

CMSC203

Assignment #6



Bradley Beverage Shop

Bradley shop is a family owned store that sells beverages. The store offers 3 types of beverages: Coffee, Alcohol, and Smoothie. The store is open from 8 in the morning to 11 pm in the afternoon. The owner of the shop likes to automate the order transactions and reports and purchase a software for testing order activities for one month. You are asked to implement this software based on the following requirements.

Assignment Description

BevShop (The Data Manager Class)

The BevShop offers 3 types of beverages: Coffee, Alcoholic and Smoothie. Beverages can be ordered in 3 different sizes: Small, medium and large. All the beverage types have a base price. In addition there are additional charges depending on the size and specific add-ons for each type of beverage.

The BevShop has the following functionality:

- Create and process orders of different types of beverages
- Provide information on all the orders
- Total amount on a specific order
- Monthly total number of orders
- Monthly sale report

Concepts tested by this assignment

- Aggregation
- Searching an Arraylist
- Selection sort
- Enumeration
- Inheritance
- Interface
- Polymorphism
- Abstract classes
- Overriding methods
-

Interfaces

OrderInterface

This interface is provided for you and is implemented by the **Order** class.

BevShopInterface

This interface is provided for you and is implemented by the **BevShop** class.

Classes

Enumerated Type – DAY

Create an enumerated type called **DAY**. The valid values will be MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY.

Enumerated Type – SIZE

Create an enumerated type called **SIZE**. The valid values will be SMALL, MEDIUM, LARGE.

Enumerated Type – TYPE

Create an enumerated type called **TYPE**. The valid values will be COFFEE, SMOOTHIE, ALCOHOL

Data Element - Beverage

Create an abstract class called **Beverage** with:

- Instance variables for beverage name, beverage type, size, and constant attributes for the base price (\$2.0) and size price (additional \$1 to go a size up).
- A parametrized constructor to create a Beverage object given its name, type and size
- An abstract methods called **calcPrice** that calculates and returns the beverage price.
- An Overridden **toString** method: String representation for Beverage including the name and size
- An Overridden **equals** method: checks equality based on name, type, size of the beverage
- getters and setters and any other methods that are needed for your design.
- Use finals to represent constants.

Data Element – subclasses of Beverage

Create the following subclasses of Beverage for the 3 types of beverages:

Coffee

- Contains additional instance variables of type boolean to indicate if it contains extra shot of coffee (additional cost of 50 cents) and extra syrup (additional cost of 50 cents).
- A parametrized constructor
- An Overridden **toString** method: String representation of Coffee beverage, including the name, size, whether it contains extra shot, extra syrup and the price of the coffee
- An Overridden **calcPrice** method.
- An Overridden **equals** method: checks equality based on the Beverage class **equals** method and additional instance variables for this class.
- getters and setters and any other methods that are needed for your design.
- Use finals to represent constants.

Smoothie

- Contains additional instance variables for number of fruits and boolean variable to indicate if protein powder is added to the beverage. The cost of adding protein is \$1.50 and each additional fruit costs 50 cents.
- A parametrized constructor
- An Overridden **toString** method: String representation of a Smoothie drink including the name , size, whether or not protein is added , number of fruits and the price
- An Overridden **equals** method: checks equality based on the Beverage class **equals** method and additional instance variables for this class.
- An Overridden **calcPrice** method.
- getters and setters and any other methods that are needed for your design.
- Use finals to represent constants.

Alcohol

- Contains additional instance variable for weather or not it is offered in the weekend. The additional cost for drinks offered in the weekend is 60 cents.
- A parametrized constructor
- An Overridden **toString** method: String representation of a alcohol drink including the name, size, whether or not beverage is offered in the weekend and the price.
- An Overridden **equals** method: checks equality based on the Beverage class **equals** method and additional instance variables for this class.
- An Overridden **calcPrice** method.
- getters and setters and any other methods that are needed for your design.
- Use finals to represent constants.

Data Element – Customer

Create a class to represent a customer.

