Take Assessment: Exercise 5

Please answer the following question(s).
If the assessment includes multiple-choice questions, click the "Submit Answers" button when you have completed those questions.

1. Go to bottom of question.

Using Design Patterns in the Gourmet Coffee System

Prerequisites, Goals, and Outcomes

Prerequisites: Before you begin this exercise, you need mastery of the following:

- *Object-oriented Programming*
  - How to define interfaces
  - How to implement interfaces

- *Design Patterns*:
  - Knowledge of the singleton pattern
  - Knowledge of the strategy pattern

Goals: Reinforce your ability to use the singleton and strategy patterns

Outcomes: You will demonstrate mastery in the following:

- Producing applications that use the singleton pattern
- Producing applications that use the strategy pattern

Background

In this assignment, you will create another version of the *Gourmet Coffee System*. This version will present the user with four choices:

[0] Quit
[1] Display sales (Plain Text)

```
[2] Display sales (HTML)
[3] Display sales (XML)
choice>
```

The user will be able to display the sales information in three formats: plain text, HTML, or XML. Part of the work has been done for you and is provided in the student archive. You will implement the code that formats the sales information. This code will use the singleton and strategy patterns.

Description

The following class diagram shows how the singleton and strategy pattern will be used in your implementation:

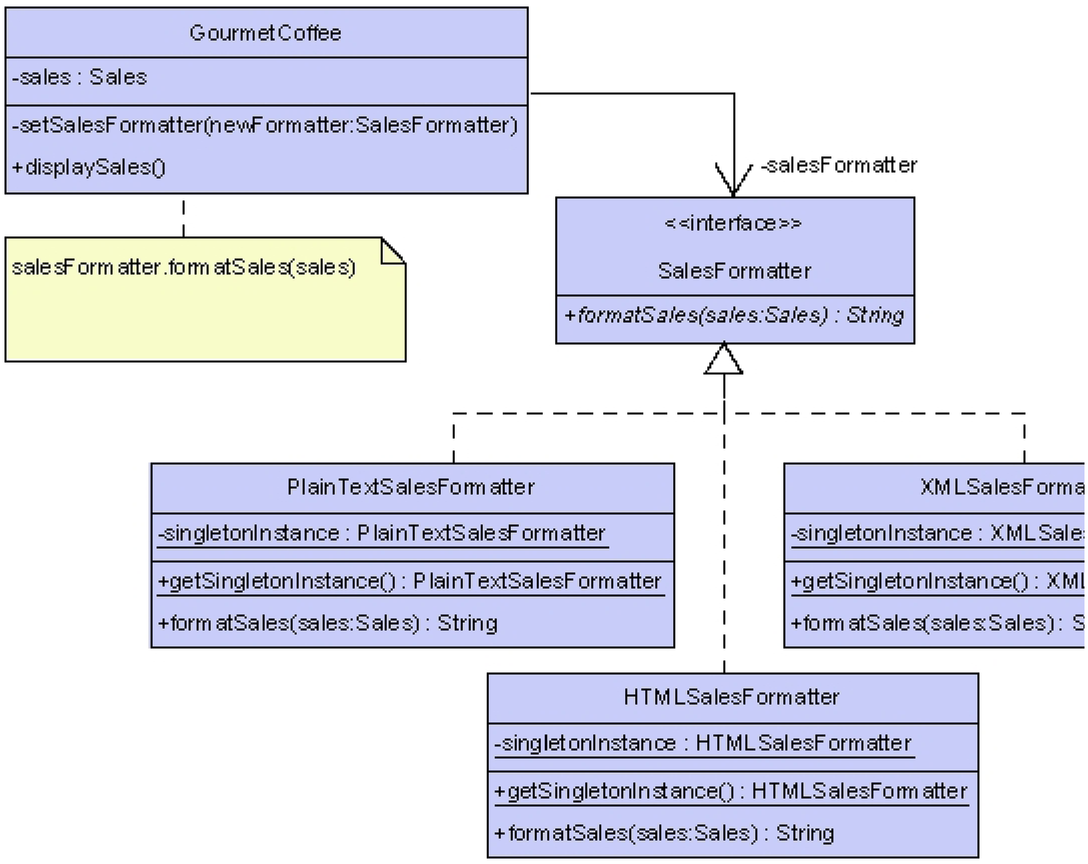
GourmetCoffee
-sales : Sales
-setSalesFormatter(newFormatter:SalesFormatter)
+displaySales()
salesFormatter.formatSales(sales)
-salesFormatter
<<interface>>
SalesFormatter
+formatSales(sales:Sales) : String
PlainTextSalesFormatter
-singletonInstance : PlainTextSalesFormatter
+getSingletonInstance() : PlainTextSalesFormatter
+formatSales(sales:Sales) : String
XMLSalesForma
-singletonInstance : XMLSale
+getSingletonInstance() : XML
+formatSales(sales:Sales) : S
HTMLSalesFormatter
-singletonInstance : HTMLSalesFormatter
+getSingletonInstance() : HTMLSalesFormatter
+formatSales(sales:Sales) : String

