Project: Buyer – Part 3

Deliverables

Your project files should be submitted to Web-CAT by the due date and time specified. Note that there is also an optional Skeleton Code assignment which will indicate level of coverage your tests have achieved (there is no late penalty since the skeleton code assignment is ungraded for this project). The files you submit to skeleton code assignment may be incomplete in the sense that method bodies have at least a return statement if applicable or they may be essentially completed files. To avoid a late penalty for the project, you must submit your completed code files to Web-CAT no later than 11:59 PM on the due date for the completed code assignment. If you are unable to submit via Web-CAT, you should e-mail your project Java files in a zip file to your TA before the deadline. Test files are not required for this project. If submitted, you will be able to see your code coverage, but this will not be counted as part of your grade.

Files to submit to Web-CAT (***test files are optional***):

From Buyer – Part 1

- Buyer.java
- PreferredBuyer.java, *PreferredBuyerTest.java*
- EliteBuyer.java, *EliteBuyerTest.java*
- PremierEliteBuyer.java, *PremierEliteBuyerTest.java*

From Buyer – Part 2

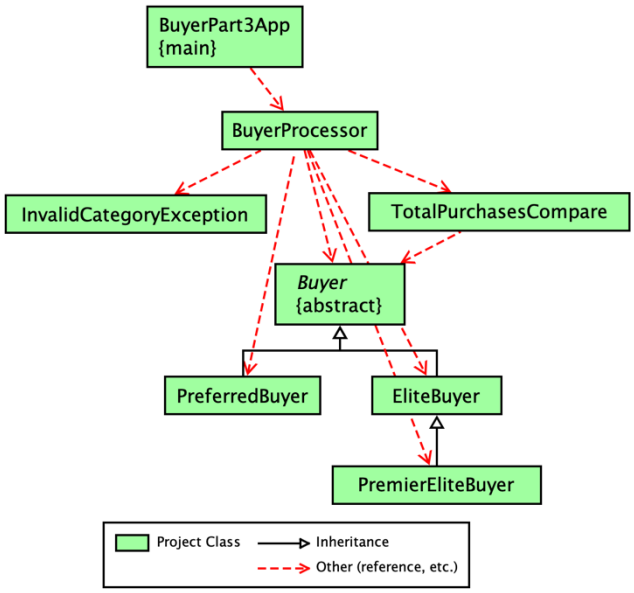
- TotalPurchasesCompare
- BuyerProcessor.java (update with exception handling), *BuyerProcessorTest.java*

New in From Buyer – Part 3

- InvalidCategoryException (new exception class)
- BuyerPart3App.java (an update with exception handling of BuyerPart2App), *BuyerPart3AppTest.java*

Specifications – Use arrays in this project; ArrayLists are not allowed!

Overview: This project is the third of three to process the monthly purchases made by members of the "Buyers" online store. Part 3 of the project focuses on handling exceptions that are thrown because of erroneous input from the command line or from the data file. The main method in the BuyerPart3App class, which reads in the file name from the command line, will need handle any FileNotFoundException that results from attempting to open the file (e.g., if the file does not exist). The readBuyerFile method in BuyerProcessor will need to handle exceptions that occur while processing the data file, including a new exception called InvalidCategoryException. The completed class hierarchy is shown in the UML class diagram at right.



Recommendations

You should create new folder for Buyer – Part 3 and copy your relevant Part 2 source, data, and optional test files to it. You should create a jGRASP project with these files in it, and then add the new source and optionally add test files as they are created if you decide to do them.

- **Buyer, PreferredBuyer, EliteBuyer, PremierEliteBuyer**

  Requirements and Design: No changes from the specifications in Buyers – Part 1.

- **TotalPurchasesCompare**

  Requirements and Design: No changes from the specifications in Buyers – Part 2.

- **InvalidCategoryException.java**

  Requirements and Design: InvalidCategoryException is a user defined exception created by extending the Exception class. This exception is to be thrown and caught in the readBuyerFile method in the BuyerProcessor class when a line of input data contains an invalid buyer category. The constructor for InvalidCategoryException takes a single String parameter representing *category* and invokes the super constructor with the following String:

  ```
  "For category: " + "\"" + category + "\""
  ```

  This string will be the toString() value of an InvalidCategoryException when it occurs. For a similar constructor, see InvalidLengthException.java in Examples\Polygons from the class notes on Exceptions.

