1.

System.*out*.println("p1.equals(p2)? " + p1.equals(p2)); //false

=> This is because the instance p1 is of type PersonWithJob, and the equals method checks whether the other object is an instance of PersonWithJob or not.

System.out.println("p2.equals(p1)? " + p2.equals(p1)); //true
=> This is because p2 is an instance of Person. The equals method inside the Person class checks whether the instance is of type Person or not, and PersonWithJob is a subclass of Person.

Solution: Instead of using inheritance, use composition. Create a class PersonRole and add an attribute personRole to the Person class, which holds the role associated with the person. Adjust the equals method to check the role.

2

```
class LandlordInfo {
    List<Building> buildings = new ArrayList<>();

    void addBuilding(Building b) {
        buildings.add(b);
    }
    double calcProfits() {
        double profit = 0.0;
        for(Building b: buildings) {
            profit += b.calculateBuildingProfit();
        }
        return profit;
    }
}

class Apartment {
    double apartmentRent;
    Apartment(double rent) {
        apartmentRent = rent;
    }
    double getRent() {
        return apartmentRent;
    }
}
```

```
class Building {
    List<Apartment> apartments = new ArrayList<>();
    double buildingMaintenance;

    Building(double maintenance, double initialRent) {
        buildingMaintenance = maintenance;
        addApartment(new Apartment(initialRent));
    }

    void addApartment(Apartment a) {
        apartments.add(a);
    }

    double calculateBuildingProfit(){
        double totalRent = 0.0;

        for(Apartment a: apartments) {
            totalRent += a.getRent();
        }
        return totalRent - buildingMaintenance;
    }
}
```

3. Cylinder is like a 3D version of a circle. The Circle class includes common features, while the Cylinder class adds height and volume calculations. So, the relationship is like Cylinder **has a** Circle instead of Cylinder **is a** Circle. In this case composition works better instead of inheritance.

```
public class Circle {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

    public double computeArea() {
        return Math.PI * radius * radius;
    }
}

public class Cylinder {
    private Circle base;
    private double height;
```

```
    public Cylinder(double radius, double height) {
        this.base = new Circle(radius);
        this.height = height;
    }

    public double computeVolume() {
        return base.computeArea() * height;
    }
}
```

4.

```
public class Address {
    String street;
    String city;
    String state;
    String zip;

    Address() {}

    Address(String street,String city,String state, String zip) {}
}

public abstract class Property {
    abstract double computeRent();
}

public class Condo extends Property {
    int floors;
    Address address;

    Condo(int numberOfFloors) {
        floors = numberOfFloors;
        address = new Address();
    }

    Condo(int numberOfFloors, Address ad) {
        floors = numberOfFloors;
        address = ad;
    }

    @Override
    double computeRent() {
       return 400 * floors;
```

```
    }
}
```

```
public class House extends Property {
    double lotSize;
    Address address;

    House(double ltSize){
        this.lotSize = ltSize;
        address = new Address();
    }

    House(double ltSize,  Address ad){
        this.lotSize = ltSize;
        address = ad;
    }

    @Override
    double computeRent() {
        return 0.1 * lotSize;
    }
}

public class Trailer extends Property {
    Address address;
    private static final double defaultRent = 500;

    Trailer() {
    }

    Trailer(Address ad) {
       address = ad;
        address = new Address();
    }

    @Override
    double computeRent() {
        return defaultRent;
    }
}
```

```java
public class Driver {

    public static void main(String[] args) {

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

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