Object Oriented Programming
Lab Manual 05

Interface

Problem 01:

In this problem we will model a Shop. Let us assume that every shop performs 5 basic tasks.
● Buys items
● Sells items
● Manages Log of buying and selling
● Keeps track of items in the inventory
● Calculates balance (initial balance + profit)
Every shop sells a number of items. Each of the items has some common properties whereas there may be some additional properties for a particular item. The common properties for every shop item are,
● Item Name
● Buying price per unit
● Selling price per unit.
The shop under our consideration is retail fruit shop and it sells three items (fruits).
The details of the items can be found in the table below.

Item Name	Buying Price Per Unit	Selling Price Per Unit	Additional property
Apples	3$	5$	Color of apples can be either green or red.
Oranges	3$	6$	None
Strawberries	Canned:8$ Packed: 5$	Canned: 10$ Packed: 8$	Can be either canned or packed.

Look at the following class and interface.

```
class ShopItem {
private String name;
private double sellingPricePerUnit;
private double buyingPricePerUnit;
}
interface Shop{
//Buys 'amount' number of items
//type=1: Green Apple, type=2: Red Apple,
//type=3: Orange, type=4: Canned Strawberries, type=5: Packed Strawberries
//Update balance
//Update inventory
//Add log entry
//Perform necessary error checking
void buy(int type,int amount);
//Sell 'x' type of item.
//type=1: Green Apple, type=2: Red Apple,
//type=3: Orange, type=4: Canned Strawberries, type=5: Packed Strawberries
//If there are not enough amount in inventory print error message
//Otherwise do necessary calculations.
//Update inventory
//Update balance
void sell(int x, int amount);
// Return all the LogEntries generated so far.
LogEntry[] getLog();
//Returns an array of ShopItem that contains all
//the ShopItems that were bought but not yet sold.
//Order of the items is not important
ShopItem[] getInventory();
//return balance
double getBalance();
}
```

Implement the above scenario. Make sure you have done the following,

1. Write down three classes, Apple, Orange and Strawberries, that extends ShopItem class.

2. The Apple class must have a constructor that takes a boolean value as parameter to determine whether it is green or red. The Orange class must only have a constructor with no parameter. The Strawberries class must have a constructor with a boolean value as parameter to determine whether it is canned or packed.

3. Write down a class FruitShop that implements Shop interface. Fruit Shop constructor will take two parameters. First parameter indicates the capacity of inventory (highest number of ShopItems that the shop can store) and the second one indicates the initial balance.

4. Write down LogEntry class. Each log entry must contain at least the following,
    a. Timestamp

    b. Involved Item Name
    c. Sold or Bought
    d. Amount in units
5. You can use the following main function to test your code. But your code might be tested using custom main function.

```
public static void main(String[] args) {
FruitShop fruitShop= new FruitShop(20, 60);
System.out.println(fruitShop.getBalance());
fruitShop.buy(1, 20);
System.out.println(fruitShop.getBalance());
fruitShop.sell(1, 5);
System.out.println(fruitShop.getBalance());
fruitShop.buy(4, 5);
fruitShop.buy(4, 10);
fruitShop.sell(4, 5);
fruitShop.sell(1, 15);
fruitShop.buy(3, 10);
fruitShop.buy(4, 2);
fruitShop.buy(5, 3);
fruitShop.sell(4,1);
System.out.println(fruitShop.getBalance());
System.out.println("Generated Logs...");
System.out.println("Time
Stamp"+"\t"+"Name"+"\t"+"Amount"+"\t"+"BoughtOrSold");
for (LogEntry logEntry : fruitShop.getLog()) {
System.out.println(logEntry.toString());
}
System.out.println("Items in inventory...");
System.out.println("Name"+" " +"Buying Price"+" " +"Selling Price");
for (ShopItem shopItem : fruitShop.getInventory()) {
System.out.println(shopItem.toString());
}
}
```

```
Output for sample main
60.0
0.0
25.0
Not enough balance
Not enough space in inventory.
Not enough Amount
40.0
Generated Logs...
Time Stamp Name Amount BoughtOrSold
```

```
Mon Oct 24 21:18:08 BDT 2016 Apples 20 Bought
Mon Oct 24 21:18:08 BDT 2016 Apples 5 Sold
Mon Oct 24 21:18:08 BDT 2016 Canned Strawberries 5 Bought
Mon Oct 24 21:18:08 BDT 2016 Apples 15 Sold
Mon Oct 24 21:18:08 BDT 2016 Oranges 10 Bought
Mon Oct 24 21:18:08 BDT 2016 Canned Strawberries 2 Bought
Mon Oct 24 21:18:08 BDT 2016 Packed Strawberries 3 Bought
Mon Oct 24 21:18:08 BDT 2016 Canned Strawberries 1 Sold
Items in inventory...
Name Buying Price Selling Price
Oranges 3.0 6.0
Oranges 3.0 6.0
Oranges 3.0 6.0
Oranges 3.0 6.0
Oranges 3.0 6.0
Oranges 3.0 6.0
Oranges 3.0 6.0
Oranges 3.0 6.0
Oranges 3.0 6.0
Oranges 3.0 6.0
Packed Strawberries 8.0 10.0
Canned Strawberries 8.0 10.0
Packed Strawberries 8.0 10.0
Packed Strawberries 8.0 10.0
```

Problem 02:

1. Download InterTest.java file.
2. By copying the pattern from the other interfaces in this le, write an interface IsEmergency which extends no other interface and contains just one method soundSiren which takes no arguments and returns no value.
3. Write a class PoliceCar that implements the IsEmergency and IsLandVehicle interfaces.
4. Construct a PoliceCar object and add it to the array myArray in the main method.
5. By copying the pattern for the existing code inside the for loop, add some code that tests the array elements to see if they are instances of classes that implement the IsEmergency interface and if so, calls the soundSiren method.
   NOTE: In an expression like:
   if (myArray[i] instanceof IsLandVehicle) {
   we are testing whether or not the object referenced by myArray[i] is an instance of a class that implements the IsLandVehicle interface. Remember from the introduction that it is technically not possible to have an object being an instance of an interface. The name of the instanceof operator is a bit misleading in the case of interfaces.