WIX1002 Fundamentals of Programming
Tutorial 9 Inheritance

1. Write statements for each of the following
   a. Write a static method compare that return true if two objects parameter (Student s, Teacher t) are belongs to the same class. Otherwise return false.

```
public static boolean compare(Object s, Object t){
   return s.getClass() == t.getClass()
}
```

   b. Write a static method isClass that return true if the object parameter (Student s) is belong to any descendent class of Person. Otherwise return false.\

```
public static isClass(Object s){
   return s instanceOf Person;
}
```

2. Define a class Organism. The class contains the initial size of the organism and the growth rate. Create a constructor to initialize the instance variables. Then**, d**efine a class Animal. Animal is an organism that has extra instance variable which is the amount of eating need. Create a constructor to initialize the instance variable and a method to display the Animal instance variables.

```
class Organism{
   private int size;
   private double growthRate;

   public Organism(){
      this.size = 0;
      this.growthRate = 0;
   }

   public Organism(int size, double growthRate){
      this.size = size;
      this.growthRate = growthRate;
   }
}

class Animal extends Organism{
   private double amountOfEatingNeed;

   public Animal(){
      super();
      this.amountOfEatingNeed = 0;
   }
```

```
    public Animal(int size, double growthRate, double amountOfEatingNeed){
        super(size, growthRate);
        this.amountOfEatingNeed = 0;
    }

    public void display(){
        System.out.println("Size : " + size
            + "\n" + "Growth Rate : " + growthRate
            + "\n" + "Amount of Eating Need : " + amountOfEatingNeed);
    }
}
```

3. Define a class PaySystem. The class consists of the payrate per hour and the number of hours. The class also contains a method to return the total pay for a given amount of hours and a method to display the total payment. Derive a class RegularPay from PaySystem that has a constructor and did not override any base methods. Derived a class SpecialPay from PaySystem that override the base method and return the total pay plus 30% commission.

```
class PaySystem{
    private double payRatePerHour;
    private int numberOfHour;

    public PaySystem(){
        this.payRatePerHour = 0;
        this.numberOfHour = 0;
    }

    public PaySystem(double payRatePerHour, int numberOfHour){
        this.payRatePerHour = payRatePerHour;
        this.numberOfHour = numberOfHour;

    }

    public double calculateTotalPay(int numberOfHour){
        return this.payRatePerHour * numberOfHour;
    }

    public double getTotalPay(){
        return this.payRatePerHour * this.numberOfHour;
    }
}

class RegularPay extends PaySystem{
    public RegularPay(){
        super();
    }
```

```
    RegularPay(double payRatePerHour, int numberOfHour) {
        super(payRatePerHour, numberOfHour);
    }
}

class SpecialPay extends PaySystem{
    public SpecialPay(){
        super();
    }

    SpecialPay(double payRatePerHour, int numberOfHour) {
        super(payRatePerHour, numberOfHour);
    }

    @Override
    public double getTotalPay(){
        return super.getTotalPay() * 1.3;
    }
}
```

4. Define a class PurchaseSystem. The class consists of product code, unit price, quantity and total price. Besides the class consists of a method to compute the total price and a display method. Derived a class SugarPurchase from PurchaseSystem. This new class add a sugar weight attributed and override the method to compute the total price as unit price*quantity*sugar weight.

```
class PurchaseSystem{
    private String productCode;
    private double unitPrice;
    private int quantity;
    private double totalPrice;

    public PurchaseSystem(){
        this.productCode = null;
        this.unitPrice = 0;
        this.quantity = 0;
        this.totalPrice = 0;
    }

    public PurchaseSystem(String productCode, double unitPrice, int quantity){
        this.productCode = productCode;
        this.unitPrice = unitPrice;
        this.quantity = quantity;
        this.computeTotalPrice();
    }

    public void computeTotalPrice(){
        this.totalPrice = this.unitPrice * this.quantity;
    }
```

```
    public void display(){
        System.out.println("Product Code : " + productCode
                + "\n" + "Unit Price : " + unitPrice
                + "\n" + "Quantity : " + quantity
                + "\n" + "Total Price : " + totalPrice);
    }
}

class SugarPurchase extends PurchaseSystem {
    private double sugarWeight;

    public SugarPurchase(){
        super();
    }

    public SugarPurchase(String productCode, double unitPrice, int quantity, double
sugarWeight){
        super(productCode, unitPrice, quantity);
        this.sugarWeight = sugarWeight;
        this.computeTotalPrice();
    }

    @Override
    public void computeTotalPrice(){
        this.totalPrice = this.unitPrice * this.quantity;
    }
}
```