- **BuyerProcessor.java**

  Requirements: The BuyerProcessor class provides methods for reading in the data file and generating the monthly report. The readBuyerFile method should re-designed to handle exceptions in the data. A "good" line of data results in a new element added the Buyer array, and "bad" line of data results in a record (described below) being added the to the invalid records array. A new report method produces the Invalid Records Report.

  Design: The readBuyerFile method from Part 2 should be redesigned to handle exceptions. BuyerProcessor class has fields, a constructor, and methods as outlined below.

  (1) **Fields:** no change from Part 2.
  (2) **Constructor:** no change from Part 2.
  (3) **Methods:**
  - `readBuyerFile` needs to be reworked and the generateInvalidRecordsReport method needs to be added. See Part 2 for the full description of other methods in this class. `readBuyerFile` has no return value and accepts the data file name as a String. If an exception `occurs` when attempting to open the data file, it should be ignored in this method so that it can be handled in the calling method (i.e., main). If a line from the file is processed successfully, a Buyer object of the appropriate category is added to the Buyers array field. However, when an exception occurs as a result from erroneous data in a line from the file, it

should be caught and handled as follows. The exception message and line should be concatenated as shown in the next paragraph and then the resulting String should be added to the invalid records array field in the class. The two exceptions that should be caught in this method are the InvalidCategoryException (described above) and NumberFormatException. Note that the InvalidCategoryException must be explicitly thrown by your code if the category is not 1, 2, or 3. The NumberFormatException will be thrown automatically if the item scanned in the line from the file is not a double when Double.parseDouble expects it to be a double.

The code in the while loop that reads in the buyer records and checks for buyer category should be in a try statement followed by two catch blocks: one for each of InvalidCategoryException and NumberFormatException. In each catch block, a String object should be created consisting of

```
"*** " + e + " in:\n" + line
```

where *e* is the exception and *line* is the record with the invalid data. The String object should be added to the invalidRecordsarray.

- `generateInvalidRecordsReport` processes the invalid records array to produce the Invalid Records Report and then returns the report as String. See the example result near the end of the output (i.e., the last report) for BuyerPart3App that begins on page 4 and ends on page 6.

- **BuyerPart3App.java**

  Requirements and Design: The BuyerPart3App class has only a main method as described below. In addition to the specifications in Buyer – Part 2, the main method should be modified to catch and handle an FileNotFoundException if one is thrown in the readBuyerFile method.

  - As before, the `main` method reads in the file name as the first argument, args[0], of the command line arguments, creates an instance of BuyerProcessor, and then calls the readBuyerFile method in the BuyerProcessor class to read in the data file and generate the reports as shown in the output examples beginning on the next page. The main method should not include the *throws FileNotFoundException* in the declaration. Instead, the main method should include a try-catch statement to catch FileNotFoundException when/if it is thrown in the readBuyerFile method in the BuyerProcessor class. This exception is most likely to occur when an incorrect file name is passed to the readBuyerFile method. After this exception is caught and the appropriate message is printed in the main method, your program should end. See the second example output below.

    Example data file can be downloaded from the assignment page in Canvas, and the program output for *Buyer_data_2.txt* file is shown as the third example on the next page.

Example Output when file name is missing as command line argument

```
----jGRASP exec: java BuyerPart3App
File name expected as command line argument.
Program ending.

 ----jGRASP: operation complete.
```

Example Output when attempting to read a file that is not found (fake_data.txt)

```
----jGRASP exec: java BuyerPart3App fake_data.txt
Attempted to read file: fake_data.txt (No such file or directory)
Program ending.

 ----jGRASP: operation complete.
```

