Question 1: OOP

1. Instruction

1. Create an IntelliJ project named "**Q1**" inside your folder CP_seatno_{ID}_{firstname} inside c:\temp
2. Copy all folders in folder "Q1" of the given zip file to your project **src** folder.
3. You are to implement or fill in the blanks for the following classes (details for each class are given in section 3.)

    1) Food (package entity)
    2) FoodCounter (package entity)
    3) Market (package entity)
    4) People (package entity)
    5) Refrigerator (package entity)
    6) GameSystem (package logic)

4. JUnit for testing is in package test.
5. Export jar file (**q1.jar**) that **includes your source code and your .class files** and put it in **root** folder of your project. YOU MUST EXPORT JAR FILE. IF YOU DO NOT, YOUR CODE WILL NOT BE MARKED.
6. You also need to create a **UML Diagram** containing all classes in the package entity. Save it as **umlQ1.png** at the **root** folder of your project.
7. You should read the whole document before start coding. You may need to use some methods from the already provided classes.

2. Problem Statement : Food Management

You are tasked to implement Food, FoodCounter, Market, People, Refrigerator and GameSystem classes of the Food management program.

The console of the application is already provided. When the application opens, the console will show all commands for the player.

```
Welcome to my shop !
Please Select number to continue
<1> Go to market
<2> List all Food in Refrigerator
<3> List people Status
<4> Eating just 1 piece of Food in Refrigerator
<5> QUIT GAME
```

If the player goes to the market, the console will show commands for the market page.

```
1
Welcome to Market
<1> List all Food in Market
<2> Buy food in market and add to Refrigerator
<3> Back to main menu
```

If the player selects to buy food. The application will ask him which one he wants to buy. If the purchase fails, the console will output like the figure below.

```
1
Welcome to Market
<1> List all Food in Market
<2> Buy food in market and add to Refrigerator
<3> Back to main menu
2
Please enter index of food that you want to buy
3
Please enter amount of food that you want to buy
6
You don't have enough money to buy this food
Add food to refrigerator failed
Welcome to Market
<1> List all Food in Market
<2> Buy food in market and add to Refrigerator
<3> Back to main menu
```

Players can eat a single food in the refrigerator. The application will show below.

```
4
=========================================================
Food in Refrigerator
=========================================================
[0]  name :Bottle of Milk , amount : 1 , tasteScore : 3
=========================================================
Please enter index of food that you want to eat
0
Eating food successfully
Please Select number to continue
<1> Go to market
<2> List all Food in Refrigerator
<3> List people Status
<4> Eating just 1 piece of Food in Refrigerator
<5> QUIT GAME
```

4.1) Package entity

You must implement this package from scratch.

4.1.1) **Class Food**: This class represents the food. You must implement this class from scratch.

Field

Name	Description
- String name	The name of the food
- int price	The price of the food
- int tasteScore	The taste score of the food refers to customers' satisfaction. This value is in range [1,5]

Constructors

Name	Description
+ Food(String name, int price, int tasteScore)	Create a new food object with specified name, price and tasteScore.

Method

Name	Description
+ void setTasteScore(int s)	Set tasteScore of the food object. taseScore of the food will be in range 1 to 5. If s is below 1, set tasteScore to 1. If s exceeds 5, set tasteScore to 5.
+ boolean equals(Food food)	the food will be the same as this if all of the followings are true: 1) Both names are the same. 2) Both prices are the same. 3) Both tasteScores are the same.
+ getter/setter for each variable	

4.1.2) **Class FoodCounter**: This class is used to store a single type of food and amount of that food.
You must implement this class from scratch.

Field

Name	Description
- Food food	The food in this FoodCounter object.
- int amount	The amount of the food. The value can't be below 1.

Constructors

Name	Description
+ FoodCounter(Food food)	Set the food as given by parameter and set amount to 1.
+ FoodCounter(Food food, int amount)	Set related fields with given parameters.

Method

Name	Description
+ getter/setter for each variable	

4.1.3) **Class Person**: This class refers to individuals of customers. You must implement this class from scratch.

Field

Name	Description
- String name	The name of the person.
- int totalMoney	The total money of the person. The value can't be below 0.
- int happinessLevel	The happiness level of the person.
- int dailyIncome	The daily income of the person. The value can't be below 0.

Constructors

Name	Description
+ Person()	Set daily income to 300. Set happiness level to -2. Set total money to 20. Set name to "John Doe"

Method

Name	Description
+ void eat(Food food)	The person will eat the food. After that, his/her happiness will be increased by the taste score of that food he/she eats.
+ void setTotalMoney(int t)	Set total money to value of t. If t value is below 0, set totalMoney of object to 0.
+ void setDailyIncome(int d)	Set daily income using value of d. If d value is below 0, set dailyIncome of object to 0.
+ getter/setter for each variable	

4.1.4) **Class Refrigerator**

You must implement this class from scratch.

Field

Name	Description
- ArrayList<FoodCounter> foods	The array of all foods stored in the refrigerator.

Constructors

Name	Description
+Refrigerator()	Create a new refrigerator. At first, there are no food inside.

