Graded FRQ A2.2

Name: Shreyash Ranjan

1. Consider a hierarchy of classes used by a power company to keep track of the buildings where they supply electricity. The hierarchy is represented by the following diagram:



\*\* Note that ApartmentBuilding is a subclass of Building.

A building is represented by the class defined below:

*public class Building {*

*public static final double rate = 3.25;*

*private String address;*

*private double wattHours; // Units of electricity used in 1 month*

*public Building( String ad ){*

*address = ad;*

*wattHours = 0;*

*}*

*// returns the amount owed by this building*

*public double amtOwed( ){ /\* not shown \*/ }*

*// Other methods not shown*

*}*

Part a:

An ApartmentBuilding is different from a regular building because instead of keeping track of the watt hours used for the whole building, it needs to keep track of the watt hours used by each of the individual apartments in the building.

Write a complete declaration of class ApartmentBuilding including the following:

* A private instance variable (an array) to be used to store the apartments’ watt hours
* A constructor with two parameters: the address of the apartment building and the number of apartments. The constructor should initialize the building’s address field as well as initializing the array to be big enough to store watt hours for each apartment in the building.
* An implementation of amtOwed that returns the amount of money owed by the entire building for the electricity used. (Calculated by multiplying the sum of the wattHours for the entire building by the rate variable).

Write the complete ApartmentBuilding class declaration.

Public class ApartmentBuilding extends Building {

Private double[] hoursWattage;

Public ApartmentBuilding(String ad, int num) {

super(ad);

hoursWattage = new double [num];

}

Public double amtOwed[]{

Double sum = 0;

For (double d:hoursWattage){

sum+=d;

}

Return sum\*rate;

}

}

Part b:

Consider the following partial declaration for the ServiceArea class. A ServiceArea represents an entire area being serviced by this power company.

*public class ServiceArea{*

*private ArrayLIst< Building > all Buildings: // a list of Buildings*

*public serviceArea( ) { /\* Constructor not shown \*/ }*

*public double totalSales( ) { /\* part b \*/ }*

*// other methods not shown*

*}*

Write the totalSales method of class ServiceArea. The method should return the total amount of money owed by all of the buildings in the allBuildings list. Assume that the buildings’ amtOwed methods work correctly regardless of what you wrote in part a.

Complete method totalSales below:

public double totalSales( ) {

Double sum = 0;

For (Building build: allBuilds) {

sum+= build.amtOwed[];

}

Return sum;

}

2.

This question concerns the two classes, Product and GroceryStore, partially defined below.

*public class Product {*

*private String name:*

*private int numInStock;*

*// Constructor not shown*

*public String getName( ){ return name: }*

*public int getNumInStock( ){ return numInStock; }*

*public void sellOne( ){ numInStock--; }*

*}*

*public class GroceryStore {*

*private Product[ ] stock;*

*// Constructor not shown*

*// Precondition: no 2 Products in the stock array have the same name*

*// Postcondition: returns the index of the Product in the stock array with the given*

*// name, or -1 if there is no such Product in the array.*

*private int findItem( String name ){*

*/\* Part a \*/*

*}*

*// Precondition: no 2 Products in the stock array have the same name*

*// Postcondition: Carries out the sale of the named product if possible and returns*

*// true or false depending on whether the sale is successful or not*

*public boolean oneSale( String name ){*

*/\* Part b \*/*

*}*

*// Precondition: no 2 Products in the stock array have the same name*

*// Postcondition: attempts to carry out a sale for each name in the orders array,*

*// creating and returning an ArrayList containing the names of all products*

*// for which the sale is not successful.*

*public ArrayList< String > allSales( String[ ] orders ){*

*/\* Part c \*/*

*}*

*}*

Part a:

Write the findItem method of the GroceryStore class. As specified by its postcondition, findItem should return the index of the Product in the stock array with the given name, or it should return -1 if there is no such Product in the array.

Complete the method findItem below:

*// Precondition: no 2 Products in the stock array have the same name*

*// Postcondition: returns the index of the Product in the stock array with the given*

*// name, or -1 if there is no such Product in the array.*

*private int findItem( String name ){*

*For (int k = 0; k<stock.length; k++){*

*If (stock[k].getName().equals(name){*

*Return k;*

*}*

*}*

*Return -1;*

*}*

Part b:

Write the oneSale method of the GroceryStore class. Method oneSale has one parameter: the name of one product (that a customer would like to buy). Method oneSale should attempt to carry out the sale of the named product, and it should return true or false depending on whether the sale is successful or not. The sale is successful if there is a product in the stock array with the given name AND if the number of items in stock of that product is greater than zero. In that case, oneSale should subtract one from the number of items in stock and return true. If there is no product in the stock array with the given name OR if the number of items in stock for that product is less than or equal to zero, oneSale should return false.

For example, assume that stock.length is four and that the elements in the array are as shown below.

| Index | 0 | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| name | “milk” | “eggs” | “butter” | “coffee” |
| numInStock | 20 | 3 | 0 | 1 |

If oneSale is called with the name “eggs”, it should subtract one from the number of eggs in stock and return true. If oneSale is called with the name “juice”, it should return false, because there is no juice in the stock array. If oneSale is called with the name “butter”, it should return false, because there are no butter items currently in stock.

In writing the method, oneSale, you may include calls to method findItem. Assume that findItem works correctly, regardless of what you wrote for part a.

Complete method oneSale below:

*// Precondition: no 2 Products in the stock array have the same name*

*// Postcondition: Carries out the sale of the named product if possible and returns*

*// true or false depending on whether the sale is successful or not*

*public boolean oneSale( String name ){*

*Int k =findItem{name};*

*if(k==-1){return false }*

*if(stock[k].getNumINStock()<=){return false ;}*

*stock[k].sellOne()*

*Return true;*

Part c:

Write the allSales method of the GroceryStore class. Method allSales has one parameter: an array of product names called orders. For each name in the orders array, allSales should attempt to carry out the sale of the named product. It should create a new ArrayList containing the names of all products for which a sale is not successful. (note: A product name should appear more than once in the new ArrayList if there is more than one failed sale of that product) Finally, it should return the new ArrayList.

For example, given the following arrays:

**stock array**

| Index | 0 | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| name | “milk” | “eggs” | “butter” | “coffee” |
| numInStock | 20 | 3 | 0 | 1 |

**orders array**

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| name | “eggs” | “milk” | “milk” | “butter” | “coffee” | “tea” | “coffee” | “milk” | “coffee” |

Method allSales should carry out five successful sales (eggs, milk, milk, coffee, milk) changing the appropriate Products in the stock array. It should create and return a new ArrayList containing the Strings:

“butter”, “tea”, “coffee”, coffee”

because the number of butter items is initially zero, there is no tea product and the number of coffee items is zero when the second and third attempts to sell one coffee item are made.

In writing the method allSales, you may include calls to the method oneSale. Assume that oneSale works correctly, regardless of what you wrote for part b.

Complete the method allSales below:

*// Precondition: no 2 Products in the stock array have the same name*

*// Postcondition: attempts to carry out a sale for each name in the orders array,*

*// creating and returning an ArrayList containing the names of all products*

*// for which the sale is not successful.*

*public ArrayList< String > allSales( String[ ] orders ){*

*ArrayList <string> failedSales = new ArrayList <String>();*

*For (string item: orders}{*

*If (!oneSale(item)){*

*failedSales.add{item);*

*}*

*}*

*Return failedSales;*

*}*