Figure 1 *Portion of Gourmet Coffee System class diagram*

The elements of the pattern are:

- Interface SalesFormatter declares a method called formatSales that produces a string representation of the sales information.
- Class PlainTextSalesFormatter implements formatSales. Its

version returns the sales information in a plain-text format.

- Class HTMLSalesFormatter implements formatSales. Its version returns the sales information in an HTML format.
- Class XMLSalesFormatter implements formatSales. Its version returns the sales information in an XML format.
- Class GourmetCoffee is the context class. It also contains client code. The client code calls:
    - Method GourmetCoffee.setSalesFormatter to change the current formatter
    - Method GourmetCoffee.displaySales to display the sales information using the current formatter

In this assignment, you should implement the following interface and classes:

- SalesFormatter
- PlainTextSalesFormatter
- HTMLSalesFormatter
- XMLSalesFormatter
- GourmetCoffee (a partial implementation is provided in the student archive)

Complete implementations of the following classes are provided in the student archive:

- Coffee
- CoffeeBrewer
- Product
- Catalog
- OrderItem
- Order
- Sales

Interface SalesFormatter

Interface SalesFormatter declares the method that every "Formatter" class will implement.

Method:

- *public String formatSales(Sales sales)*. Produces a string representation of the sales information.

Class PlainTextSalesFormatter

Class PlainTextSalesFormatter implements the interface SalesFormatter. This class is implemented as a singleton so a new object will not be created every time the plain-text format is used.

Static variable:

- *singletonInstance*. The single instance of class PlainTextSalesFormatter.

Constructor and methods:

- *static public PlainTextSalesFormatter getSingletonInstance()*. Static method that obtains the single instance of class PlainTextsalesFormatter.

- *private PlainTextSalesFormatter()*. Constructor that is declared private so it is inaccessible to other classes. A private constructor makes it impossible for any other class to create an instance of class PlainTextSalesFormatter.

- *public String formatSales(Sales sales)*. Produces a string that contains the specified sales information in a plain-text format. Each order in the sales information has the following format:
- ------------------------
- Order *number*
-
- *quantity1 code1 price1*
- *quantity2 code2 price2*
- ...
- *quantityN codeN priceN*
-

  Total = *totalCost*

  where

  - *number* is the order number.
  - *quantityX* is the quantity of the product.
  - *codeX* is the code of the product.
  - *priceX* is the price of the product.
  - *totalCost* is the total cost of the order.

  Each order should begin with a dashed line. The first order in the sales information should be given an order number of 1, the second should be given an order number of 2, and so on.

Class HTMLSalesFormatter

Class HTMLSalesFormatter implements the interface SalesFormatter. This class is implemented as a singleton so a new object will not be created every time the HTML format is used.

Static variable:

- *singletonInstance*. The single instance of class HTMLSalesFormatter.

Constructor and methods:

- *static public HTMLSalesFormatter getSingletonInstance()*. Static method that obtains the single instance of class HTMLSalesFormatter.

