Vending Machines

Introduction

It's the year 2021, and you have graduated from EWU. Your first real job is with Coca-Cola, who is trying to spearhead the vending machine market by developing specialized vending machines for university campuses. The first models will be placed in the CSE and EEE department of the varsity, and are catered specifically to Tech students. Since you have experience with the student body, you have been tasked with writing the software. However, Coke wants to make sure that their product will be an overwhelming success. For that reason, you will first write a simulator for the machine to test the functionality.

Problem Description

You will be writing classes that represent various parts of a vending machine. You will need to write two files: VendingMachine.java and VendingItem.java. A simple driver class VendingWorld.java has been provided, and it will allow you to interact with your simulation. Specific instructions for each file are given in later sections.

Solution Description

VendingItem.java

Your VendingItem class should have the following fields, methods, and constructors:

- private String name.This field is the name of an item, and should be set in the constructor.
- private double price.This field is the price of an item, and should be set in the constructor.
- VendingItem(String name, double price). This is the constructor for the class, and it takes the name and price as its parameter.
- public double getPrice(). This method is just a simple getter method for the price. price has no setter because it does not make sense for an object's price to change at runtime.
- public String toString(). This method returns a String representation of a VendingItem for use in your simulation. The String should look like: "(name): $xx.xx$" where (name) is the name of the VendingItem and xx.xx is its price.

VendingMachine.java

This class represents the vending machine itself and is the bulk of the online. It has the following fields, methods, and constructors although you can *add extra variables* if you want.

- private static double totalSales. Your boss wants a way to measure the success of this venture, so this field will keep track of the total sales across vending machines(CSE Department + EEE Department). It should be initialized to 0 and properly updated every time a sale that makes money is made. More on this later.
- private int totalItems. Your boss wants a way to measure the number of items in a vending machine, so this field will keep track of the number of items in a vending machine. It should be initialized to 10 and properly updated every time a sale that makes money is made.
- private VendingItem[] shelf. This 1D array of VendingItems represents how the items are arranged in the vending machine. shelf[i] represents the ith position where position 0 (shelf[0]) is the item at the front and each subsequent position represents the item behind it. The shelf should be initialized to have 10 positions.
- public VendingMachine(). This is the single no-arg constructor for VendingMachines. You should initialize your fields and call restock() here so that your vending machine is ready to use as soon as someone initializes it.
- public VendingItem vend(int code). This method is used to dispense an item from the vending machine. It takes in a parameter *code* which represents the position of the selected item. You must dispense the item from the proper position and move the items behind it so that there is always (if possible) an item in the 0 position.

  This method should also take care of checking the code for validity before dispensing anything and should print an error statement and return null if something goes wrong. e.g. the vending machine can hold at most 20 items at a time, so if anyone passes 23 as a parameter it should return null.

  It should also print an error statement and return null if there is no item in the position that the user selected.

  Finally, this method should update the totalSales field as needed. Be careful not to update totalSales if something went wrong
- public void restock(). This method has been implemented for you.

```
public void restock(){
   shelf[0] = new VendingItem("Lays", 1.50);
```

```
        shelf[1] = new VendingItem("Doritos", 1.50);
        shelf[2] = new VendingItem("Coke", 2.50);
        shelf[3] = new VendingItem("Rubik's Cube", 30);
        shelf[4] = new VendingItem("Pie", 3.14);
        shelf[5] = new VendingItem("Clicker", 55.55);
        shelf[6] = new VendingItem("Cheetos", 1.25);
        shelf[7] = new VendingItem("Cold Pizza", 0.99);
        shelf[8] = new VendingItem("Graphing Calculator", 120);
        shelf[9] = new VendingItem("Ramen", 3.15);
    }
```

- public static double getTotalSales(). This method is a getter for the totalSales field. It is static because totalSales is a static variable and keeps track of sales across all vending machines.
- public int getNumberOfItems(). This method should return the total number of VendingItems in the vending machine.
- public double getTotalValue(). This method should return the combined total value of all the VendingItems in the vending machine.

