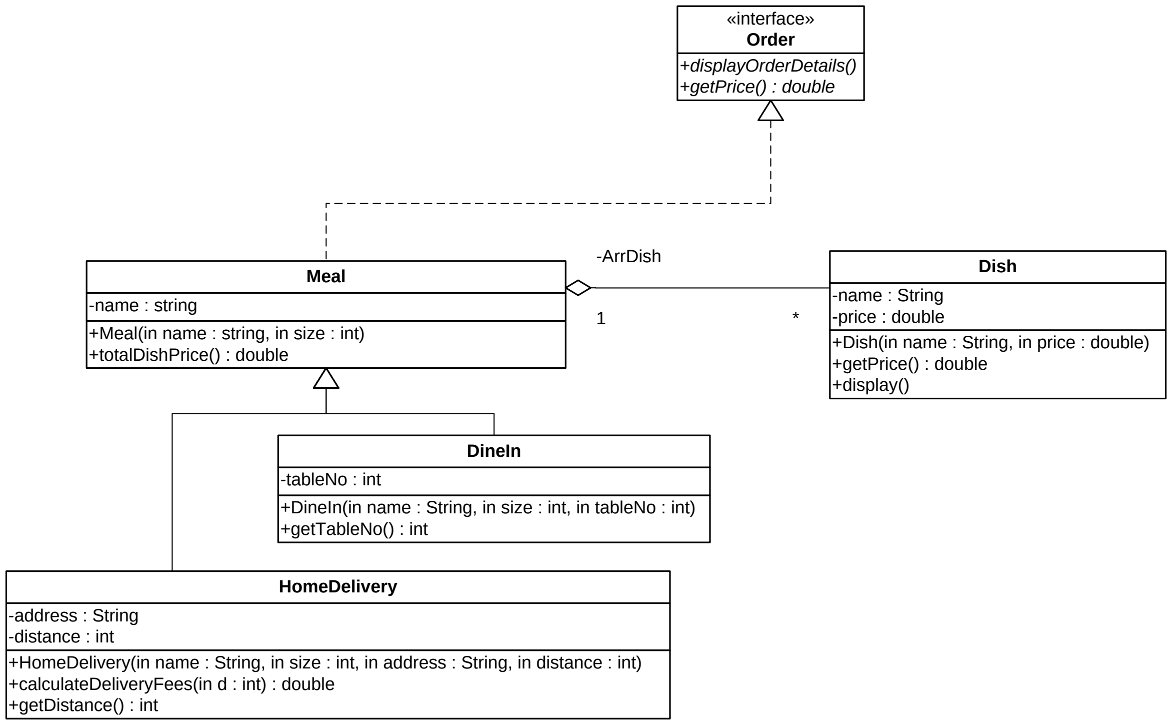
King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC113 – Computer Programming II Midterm 2 Exam – Fall 2016

Exercise 1



«interface»
Order
+displayOrderDetails()
+getPrice() : double
Meal
-name : string
+Meal(in name : string, in size : int)
+totalDishPrice() : double
-ArrDish
1
*
Dish
-name : String
-price : double
+Dish(in name : String, in price : double)
+getPrice() : double
+display()
DineIn
-tableNo : int
+DineIn(in name : String, in size : int, in tableNo : int)
+getTableNo() : int
HomeDelivery
-address : String
-distance : int
+HomeDelivery(in name : String, in size : int, in address : String, in distance : int)
+calculateDeliveryFees(in d : int) : double
+getDistance() : int

Interface ***Order:***

- Methods:
  - ***displayOrderDetails ()***: This method displays the details of the order. For ***Meal***, this method displays all the Dishes of the ***Meal***.
  - ***getPrice()***: This method returns the price of the ***order***. The price of the ***Meal*** is computed as follows:
    - For ***DineIn:*** The price of the meal = 1.05 * (the total price of all Dishes of the ***Meal***).
    - For ***HomeDelivery:*** The price of the meal = (the total price of all Dishes of the ***Meal***) + (delivery fee).

Class ***Dish:***

- Attributes:
  - ***name***: The name of the ***Dish***.
  - ***price***: The selling price of the ***Dish***.
- Methods:
  - ***Dish(name: String, price: double)***: Constructor
  - ***getPrice()***: This method returns the price of the ***Dish***. If the price is negative or greater than 100 SAR, it throws an ***Exception*** with the following message "***Wrong price***".
  - ***display()***: This method displays the name and the price of the ***Dish***.

King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC113 – Computer Programming II Midterm 2 Exam – Fall 2016

Class ***Meal***:
- Attributes:
  - ***name***: The name of the ***Meal***.
- Methods:
  - ***Meal(name: String, size: int)***: Constructor.
  - ***totalDishPrice():*** This method returns the total price of all Dishes of the ***Meal***.