Method

Name	Description
+ boolean isNewFood(Food food)	Check that food is different from every food stored in the refrigerator. If food isn't equal to any, return true. Otherwise, return false.
+ void addFood(Food food, int amount)	If food is a new food, we will add a FoodCounter with the amount assigned from the parameter to the arraylist. Otherwise, add the amount to the existing foodCounter which contains that food type.
+ boolean deleteFood(Food food)	If food doesn't exist, print "No food in refrigerator" via the console and return false. If food exists in the refrigerator, please follow these steps. 1) If that food type has only 1 amount, remove it from the existing foodcounter arraylist. Then return true. 2) If that food has more than 1 amount, reduce the amount by 1. Then return true.
+ ArrayList<FoodCounter> getFoods()	getter for foods

4.2) Package logic

4.2.1) **Class GameSystem**: This class represents the whole game system. /*Partially Provided*/

Field

Name	Description
+ static Scanner sc	the scanner object.
+ static Refrigerator refrigerator	the refrigerator object.
+ static Market market	the market object.
+ static Person player	the person object.

Method (see next page)

Name	Description
+ static void runMainMenu()	The main menu of the game. Players need to input an integer from 1 to 5 to get functions. 1) Go to market 2) List all food in refrigerator 3) List Person status 4) Eating 1 piece of food in refrigerator 5) Quit game
+ static void runMarketStatus()	This function shows the market. Player need to input an integer from 1 to 3 1) Show food in the market. 2) Buy food in the market and add it into the refrigerator. Player needs to input an integer of food index in stock and amount of that food. 3) Back to main menu
+ static void showFoodInMarket()	This function will show all food available in the market via console.
+ static boolean addFoodToRefrigerator(int index, int amount)	/* FILL CODE */ This function will let the player buy food from the market. The index means the index in an arraylist of the market's stock. If the index isn't valid, print "Error: Invalid index number!" via console and return false. Buying a food cost player by that food **price * amount** If the player doesn't have enough money, print "You don't have enough money to buy this food" in the console and return false. Otherwise, decrease player money by the cost, adding that food to the refrigerator and return true.

+ static void ShowAllFoodInRefrigerator()	This function will show all food available in the refrigerator via console. If there's no food in the refrigerator, it will show "Your refrigerator is empty. Please buy food at the market to get more food."
+ static void ShowAllPeopleStatus()	This function will show the name, money, daily income and happiness level of the player.
+ static void RunEatingFood()	This function lets the player eat food in the refrigerator. If there's no food in the refrigerator, print "Your refrigerator is empty. Please buy food at the market to get more food." Otherwise, the console will show all foods available in the refrigerator. Player needs to input the index of food he wants to eat. The food in the index which player's selected will be eaten and the console will show "Eating food successfully". Otherwise, the console will show "Eating food failed".

4.2.2) Class Main: The main class

Name	Description
+ static void main(String[] args)	Run the main menu in the game system.

4.2.3) Class Market: This class used to store all foods as a stock.

Field

Name	Description
- final ArrayList<FoodCounter> stock	used to stock all of the food.

Constructors

Name	Description
+ Market()	Create a new market. Insert 10 foodCounter objects into this stock by following. (To not numb you, the price and a taste score I will give you is for a single of food.) 1) 3 "Bottle of Milk" which has a price

	of 20 and tasteScore for 3. 2) 3 “Chocolate” which has a price of 80 and tasteScore for 2. 3) 3 “Cheese” which has a price of 50 and tasteScore for 1. 4) 3 “Bread” which has a price of 40 and tasteScore for 5. 5) 3 “Chicken” which has a price of 50 and tasteScore for 5. 6) 3 “Fish” which have a price of 60 and tasteScore for 3. 7) 3 “Prime Beef” which has a price of 1000 and tasteScore for 5. 8) 3 “Pork” which has a price of 50 and tasteScore for 3. 9) 3 “Egg” which has a price of 10 and tasteScore for 2. 10) 3 “Jasmine Rice” which has a price of 20 and tasteScore for 3.

Method

Name	Description
+ getter for the stock	

4.2.4) Class Utility:

Field

Name	Description
+ static Scanner sc	the scanner object.

Method

Name	Description
+ static void showPageBreak()	Just print a line break.
+ static int getInput(int min, int max)	This method will send the player to a hell loop. Player needs to input a number to the console between min and max to break the loop. If he can’t, he will be stuck in this loop eternally.

Score criteria

The total score of this section is 42, which will be factored to a net score of 10.

1. **FoodCounterTest** — **4 points**
   a. testConstructor (1 point)
   b. testBadConstructor (1 point)
   c. testSet (2 points)
2. **FoodTest** — **6 points**
   a. testConstructor (1 point)
   b. testBadConstructor (1 point)
   b. testSet (2 points)
   c. testEquals (2 points)
3. **GameSystemTest** — **9 points**
   a. testAddFoodToRefrigerator (5 points)
   b. testAddFoodToRefrigeratorNotEnougnMoney (2 points)
   c. testAddfoodToRefrigeratorInvalidIndex (2 points)
4. **RefrigeratorTest** — **9 points**
   a. testAddNewFood (1 point)
   b. testAddExistingFood (2 points)
   c. testDeleteExistingFood01 (2 points)
   d. testDeleteExistingFood02 (1 point)
   c. testDeleteNonExistingFood (1 point)
   d. testIsNewFood (2 points)
5. **PersonTest** — **4 points**
   a. testConstrcutor (1 point)
   b. testSetLegal (1 point)
   c. testSetOutOfRange (1 point)
   d. testEat (1 point)
6. **UML diagram** — **10 points**
   - An image file of a correct UML diagram of classes in package entity exists in the root folder of the project. Make sure you generate constructors

Export jar (**q1.jar**) file that includes source code and .class files and **put it in root folder** of your project. YOU MUST EXPORT JAR FILE. IF YOU DO NOT, YOUR CODE WILL NOT BE MARKED.