- Instance variables for name and age
- A parametrized constructor
- A Copy constructor
- An Overridden **toString** method: String representation for Customer including the name and age
- getters and setters and any other methods that are needed for your design.

Data Element – Order

Create a class to represent an order. This class implements two interfaces: **OrderInterface** and **Comparable**.

- Instance variables for order number, order time, order day and customer and a list of beverages within this order
- A method to generate a random number within the range of 10000 and 90000
- A parametrized constructor
- A method called **addNewBeverage** that adds a beverage to the order. This is an overloaded method to add different beverages to the order. Refer to the interface OrderInterface provided for you,
- An Overridden **toString** method: Includes order number, time, day, customer name , customer age and the list of beverages (with information of the beverage).

- Override the **compareTo** method to compare this order with another order based on the order number. Returns 0 if this order number is same as another order's order number, 1 if it is greater than another order's order number, -1 if it smaller than another order's order number.
- getters and setters and any other methods that are needed for your design. **Note: The getter method for the customer returns a Deep copy of the customer.**
- Refer to provided **OrderInterface** interface for additional methods.
-

Data Manager – BevShop

Create a class to represent a beverage shop. This class implements **BevShopInterface** provided to you.

- Instance variable for the number of Alcohol drinks ordered for the **current order**. The current order in process can have at most **3** alcoholic beverages.
- An instance list to keep track of orders
- The minimum age to order alcohol drink is 21
- time, order day and customer and a list of beverages **Order** within this order
- An Overridden **toString** method: The string representation of all the orders and the total monthly sale.
- Refer to provided **BevShopInterface** interface for additional methods.

Data Structure

- You will be using an ArrayList within your Order to hold beverages of the order and BevShop class to hold orders.

External Documentation

- Provide a UML diagram with all classes and their relationships.

Testing

There is no GUI provided for this project. To test your project you may (recommended but not required) to create your own driver file for each class as you gradually implement code.

- Ensure that BevShopNoGUITest.java produces the following output:

```
3
21
Start a new order
Total on the Order:0.0
John
23
true
Add alcohol drink
Total on the Order:2.0
true
Add second alcohol drink
Total on the Order:6.0
2
Add third alcohol drink
Total on the Order:8.0
3
Maximum alcohol drink for this order
Add a COFFEE to order
3
Total on the Order:11.0
Total on the Order:11.0
Start a new order
Total on the Order:0.0
Add a SMOOTHIE to order
Total on the Order:6.5
Add a COFFEE to order
Total on the Order:9.5
0
Age not appropriate for alcohol drink!!
Total on the Order:9.5
Maximum number of fruits

Total on the Order:16.0
Total amount for all orders:27.0
```

- Ensure that all the given JUnit tests .java succeed.

Deliverables

Deliverables / Submissions:

GitHub: Upload all input files and all implemented files into GitHub in the repo you created in Lab 1 in a directory named CMSC203_Assignment6.

Design: UML class diagram.

Implementation: Submit a compressed file containing the follow (see below): The Java application (it must compile and run correctly); a write-up as specified below. Be sure to review the provided project rubric to understand project expectations. The write-up will include:

- UML diagram – latest version
- Any assumptions that you are making for this project
- Highlights of your learning experience. What were you successful at, and what (if anything) were you not able to implement?

Deliverable format: The above deliverables will be packaged as follows. Two compressed files in the following formats:

Notice: Only submit the files that you implemented/Modified.

- LastNameFirstName_Assignment6_Complete.zip, a compressed file in the zip format, with the following:
 - Write up (Word document) - reflection paragraphs
 - UML Diagram - latest version (Word or jpg document)
 - subdirectories and other files
 - File1.html (example)
 - File2.html (example)
 - src (directory) – Java and JUnit tests files
 - File1.java (example)
 - File2.java (example)
 - Filetest.java (example)
- LastNameFirstName_Assignment6_Moss.zip, a compressed file containing one or more Java files:
 - File1.java (example)
 - File2.java (example)
 - Filetest.java (example)

This folder should contain Java source files only