Class ***DineIn***:
- Attributes:
  - ***tableNo***: The number of the table.
- Methods:
  - ***DineIn (name: String, size: int, tableNo: int)*****:** Constructor.
  - ***getTableNo()*****:** This method returns the table number of the ***DineIn***.

Class ***HomeDelivery***:
- Attributes:
  - ***address***: The address where the meal should be delivered.
  - ***distance***: The distance to the delivery address in ***Km***.
- Methods:
  - ***HomeDelivery (name:String, size: int, address:String, distance: int)*****:** Constructor.
  - ***calculateDeliveryFees(d: int)*****:** This method returns the delivery fee computed as follows:
    The delivery fee is 5 SAR when the distance is less or equal than 10 Km.
    *Otherwise the delivery fee of the distance d = 1.05 * delivery fee of the distance (d-1).*
  - ***getDistance()*****:** This method returns the distance.

QUESTION:

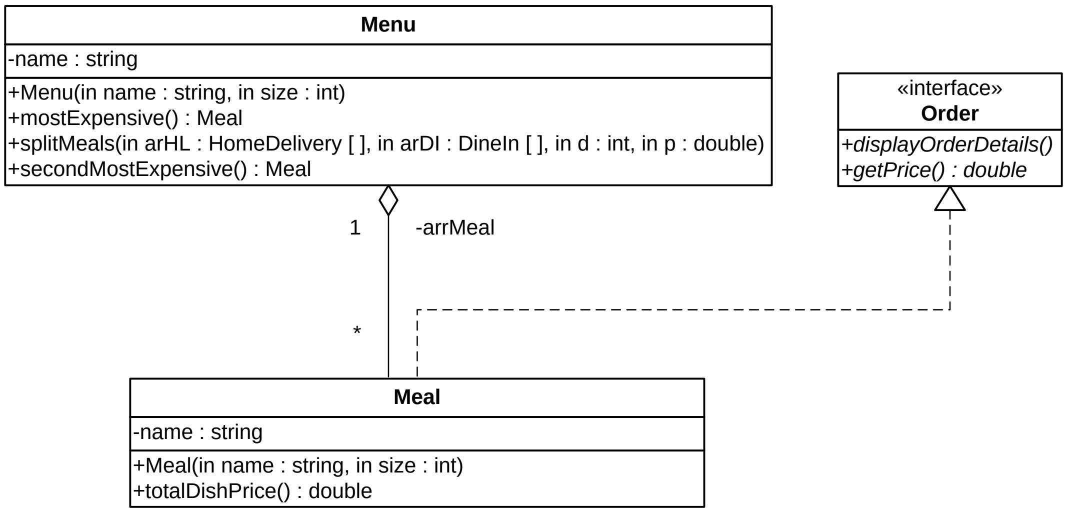
Translate into Java code the interface ***Order*** and the classes ***Meal*** and ***HomeDelivery***.

- For the method ***calculateDeliveryFees*** , give 2 solutions (an **iterative solution** and a **recursive solution**).

King Saud University
College of Computer and Information Sciences
Department of Computer Science
CSC113 – Computer Programming II Midterm 2 Exam – Fall 2016

Exercise 2:

Let's consider the same class *Meal* described in exercise 1.



Menu
-name : string
+Menu(in name : string, in size : int)
+mostExpensive() : Meal
+splitMeals(in arHL : HomeDelivery [ ], in arDI : DineIn [ ], in d : int, in p : double)
+secondMostExpensive() : Meal
«interface»
Order
+displayOrderDetails()
+getPrice() : double
1
-arrMeal
*
Meal
-name : string
+Meal(in name : string, in size : int)
+totalDishPrice() : double

Class ***Menu***:

- Attributes:
  - ***name***: The name of the ***Menu***.
- Methods:
  - ***Menu(name: String, size: int)*****:** Constructor.
  - ***MostExpensive()*****:** This method returns the most expensive ***Meal*** of the menu.
  - ***SplitMeals(arHL: HomeDelivery[], arDI: DineIn[], d: int, p: double)*****:** This method splits the array of ***Meals*** into two arrays:

    (i) ***arHL*** includes the ***HomeDelivery*** meals which the distance to the delivery address is equal to ***d***. If the array **arHL** is full, this method throws an ***Exception*** with the following message "***Number of Home Delivery exceeded!***".

    (ii) ***arDI*** includes ***DineIn*** meals which the price is greater than ***p***. If the array ***arDI*** is full, this method throws an ***Exception*** with the following message "***Number of DineIn exceeded!***".

