In this tutorial, you will create a system that serves espresso-based coffee. Our goal is to create a system which monitors the coffee orders made by the customers and ensure that we have enough materials before making an order. The café has fixed starting resources (espresso, milk and foam), as well as the amount of drinks they can serve due to space constraints.

As our newly hired barista has just graduated from training, our system must also provide him with the necessary ingredients for each order that is made in the system via the `showOrders()` method. Currently, the café’s system only serves espresso.

The skeleton of the program code is given below:

[Understanding UML]

a) From the program skeleton, deduce the relationships between the classes and draw a UML Class Diagram for the following classes: Café, Espresso, Latte and Cappuccino
b) The CafeSystem application is provided in a separate file CafeSystem.java. This application creates an instance of the Cafe class, which has initial amounts of espresso, foam, milk and the amount of drinks the café can accommodate. When an order is made, the system must check if the café can accommodate the order. If the order is successful, the order is stored in an array and the number of drinks ordered is updated. Complete the Espresso, Latte and Cappuccino constructor to generate an orderID beginning from 100, 200 and 300 respectively. The orderID increments by 10 for subsequent orders. Modify the Java source code below for the Latte and Cappuccino (incomplete) classes so that their ingredients can be accessed by the getMilk() and getFoam() methods and complete the showIngredients() method to print out the orderID and ingredients. Overload the order() method in the Cafe class, so that Latte and Cappuccino can also be ordered. Read more about method overloading here: http://www.techrepublic.com/article/use-method-overloading-in-java/

```java
class Espresso{
    private static final int espresso=1;
    private int orderID;
    private static int nextOrderID = 100;

    public Espresso(){
        orderID=nextOrderID;
        nextOrderID+=10;
    }
    public int getOrderId(){
        return orderID;
    }
    public void showIngredients(){
        System.out.println("OrderID: "+orderID);
        System.out.println("Espresso: "+getEspresso()+"oz");
    }
    public int getEspresso(){
        return espresso;
    }
}

class Latte{
    private static final int espresso=1;
    private static final int foam=1;
    private static final int milk=2;
    private int orderID;
    private static int nextOrderID = 200;

    public Latte(){
        orderID=nextOrderID;
        nextOrderID+=10;
    }
    public int getOrderId(){
        return orderID;
    }
    public void showIngredients(){
        System.out.println("OrderID: "+orderID);
        System.out.println("Espresso: "+getEspresso()+"oz");
        System.out.println("Foam: "+getFoam()+"oz");
        System.out.println("Milk: "+getMilk()+"oz");
    }
    public int getEspresso(){
        return espresso;
    }
    public int getFoam(){
```
class Cuppaccino{
    private static final int espresso=1;
    private static final int foam=2;
    private static final int milk=1;
    private int orderID;
    private static int nextOrderID = 300;

    public Cuppaccino(){
        orderID=nextOrderID;
        nextOrderID+=10;
    }
    public int getOrderID(){
        return orderID;
    }
    public void showIngredients(){
        System.out.println("OrderID: "+orderID);
        System.out.println("Espresso: "+getEspresso()+"oz");
        System.out.println("Foam: "+foam+"oz");
        System.out.println("Milk: "+milk+"oz");
    }
    public int getEspresso(){
        return espresso;
    }
    public int getFoam(){
        return foam;
    }
    public int getMilk(){
        return milk;
    }
}

class Cafe{
    private Espresso espressoOrders[];
    private Latte latteOrders[];
    private Cuppaccino cuppaccinoOrders[];
    private int espresso, milk, foam; // initial materials
    private int numEspressoOrdered=0;
    private int numLatteOrdered=0;
    private int numCuppaccinoOrdered=0;
    public Cafe(int espresso, int milk, int foam, int amt){
        this.espresso=espresso;
        this.milk=milk;
        this.foam=foam;
        this.espressoOrders = new Espresso[amt];
        this.latteOrders = new Latte[amt];
        this.cuppaccinoOrders = new Cuppaccino[amt];
    }
    public int getMilk(){
        return milk;
    }
    public int getFoam(){
        return foam;
    }
    public void setMilk(int milk){
        this.milk=milk;
    }
    public void setFoam(int foam){
        this.foam=foam;
    }
this.foam=foam;

public boolean order(Espresso e){
    // Is cafe orders full?
    if(numEspressoOrdered==espressoOrders.length()){
        return false;
    }else{
        // Do we have enough materials to make the espresso?
        if(espresso<e.getEspresso()){
            return false;
        }
        System.out.println("Available espresso: "+espresso+" After order: "+(espresso-e.getEspresso()));
        espresso-=e.getEspresso();
        espressoOrders[numEspressoOrdered]= e;
        numEspressoOrdered++;
        return true;
    }
}