  NOTE: Although the method is named get...(), it is not a getter method and requires some logic.
- public String toString(). This method returns the names of the items in the shelf separated by a comma.

Running and Testing

VendingWorld.java has been provided for you. It creates several instances of VendingMachine, and allows the user to interact with them. You can run the main method to start a simulation, and test from there.

```
public class VendingWorld {

    public static void main(String[] args) {
        VendingMachine cseVendingMachine = new VendingMachine();
        VendingMachine eeeVendingMachine = new VendingMachine();

        System.out.println(cseVendingMachine);
        System.out.println(VendingMachine.getTotalSales());
        System.out.println(cseVendingMachine.getTotalItems());
        System.out.println(cseVendingMachine.getTotalValue());
        System.out.println();

        VendingItem boughtItem = cseVendingMachine.vend(0);
        System.out.println(boughtItem);
        System.out.println(cseVendingMachine);
```

```
System.out.println(VendingMachine.getTotalSales());
System.out.println(cseVendingMachine.getTotalItems());
System.out.println(cseVendingMachine.getTotalValue());
System.out.println();

boughtItem = cseVendingMachine.vend(3);
System.out.println(boughtItem);
System.out.println(cseVendingMachine);
System.out.println(VendingMachine.getTotalSales());
System.out.println(cseVendingMachine.getTotalItems());
System.out.println(cseVendingMachine.getTotalValue());
System.out.println();

boughtItem = cseVendingMachine.vend(6);
System.out.println(boughtItem);
System.out.println(cseVendingMachine);
System.out.println(VendingMachine.getTotalSales());
System.out.println(cseVendingMachine.getTotalItems());
System.out.println(cseVendingMachine.getTotalValue());
System.out.println();

//case for vend method when code position does not contain any item
boughtItem = cseVendingMachine.vend(9);
System.out.println(boughtItem);
System.out.println(cseVendingMachine);
System.out.println(VendingMachine.getTotalSales());
System.out.println(cseVendingMachine.getTotalItems());
System.out.println(cseVendingMachine.getTotalValue());
System.out.println();

//case for vend method when code position is invalid >=10
boughtItem = cseVendingMachine.vend(20);
System.out.println(boughtItem);
System.out.println(cseVendingMachine);
System.out.println(VendingMachine.getTotalSales());
System.out.println(cseVendingMachine.getTotalItems());
System.out.println(cseVendingMachine.getTotalValue());
System.out.println();

//check if getTotalSales() works correctly
boughtItem = eeeVendingMachine.vend(0);
System.out.println(boughtItem);
System.out.println(eeeVendingMachine);
System.out.println(VendingMachine.getTotalSales());
System.out.println(eeeVendingMachine.getTotalItems());
System.out.println(eeeVendingMachine.getTotalValue());
```

```
        System.out.println();
    }

}
```

Output of the tester

```
Lays, Doritos, Coke, Rubik's Cube, Pie, Clicker,
Cheetos, Cold Pizza, Graphing Calculator, Ramen,
0.0
10
219.58

Lays: 1.5
Doritos, Coke, Rubik's Cube, Pie, Clicker, Cheetos,
Cold Pizza, Graphing Calculator, Ramen,
1.5
9
218.08

Pie: 3.14
Doritos, Coke, Rubik's Cube, Clicker, Cheetos, Cold
Pizza, Graphing Calculator, Ramen,
4.640000000000001
8
214.94

Graphing Calculator: 120.0
Doritos, Coke, Rubik's Cube, Clicker, Cheetos, Cold Pizza, Ramen,
124.64
7
94.94

null
Doritos, Coke, Rubik's Cube, Clicker, Cheetos, Cold
Pizza, Ramen,
124.64
7
94.94

null
Doritos, Coke, Rubik's Cube, Clicker, Cheetos, Cold
Pizza, Ramen,
124.64
7
94.94
```

```
Lays: 1.5
Doritos, Coke, Rubik's Cube, Pie, Clicker, Cheetos,
Cold Pizza, Graphing Calculator, Ramen,
126.14
9
218.08
```