- *private HTMLSalesFormatter()*. Constructor that is declared private so it is inaccessible to other classes. A private constructor makes it impossible for any other class to create an instance of class HTMLSalesFormatter.

- *public String formatSales(Sales sales)*. Produces a string that contains the specified sales information in an HTML format.

  - The string should begin with the following HTML:

    ```
    <html>
      <body>
        <center><h2>Orders</h2></center>
    ```

  - Each order in the sales information should begin with horizontal line, that is, an <hr> tag.
  - Each order in the sales information should have the following format:

```
<hr>
<h4>Total = totalCost</h4>
  <p>
    <b>code:</b> code1<br>
    <b>quantity:</b> quantity1<br>
    <b>price:</b> price1
 </p>
     ...
 <p>
```

```
    <b>code:</b> codeN<br>
    <b>quantity:</b> quantityN<br>
    <b>price:</b> priceN
  </p>
```

where:

- *quantityX* is the quantity of the product.
- *codeX* is the code of the product.
- *priceX* is the price of the product.
- *totalCost* is the total cost of the order.

- The string should end with the following HTML:

```
  </body>
</html>
```

Class XMLSalesFormatter

Class XMLSalesFormatter implements the interface SalesFormatter. This class is implemented as a singleton so a new object will not be created every time the XML format is used.

Static variable:

- *singletonInstance*. The single instance of class XMLSalesFormatter.

Constructor and methods:

- *static public XMLSalesFormatter getSingletonInstance()*. Static method that obtains the single instance of class XMLSalesFormatter.

- *private XMLSalesFormatter()*. Constructor that is declared private so it is inaccessible to other classes. A private constructor makes it impossible for any other class to create an instance of class XMLSalesFormatter.

- *public String formatSales(Sales sales)*. Produces a string that contains the specified sales information in an XML format.

  - The string should begin with the following XML:

```
<Sales>
```

- Each order in the sales information should have the following format:

-
  ```
      <Order total="totalCost">
      <OrderItem quantity="quantity1"
  price="price1">code1</OrderItem>
      ...
      <OrderItem quantity="quantityN"
  price="priceN">codeN</OrderItem>
    </Order>
  ```

  where:

  - *quantityX* is the quantity of the product.
  - *codeX* is the code of the product.
  - *priceX* is the price of the product.
  - *totalCost* is the total cost of the order.

- The string should end with the following XML:

  ```
  </Sales>
  ```

Class GourmetCoffee

Class GourmetCoffee lets the user display the sales information in one of three formats: plain text, HTML, or XML. A partial implementation of this class is provided in the student archive.

Instance variables:

- *private Sales sales*. A list of the orders that have been paid for.

- *private SalesFormatter salesFormatter*. A reference variable that refers to the current formatter: a PlainTextSalesFormatter, HTMLSalesFormatter, or XMLSalesFormatter object.

Constructor and methods:

The following methods and constructor are complete and require no modification:

- *public static void main(String[] args) throws IOException*. Starts the application.

- *private GourmetCoffee()*. Initialize instance variables sales and salesFormatter.

- *private Catalog loadCatalog().* Populates the product catalog.
- *private void loadSales(Catalog catalog).* Populates the sales object.
- *private int getChoice() throws IOException.* Displays a menu of options and verifies the user's choice.

The following methods should be completed:

- *private void setSalesFormatter(SalesFormatter newFormatter).* Changes the current formatter by updating the instance variable salesFormatter with the object specified in the parameter newFormatter.
- *private void displaySales().* Displays the sales information in the standard output using the method salesFormatter.formatSales to obtain the sales information in the current format.
- *private void run() throws IOException.* Presents the user with a menu of options and executes the selected task
    - If the user chooses option 1, run calls method setSalesFormatter with the singleton instance of class PlainTextSalesFormatter, and calls method displaySales to display the sales information in the standard output.
    - If the user chooses option 2, run calls method setSalesFormatter with the singleton instance of class HTMLSalesFormatter, and calls method displaySales to display the sales information in the standard output.
    - If the user chooses option 3, run calls method setSalesFormatter with the singleton instance of class XMLTextSalesFormatter, and calls method displaySales to display the sales information in the standard output.