// overload the order method here
public boolean order(Latte l){
    // Is cafe orders full?
    if(numLatteOrdered==latteOrders.length){
        return false;
    }else{
        // Do we have enough materials to make the latte?
        if(espresso<l.getEspresso() || milk<l.getMilk() || foam<l.getFoam()){ return false; }
        System.out.println("Available espresso: "+espresso+" After order: "+(espresso-l.getEspresso()));
        espresso-=l.getEspresso();
        System.out.println("Available milk: "+milk+" After order: "+(milk-l.getMilk()));
        milk-=l.getMilk();
        System.out.println("Available foam: "+foam+" After order: "+(foam-l.getFoam()));
        foam-=l.getFoam();
        latteOrders[numLatteOrdered]= l;
        numLatteOrdered++;
        return true;
    }
}

public boolean order(Cuppaccino c){
    // Is cafe orders full?
    if(numCappaccinoOrdered==cuppaccinoOrders.length){
        return false;
    }else{
        // Do we have enough materials to make the cuppacinno?
        if(espresso<c.getEspresso() || milk<c.getMilk() || foam<c.getFoam()){ return false; }
        System.out.println("Available espresso: "+espresso+" After order: "+(espresso-c.getEspresso()));
        espresso-=c.getEspresso();
        System.out.println("Available milk: "+milk+" After order: "+(milk-c.getMilk()));
        milk-=c.getMilk();
        System.out.println("Available foam: "+foam+" After order: "+(foam-c.getFoam()));
        foam-=c.getFoam();
As part of the growing coffee culture, the café has decided to allow customers to customize their espresso-based drinks based on how much milk and foam they prefer.

c) Complete the code for the **Custom** class that takes in two parameters milk and foam. Custom orderIDs start from 400. What other classes have to be modified?
CS1020: Data Structures and Algorithms I

//what other classes have to be modified?
//we need to add an array of custom orders in the Café class
//we also need to overload the Café class to serve Custom
public boolean order(Custom c){
    //Is cafe orders full?
    if(numCustomOrdered==customOrders.length){
        return false;
    }else{
        //Do we have enough materials to make the espresso?
        if(espresso<c.getEspresso() || milk<c.getMilk() ||
          foam<c.getFoam()){                    
            return false;
        }
        System.out.println("Available espresso: "+espresso+" After order: "+
        (espresso-c.getEspresso()));
        espresso=c.getEspresso();
        System.out.println("Available milk: "+milk+" After order: "+
        (milk-c.getMilk()));
        milk=c.getMilk();
        System.out.println("Available foam: "+foam+" After order: "+
        (foam-c.getFoam()));
        foam=c.getFoam();
        customOrders[numCustomOrdered]= c;
        numCustomOrdered++;
        return true;
    }
}
//we need to add an option in the CafeSystem to allow the administrator to order
a custom drink
case 4:
    System.out.println("Enter custom milk amount:");
    int milk = sc.nextInt();
    System.out.println("Enter custom foam amount:");
    int foam = sc.nextInt();
    if ( myCafe.order( new Custom(milk,foam) ) ) {
        System.out.println("Custom ordered!");
    } else {
        System.out.println("Either cafe is full or not enough ingredients
        to make order!");
    }
break;

To accommodate customers who are indecisive, the café intends to allow the administrator to search for a particular order to change the ingredients of a Custom drink.

[Understanding Object Oriented Modeling]
d) Modify the class CafeSystem to include the following static method:

int findOrderID(Custom[] customOrders, int targetOrderID);

The method takes in an array of orders and returns the index of the order with the targetOrderID. The method returns -1 if target is not found.

public static int findOrderID(Custom[] customOrders, int target){
    for(int i=0; i<customOrders.length; i++){
        if(customOrders[i]==null){
            return -1;
        }
        if(customOrders[i].getOrderID()==target){
            return i;
        }
    }
    return -1;
}
e) Modify the class CafeSystem to include an option to change a custom drink. The system should prompt the administrator for an orderID.

```java
    case 6:
      System.out.println("Which order to modify?");
      int orderID = sc.nextInt();
      int index = findOrderID(myCafe.getOrders(), orderID);
      if(index==-1){
        System.out.println("Invalid orderID!");
      }else{
        System.out.println("Enter custom milk amount:");
        int milk2 = sc.nextInt();
        System.out.println("Enter custom foam amount:");
        int foam2 = sc.nextInt();
        Custom c = myCafe.getCustomOrders()[index];
        //check if there is enough materials to make this change
        if(myCafe.getFoam()+c.getFoam()<foam2 ||
          myCafe.getMilk()+c.getMilk()<milk2){
          System.out.println("Not enough materials to make this change!");
        }else{
          myCafe.setMilk(myCafe.getMilk()+c.getMilk()-milk2);
          myCafe.setFoam(myCafe.getFoam()+c.getFoam()-foam2);
          c.setMilk(milk2);
          c.setFoam(foam2);
          System.out.println("Order modified");
        }
      }
      break;
```

The full running solution is provided in CafeSystem.java