

codesnap.dev

Main

```java
public static void main(String[] args) {
        Apartment[] apts0 = {
                new Apartment(600),
                new Apartment(700)};
        Apartment[] apts1 = {
                new Apartment(500),
                new Apartment(450)
        };
        Apartment[] apts2 = {
                new Apartment(1100),
                new Apartment(650)};

        Apartment[] apts3 = {
                new Apartment(6750),
                new Apartment(945)
        };
        Building[] bldgs = {
                new Building(150, 400),
                new Building(175, 900),
                new Building(150, 300),
                new Building(175, 775)
        };
        for(Apartment a : apts0) {...}
        for(Apartment a : apts1) {...}
        for(Apartment a : apts2) {...}
        for(Apartment a : apts3) {...}

        LandlordInfo landlord = new LandlordInfo();
        for(Building b: bldgs) {
            landlord.addBuilding(b);
        }
        System.out.println(landlord.calcProfits());
        }
```

codesnap.dev

3.

A. Does it make sense to use inheritance here? Explain.

Source:

Lab3 - code/code_for_prob3/inheritance

Cylinder

```java
public class Cylinder{
    private Double height;
    private Double radius;
    public Cylinder(double radius, double height){
        this.radius = radius;
        this.height = height;
    }
    public Double getHeight() {
        return height;
    }
    // unused-can introduce problems
    public void setHeight(double height) {
        this.height = height;
    }
    public double computeVolume() {
        return Math.PI * Math.pow(radius, 2) * height;
    }
}
```

codesnap.dev

Circle

```java
public class Circle extends Cylinder {
    public Circle(double radius) {
        super(radius, radius);
    }
    public double getRadius() {
        return super.getHeight();
    }
    public double computeArea() {
        return Math.PI * Math.pow(getRadius(), 2);
    }
}
```

codesnap.dev

Main

```java
public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle(5);
        Cylinder otherCircle = new Circle(5);
        System.out.println("Radius of circle: " + circle.getRadius());
        System.out.println("Area of circle: " + circle.computeArea());
        // unused
        circle.setHeight(7);
        System.out.println("Radius of circle: " + circle.getRadius());
        System.out.println("Area of circle: " + circle.computeArea());

        System.out.println("Volume of Other circle: " + otherCircle.computeVolume());
        System.out.println("Height of Other circle: " + otherCircle.getHeight());

    }
}
```

codesnap.dev

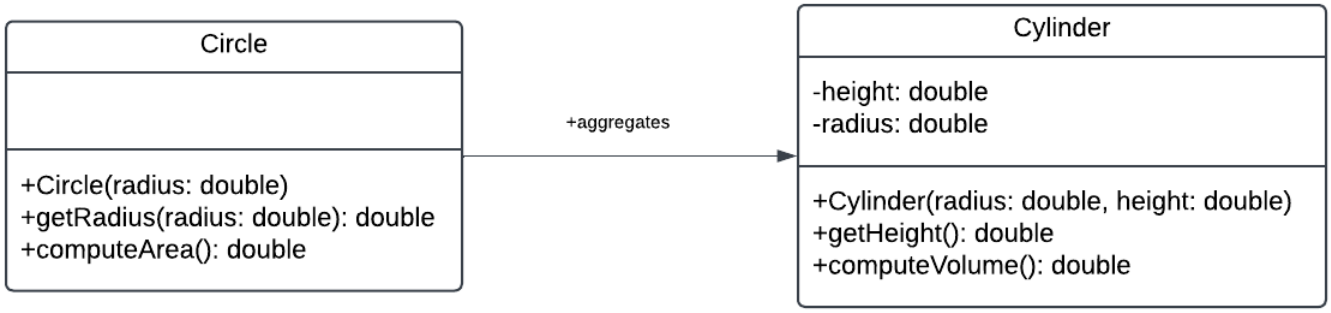
In the code implementation, it is clear to see that inheritance is not the best implementation for this relationship. It also violates both the guiding principles for using inheritance as a circle is not a cylinder and cannot be substituted for every case a cylinder object is needed. From the Main method, it is clear

that you can create a circle instance type and for polymorphism, create a circle of type Cylinder and everything works fine. But the problem arises when you add setters to the superclass, setHeight() in this scenario. Although a circle has nothing to do with the height property but when you update the height property in this case, the side value and area of the circle will change even though it should not because a circle has a side property and not a height property. To address this issue, you can use composition.

