CISC 124 Assignment 3 - Encapsulation

Due by onQ Submission Before 7pm Friday Feb. 28

Description

(*This assignment was inspired by problems 4.12 and 5.8 in the Savitch text.*)

In order to complete a pizza ordering system you need to write three classes: Pizza, IllegalPizza and LineItem. In brief, Pizza describes a single pizza, IllegalPizza is the exception class and LineItem contains the pizza and the number of those pizzas for a single line in the pizza order. A pizza order will be a collection of LineItem objects.

There are two possible approaches that you can take when choosing attribute types for this assignment. You can either make the toppings and size attributes all of type String, or you can make them Enum types. You cannot mix up these two approaches - you need to do one or the other. Read the section below on Enums to learn more about how they can be used in this assignment to automatically restrict arguments to legal values.

The **Pizza** object describes a fairly plain pizza:

- The attributes are:
  - size, which can be "Small", "Medium" or "Large", only.
  - vegetarian, which is true or false.
  - cheese, which can be "Single", "Double" or "Triple".
  - pineapple, which can be "None" or "Single".
  - green pepper, which can be "None" or "Single".
  - ham, which can be "none" or "Single".

  In addition to the restrictions listed above, a Pizza cannot have ham if vegetarian is true. Cheese is still OK, however.
- Your fully parameterized constructor will accept arguments for all six attributes listed above.
- Write a five parameter constructor that assumes false for vegetarian.
- A third default (or "empty") constructor will create a small, non-vegetarian pizza with just single cheese and ham.
- A small cheese pizza with single cheese only is $7.00, a medium with cheese only is $9.00 and a large with cheese only is $11.00. Each additional topping costs $1.50 each.
- Your Pizza object must be immutable.
- You only need a single accessor called "getCost" that returns the cost of the pizza.
- You will need a toString method. Here are some examples of the kind of string to be returned by this method:

```
Medium vegetarian pizza, Double cheese. Cost: $10.50 each.
Small pizza, Single cheese, pineapple, green pepper, ham. Cost: $11.50 each.
Large pizza, Triple cheese, pineapple, ham. Cost: $17.00 each.
```

- Include a proper equals method where the basis for equality is all attributes being identical.
- You won't need it because the Pizza object is immutable, but include a clone method anyways. This method will get tested in the JUnit tests.
- The Pizza object must implement the `java.io.Serializable` interface so that it can be saved to a file inside a LineItem object.

Your **IllegalPizza** object is a typical Exception object. Write just the one constructor that takes a String type message.

The **LineItem** object describes a single line item from your pizza order:

- It will contain a pointer to the Pizza object and the number of those pizzas ordered. The number of pizzas must lie between 1 and 100 inclusive.
- A second constructor defaults to a single pizza for the order.
- The number of pizzas attribute should be mutable, but the Pizza attribute should not.
- You will need accessors for both attributes as well as a third accessor that returns the total cost of the line item in the order. The line item cost is subject to a "bulk discount" - 10 to 20 pizzas, inclusive, gets a 10% discount and an order of more than 20 gets a 15% discount.

- You will need a toString that prefixes the number ordered to the result of the Pizza toString to show a complete line item. If the number of pizzas for the line item is less than ten, prefix the string with a space, so that the strings will line up when the order is displayed. Ignore tax.
- Finally, you will need a compareTo method that is based only on the total cost of the line item. To use the ArrayList sort you will need to implement the `Comparable<LineItem>` interface. When the order is listed on the screen it should list the highest total cost line items first followed by the lower total cost line items. This method should return zero if the cost difference is less than one dollar.
- The class must also implement the `java.io.Serializable` interface so that instances of LineItem can be saved to a file directly.

Using Enums

To use Enums or "Enumerated Types" for your attribute types, start by defining these Enums in your Pizza class where you declare your attributes:

```
public enum Size {Small, Medium, Large};
public enum Cheese {Single, Double, Triple};
public enum Topping {None, Single};
```

Now, you can use the Enum type as a parameter type in your methods. When an argument is supplied for an Enum parameter it must be a defined member of the Enum. The argument could still be null, but otherwise it is limited to what is defined in the Enum. So, error checking is now less complicated. Within the Pizza class you can refer to the Enums using Size, Cheese or Topping. Since they have been declared public, you can refer to the Enums outside of Pizza using Pizza.Size, Pizza.Cheese or Pizza.Topping. Fortunately, Enums are immutable once declared, so they cannot be changed as a result of this public access. Also Enum variables and members can be compared for equality directly, using ==.

If you decide to take advantage of Enums in your solution, you will need to use different testing classes and a different order system class as given below. Otherwise your classes must meet all the requirements listed above.

Testing

Here are three JUnit (version 5.1, "Jupiter") testing classes - a suite and classes for both the Pizza and LineItem objects: PizzaTest.java, LineItemTest.java and Assn3UnitTestSuite.java. Your grader will use these testing classes to help grade your submitted code. You should make sure your classes pass all tests before using them with a supplied order system class: PizzaOrderSystem.java. The order system class provides more information on how your classes should behave and provides an example of the use of an ArrayList<LineItem> collection.

You will get instructions in class on how to put the JUnit testing files in your project and run them.

Testing with Enums

The equivalent testing classes and order class to be used with an Enum version of this program are PizzaEnumsTest.java, LineItemEnumsTest.java, Assn3EnumsUnitTestSuite.java and PizzaEnumsOrderSystem.java.

Javadoc, Submission and Grading Scheme

You must write javadoc comments to describe all public portions of your three classes. You do not have to include your NetID in any of the class names.

Submit all three classes - Pizza.java, IllegalPizza.java and LineItem.java along with your IOHelper class as used by the PizzaOrderSystem class in a single archive (*.zip) file. Name the file as in "Assn3_NetID.zip", using your own NetID. Do not submit the PizzaOrderSystem class or any of the testing classes. **Do not** submit any of your *.class files! Do not submit the javadoc documentation.

The marking scheme is out of 20:

- 10 marks for how well you followed the design structure outlined above.
- 6 marks for the correct operation of the program with the PizzaOrderSystem class and for passing all unit tests.
- 4 marks for overall style and javadoc documentation.

Home

Last modified: 02/10/2020 19:43:01