21022024 - Sharafat

Chapter 9 (Inheritance)

9.1

a) Inheritance

b) public and protected

c) is-a or inheritance

d) has-a or composition

e) hierarchical

f) public

g) ~~cosr~~ constructor

h) super()

9.2

a) True

b) False. A has-a relationship is implemented via composition. An is-a relationship is ~~is~~ implemented via inheritance.

c) False. This is an example of has-a relationship. Class car has an is-a relationship with class vehicle.

d) False. This is known as overriding, not overloading - an overloaded method has the same name, but a different signature.

9.3

```
public class BasePlusCommissionEmployee{
    private double baseSalary;
    private CommissionEmployee commissionEmployee;

    public BasePlusComissionEmployee(String first,
String last, String ssn, double sales,
double rate, double salary){
        commissionEmployee = new ComissionEmployee(
last, ssn, sales, rate);
    setBaseSalary(salary);
    }
    public void setFirstname(String first)
        commissionEmployee.setFirstName(first);
}
```

```
public void setLastName(String last){
    commissionEmployee.setLastName(last);
}
public void setSocialSecurityNumber(String ssn){
    commissionEmployee.setSocialSecurityNumber(ssn);
}
public void setGrossSales(double gross){
    commissionEmployee.setGrossSales(gross);
}
public void setCommission(double commission){
    commissionEmployee.setCommission(commission);
}
public void setBaseSalary(double salary){
    if (salary >= 0.0f)
        this.baseSalary = salary;
    else
        throw new IllegalArgumentException(
            "Base salary must be >= 0.0f");
}
```

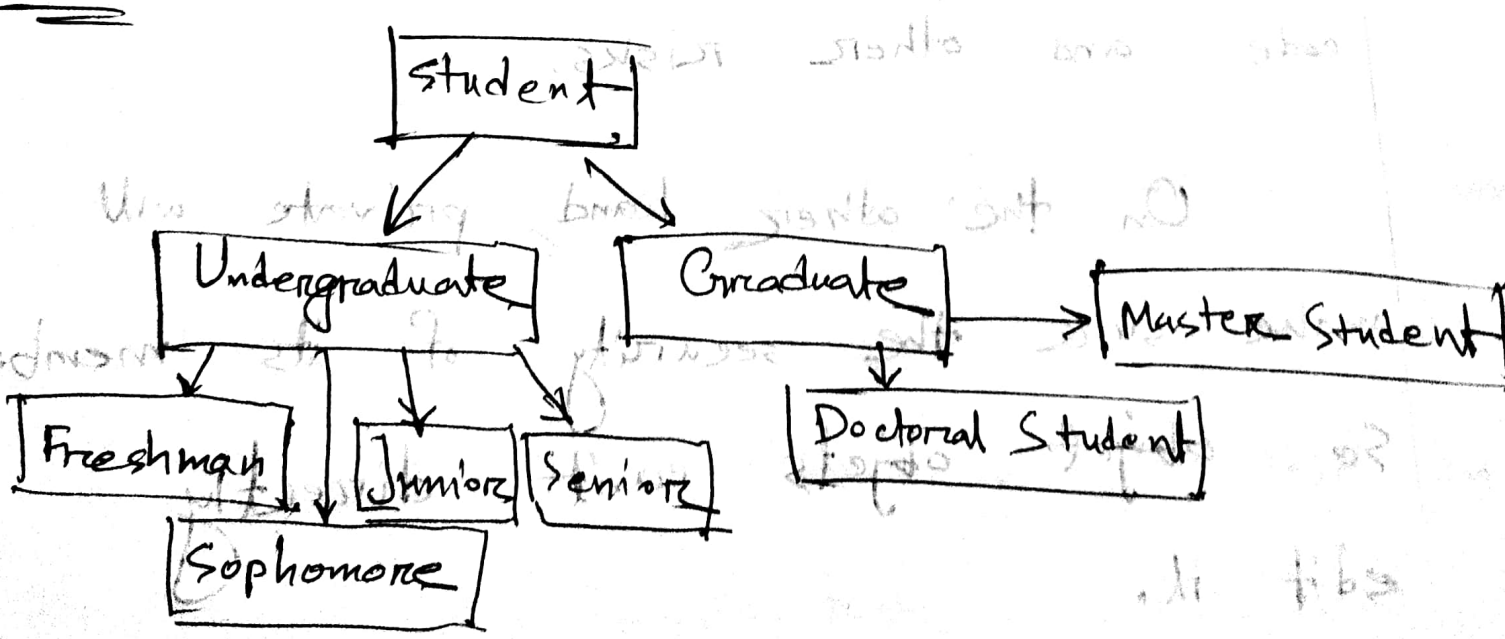
```
public String getFirstName() {
    return commissionEmployee.getFirstName();
}
public String getLastName() {
    return commissionEmployee.getLastName();
}
public String getSocialSecurityNumber() {
    return commissionEmployee.getSocialSecurityNumber();
}
public double getGrossSales() {
    return commissionEmployee.getGrossSales();
}
public double getCommissionRate() {
    return commissionEmployee.getCommissionRate();
}
public double getBaseSalary() {
    return this.baseSalary;
}
public double earnings() {
    return getBaseSalary() +
commissionEmployee.earnings();
}
```

```
public String toString(){
    return String.format(
        "%s %s\n%s: %.2f", "base salaried",
commissionEmployee.toString(); "base salary", getBaseSalary());
    }
}
```