Example Output for *Buyer_data_2.txt*

```
----jGRASP exec: java BuyerPart3App Buyer_data_2.txt
----------------------------
Buyer Report
----------------------------
Preferred Buyer
AcctNo/Name: 10001 Smith, Sam
Subtotal: $548.00
Tax: $43.84
Total: $591.84
Award Points: 548

Elite Buyer
AcctNo/Name: 10005 Smith, Jackie
Subtotal: $351.01
Tax: $28.08
Total: $379.09
Award Points: 3510
(includes 0.05 discount rate applied to Subtotal)

Premier Elite Buyer
AcctNo/Name: 10006 Williams, Jo
Subtotal: $1,273.29
Tax: $101.86
Total: $1,375.15
Award Points: 26460
(includes 0.15 discount rate applied to Subtotal)
(includes 1000 bonus points added to Award Points)

Preferred Buyer
AcctNo/Name: 10007 Brown, Kit
Subtotal: $598.00
Tax: $47.84
Total: $645.84
Award Points: 598
```

```
--------------------------------------
Buyer Report by Name
--------------------------------------
Preferred Buyer
AcctNo/Name: 10007 Brown, Kit
Subtotal: $598.00
Tax: $47.84
Total: $645.84
Award Points: 598

Elite Buyer
AcctNo/Name: 10005 Smith, Jackie
Subtotal: $351.01
Tax: $28.08
Total: $379.09
Award Points: 3510
(includes 0.05 discount rate applied to Subtotal)

Preferred Buyer
AcctNo/Name: 10001 Smith, Sam
Subtotal: $548.00
Tax: $43.84
Total: $591.84
Award Points: 548

Premier Elite Buyer
AcctNo/Name: 10006 Williams, Jo
Subtotal: $1,273.29
Tax: $101.86
Total: $1,375.15
Award Points: 26460
(includes 0.15 discount rate applied to Subtotal)
(includes 1000 bonus points added to Award Points)


--------------------------------------
Buyer Report by Total Purchases
--------------------------------------
Premier Elite Buyer
AcctNo/Name: 10006 Williams, Jo
Subtotal: $1,273.29
Tax: $101.86
Total: $1,375.15
Award Points: 26460
(includes 0.15 discount rate applied to Subtotal)
(includes 1000 bonus points added to Award Points)

Preferred Buyer
AcctNo/Name: 10007 Brown, Kit
Subtotal: $598.00
Tax: $47.84
Total: $645.84
Award Points: 598

Preferred Buyer
AcctNo/Name: 10001 Smith, Sam
Subtotal: $548.00
Tax: $43.84
Total: $591.84
Award Points: 548
```

```
Elite Buyer
AcctNo/Name: 10005 Smith, Jackie
Subtotal: $351.01
Tax: $28.08
Total: $379.09
Award Points: 3510
(includes 0.05 discount rate applied to Subtotal)


-----------------------
Invalid Records Report
-----------------------
*** InvalidCategoryException: For category: "0" in:
0 10002 Jones Pat 34.5 100.0 63.50 300.0

*** java.lang.NumberFormatException: For input string: "$300.0" in:
3 10003 King Kelly 34.5 100.0 63.50 $300.0

*** java.lang.NumberFormatException: For input string: "100.a" in:
3 10004 Jenkins Jordan 34.5 100.a 63.50 300.0 100.0



 ----jGRASP: operation complete.
```

Notes:

1. This project assumes that you are reading each double or int value as String using next() and then parsing it into a double with Double.parseDouble(...) or into an int with Integer.parseInt(…) as shown in the following examples.

```
... Double.parseDouble(myInput.next());
... Integer.parseInt(myInput.next());
```

   This form of input will throw a java.lang.NumberFormatException if the value is not a double.

   If you are reading in each double value as a double using nextDouble(), for example

```
... myInput.nextDouble();
```

   then a java.util.InputMismatchException will be thrown if the value read is not a double.

   For this project, you must use Double.parseDouble(...) or Integer.parseInt(…) to input a double or int value respectively since Web-CAT is expecting NumberFormatException when an input String cannot be converted to double or int value.

2. In the readBuyerFile method, for the default case (or else), which detects an invalid category, the only statement should be:

```
throw new InvalidCategoryException(category);
```

   The code that was in this block in Part 2 should be moved to the catch block for InvalidCategoryException with any changes specified for Part 3.