Files

The following files are needed to complete this assignment:

- *student-files.zip* — Download this file. This archive contains the following:

- Class files
    - *Coffee.class*
    - *CoffeeBrewer.class*
    - *Product.class*
    - *Catalog.class*
    - *OrderItem.class*
    - *Order.class*
    - *Sales.class*
- Documentation
    - *Coffee.html*
    - *CoffeeBrewer.html*
    - *Product.html*
    - *Catalog.html*
    - *OrderItem.html*
    - *Order.html*
    - *Sales.html*
- *GourmetCoffee.java*. A partial implementation of the class GourmetCoffee.

Tasks

Implement the interface SalesFormatter and the classes PlainTextSalesFormatter, HTMLSalesFormatter, XMLSalesFormatter. Finish the implementation of class GourmetCoffee. Document using Javadoc and follow Sun's code conventions. The following steps will guide you through this assignment. Work incrementally and test each increment. Save often.

1. **Extract** the files by issuing the following command at the command prompt:

    ```
    C:\>unzip student-files.zip
    ```

2. **Then**, implement interface SalesFormatter from scratch.

3. **Next**, implement class PlainTextSalesFormatter from scratch.

4. **Then**, implement class HTMLSalesFormatter from scratch.

5. **Next**, implement class XMLSalesFormatter from scratch.

6. **Then**, complete the method GourmetCoffee.setSalesFormatter.

7. **Next**, complete the method GourmetCoffee.displaySales.

8. **Then**, complete the method GourmetCoffee.run.

9. **Finally**, compile and execute the class GourmetCoffee. Sales information has been hard-coded in the GourmetCoffee template provided by iCarnegie.
   - If the user chooses to display the sales information in plain text, the output should be:

```
o    ------------------------
o    Order 1
o
o    5 C001 17.99
o
o    Total = 89.94999999999999
o    ------------------------
o    Order 2
o
o    2 C002 18.75
o    2 A001 9.0
o
o    Total = 55.5
o    ------------------------
o    Order 3
o
o    1 B002 200.0
o
   Total = 200.0
```

   - If the user chooses to display the sales information in HTML, the output should be:

```
o    <html>
   <body>
     <center><h2>Orders</h2></center>
     <hr>
     <h4>Total = 89.94999999999999</h4>
       <p>
         <b>code:</b> C001<br>
         <b>quantity:</b> 5<br>
         <b>price:</b> 17.99
       </p>
     <hr>
     <h4>Total = 55.5</h4>
       <p>
```

```
        <b>code:</b> C002<br>
        <b>quantity:</b> 2<br>
        <b>price:</b> 18.75
      </p>
      <p>
        <b>code:</b> A001<br>
        <b>quantity:</b> 2<br>
        <b>price:</b> 9.0
      </p>
    <hr>
    <h4>Total = 200.0</h4>
      <p>
        <b>code:</b> B002<br>
        <b>quantity:</b> 1<br>
        <b>price:</b> 200.0
      </p>
  </body>
</html>
```

- If the user chooses to display the sales information in XML, the output should be:

-
  ```
   <Sales>
  <Order total="89.94999999999999">
    <OrderItem quantity="5"
price="17.99">C001</OrderItem>
  </Order>
  <Order total="55.5">
    <OrderItem quantity="2"
price="18.75">C002</OrderItem>
    <OrderItem quantity="2" price="9.0">A001</OrderItem>
  </Order>
  <Order total="200.0">
    <OrderItem quantity="1"
price="200.0">B002</OrderItem>
  </Order>
</Sales>
  ```

Submission

Upon completion, submit **only** the following:

1. SalesFormatter.java
2. PlainTextSalesFormatter.java
3. HTMLSalesFormatter.java
4. XMLSalesFormatter.java
5. GourmetCoffee.java

Go to top of question.

File to submit: Upload File Forward File Refresh Ready for Grading

Go to top of assessment.