9.4

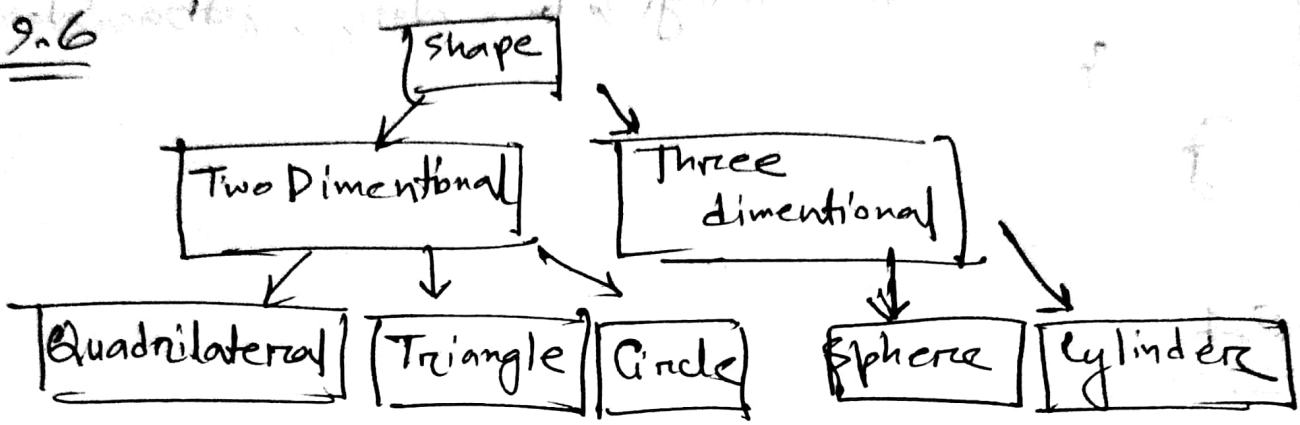
Inheritance can promote defect reduction by code reusability. In this scenario we can create a common base class. So it makes the code more easier to maintain.

9.5



Here our code will extend just like the above diagram, one and will maintain the hierarchy.

9.6

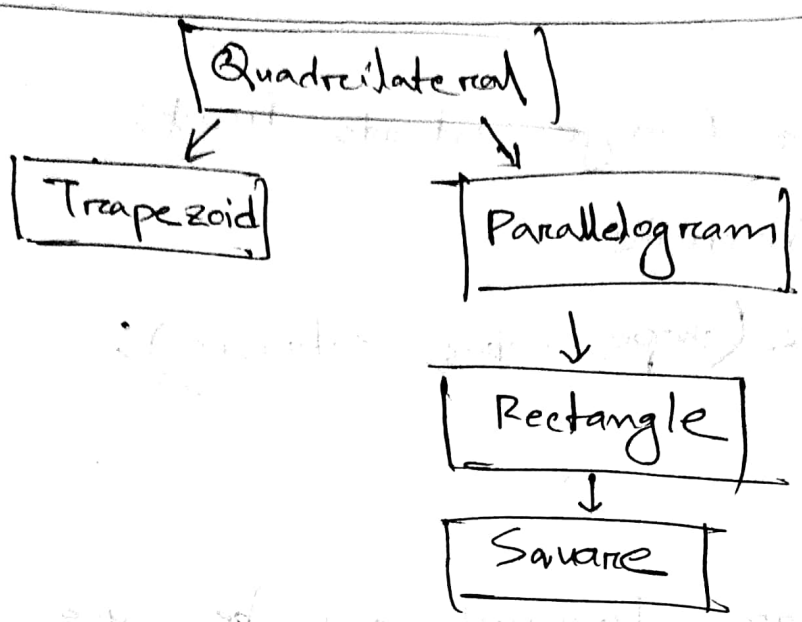


9.7

If we want to make our members of a class visible to every single objects, so that objects can modify members freely But it may cause less control over code and other risks.

On the other hand, private will make sure the security of its members So objet objets can't directly edit it.

9.8



Code is on the IDE

9.9 a) super.earnings() calls a method directly from the parent class.

b) Override will replace a ~~child~~ parent method with a ~~parent~~ child method.

c) super(firstArgument, secondArgument) will call the constructor of parent class. It must have to put before anything else in the childs constructor.

9.10

a) class Orange extends Fruit;

b) @Override

c) super (shape, color, calories);

9.11

Super keyword can be use to call the constructor of parent class. It is pretty useful here.

Super keyword is also used wh calling the method of a parent clas

9.12

To access the decode() of parce class, we can simply use super keyword. Our syntax will be like

super.decode();

2.13

Using get methods provide encapsulation and more control over members of a class. Anyone cannot just randomly access the values or modify them. It enables more flexible and bug-free way of coding. So we can reuse our code in more ways.

2.14

To create a superclass Employee, we can gather all common attributes and methods and combine them. ~~into a~~