B. Design using composition and Code.

Source:

Lab3 - uml/Prob3/Composition-Circle-Cylinder.png



Source:

Lab3 - code/code_for_prob3/composition

Circle

```java
public class Circle {
    private Cylinder cylinder;
    public Circle(double radius) {
        cylinder = new Cylinder(radius, radius);
    }
    public double getRadius() {
        return cylinder.getHeight();
    }
    public double computeArea() {
        return Math.PI * Math.pow(getRadius(), 2);
    }
}
```

codesnap.dev

Cylinder

```java
public class Cylinder{
    private Double height;
    private Double radius;
    public Cylinder(double radius, double height){
        this.radius = radius;
        this.height = height;
    }
    public Double getHeight() {
        return height;
    }
    // unused
    public void setHeight(double height) {
        this.height = height;
    }
    public double computeVolume() {
        return Math.PI * Math.pow(radius, 2) * height;
    }
}
```

codesnap.dev

Main

```java
public class Main {
    public static void main(String[] args) {
        Circle circle = new Circle(5);
        Cylinder cylinder = new Cylinder(5,5);
        System.out.println("Radius of circle: " + circle.getRadius());
        System.out.println("Area of circle: " + circle.computeArea());
        // unused
//         circle.setHeight(7); //cannot be called directly and output should still be same
        System.out.println("Radius of circle: " + circle.getRadius());
        System.out.println("Area of circle: " + circle.computeArea());

        System.out.println("Volume of Cylinder: " + cylinder.computeVolume());
        System.out.println("Height of Cylinder: " + cylinder.getHeight());

    }
}
```

codesnap.dev

4.

Source:

Lab3 - code/code for prob4/

Propert y

```java
public class Property {
    private double rent;
    private Address address;

    /*
    * constructor with only rent
    * Only for this case, not to be used in a project
    * */
    public Property(double rent) {
        this(rent, new Address("123 Default St", "Sample City", "CA", 12345));
    }
    public Property(double rent, Address address) {
        this.rent = rent;
        this.address = address;
    }
    public double getRent() {
        return rent;
    }
}
```

codesnap.dev

Address

```java
public class Address {
    private String street;
    private String city;
    private String state;
    private int zipCode;

    public Address(String street, String city, String state, int zipCode) {
        this.street = street;
        this.city = city;
        this.state = state;
        this.zipCode = zipCode;
    }
}
```

codesnap.dev

House

```java
public class House extends Property{
    private double lotSize;
    public House(double lotSize) {
        super(0.1 * lotSize);
        this.lotSize = lotSize;
    }
    @Override
    public double getRent() {
        return super.getRent();
    }
}
```

codesnap.dev

Condo

```java
public class Condo extends Property {
    private int numFloors;
    Condo(int numFloors) {
        super(400 * numFloors);
        this.numFloors = numFloors;
    }
    @Override
    public double getRent() {
        return super.getRent();
    }
}
```

codesnap.dev

Trailer

```java
public class Trailer extends Property{
    public Trailer() {
        super(500);
    }
    @Override
    public double getRent() {
        return super.getRent();
    }
}
```

codesnap.dev

Admin

```java
public class Admin {
    public static double computeTotalRent(Property[] properties) {
        double totalRent = 0;
        for (Property property : properties) {
            totalRent += property.getRent();
        }
        return totalRent;
    }
}
```

codesnap.dev

Driver

```java
public class Driver {

    public static void main(String[] args) {

        Property[] properties = { new House(9000), new Condo(2), new Trailer() };
        double totalRent = Admin.computeTotalRent(properties);
        System.out.println(totalRent);
    }
}
```

codesnap.dev