  - ***secondMostExpensive()*****:** This method returns the second most expensive ***Meal.***

QUESTION: Translate into Java code the class ***Menu***.

```
public interface Order {    …………. 1

	public void displayOrderDetails(); …………. 0.5
	public double getPrice();   …………. 0.5

}



public class Dish {

	private String name;
	private double price;

	public Dish(String s, double p) {
		name = s;
		price = p;
	}

	public double getPrice() throws Exception {
		if (price < 0 || price > 100) throw new Exception("Wrong price");
		return price;
	}

	public void display() {
		System.out.println(name + price);
	}
}
```

```
public abstract class Meal implements Order {   …………. 1 + 1    …………. /14

      private String name;
      private Dish[] arrDish;  …………. 1
      private int nbDish;   …………. 1

      public Meal(String s, int size) { …………. /2
            name = s;
            arrDish = new Dish[size];   …………. 1
            nbDish = 0;   …………. 1
      }

      public double totalDishPrice() {   …………. /6
            double t = 0.0;  …………. 1

            for (int i=0; i < nbDish; i++) { …………. 1
                  try {  …………. 1
                        t += arrDish[i].getPrice();  …………. 1
                  }
                  catch(Exception e) { …………. 1
                        System.out.println(e.getMessage());
                  }
            }

            return t; …………. 1
      }

      public void displayOrderDetails() {  …………. /2
            for (int i=0; i < nbDish; i++)  …………. 1
                  arrDish[i].display();  …………. 1
      }
}

public class DineIn extends Meal {

      private int tableNo;

      public DineIn(String name, int size, int tn) {
            super(name, size);
            tableNo = tn;
      }

      public double getPrice() {
            return this.totalDishPrice() * 1.05;
      }
}
```

```
public class HomeDelivery extends Meal {  …………. 1           …………. /14

       private String address;
       private int distance;

       public HomeDelivery(String name, int size, String adr, int d) {  …………. /2
              super(name ,size); …………. 1
              address = adr;      …………. 0.5
              distance = d;       …………. 0.5
       }

       public double calculateDeliveryFeesRecursive(int d) {  …………. /3
              if (d <= 10) …………. 1
                     return 5.0;  …………. 1
              else
                     return 1.05 * calculateDeliveryFeesRecursive(d -1); …………. 1
       }

       public double calculateDeliveryFeesIterative(int d) {      …………. /3
              double fee = 5.0;   …………. 0.5

              for (int i = 11; i < d; i++)     …………. 1
                     fee = fee * 1.05;   …………. 1

              return fee;  …………. 0.5
       }

       public double getPrice() {       …………. /4
              double p;

              p = totalDishPrice() + calculateDeliveryFees(distance); …………. 1 + 1 + 1

              return p; …………. 1
       }

       public int getDistance() {…………. /1
              return distance;
       }
}
```

```
public class Menu {        …………. /23
      private String name;
      private Meal[] arrMeal; …………. 1
      private int nbMeal; …………. 1

      public Menu(String s, int size) {       …………. /2
            name = s;
            arrMeal = new Meal[size]; …………. 1
            nbMeal = 0;   …………. 1
      }

      public Meal mostExpensive() {    …………. /5
            Meal res = arrMeal[0];     …………. 1
            for (int i = 1; i < nbMeal; i++) {      …………. 1
                  if (arrMeal[i].getPrice() > res.getPrice())   …………. 1
                        res = arrMeal[i]; …………. 1
            }
            return res; …………. 1
      }

      public Meal secondMostExpensive() {     …………. /5
            Meal most = arrMeal[0];    …………. 0.5
            Meal second = null;        …………. 0.5

            for (int i = 1; i < nbMeal; i++) {      …………. 0.5
                  if (arrMeal[i].getPrice() > most.getPrice()) {       …………. 0.5
                        second = most;      …………. 0.5
                        most = arrMeal[i];  …………. 0.5
                  }
                  else {
                        if (   second == null || …………. 0.5
                              arrMeal[i].getPrice() > second.getPrice())   ……. 0.5

                                    second = arrMeal[i];      …………. 0.5
                  }
            }

            return second;      …………. 0.5
      }
public void split(HomeDelivery[] arHL, DineIn[] arDI, int d, double p)   …………. /9
             throws Exception {  …………. 0.5
             int j = 0, k=0;     …………. 0.5 + 0.5

             for (int i=0; i < nbMeal; i++) { …………. 0.5
                    if (arrMeal[i] instanceof HomeDelivery) { …………. 0.5
                        if (((HomeDelivery) arrMeal[i]).getDistance() == d) { 0.5+0.5
                                 if (j < arHL.length) …………. 0.5
                                  arHL[j++] = (HomeDelivery) arrMeal[i]; …0.5+0.5+0.5
                                 else
                                        throw new …………. 0.5
                                        Exception("Number of HomeDeliv. exceeded!");
                          }
                    }
                    else {
                          if (arrMeal[i].getPrice() >= p) {       …………. 0.5
                                 if (k < arDI.length)       …………. 0.5
                                   arDI[k++] = (DineIn) arrMeal[i]; …………. 0.5+0.5+0.5
                                 else
                                        throw new …………. 0.5
                                        Exception("Number of Dine-In exceeded!");

                          }
                    }
             }
      }

}
```