Project: Cardholders – 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. In order 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. The grades for the Completed Code submission will be determined by the tests that you pass or fail in your test files and by the level of coverage attained in your source files as well as usual correctness tests in Web-CAT.

Files to submit to Web-CAT:

Files from Cardholders - Part 1

- Cardholder.java
- SapphireCardholder.java, SapphireCardholderTest.java
- DiamondCardholder.java, DiamondCardholderTest.java
- BlueDiamondCardholder.java, BlueDiamondCardholderTest.java

Files in Cardholders - Part 2

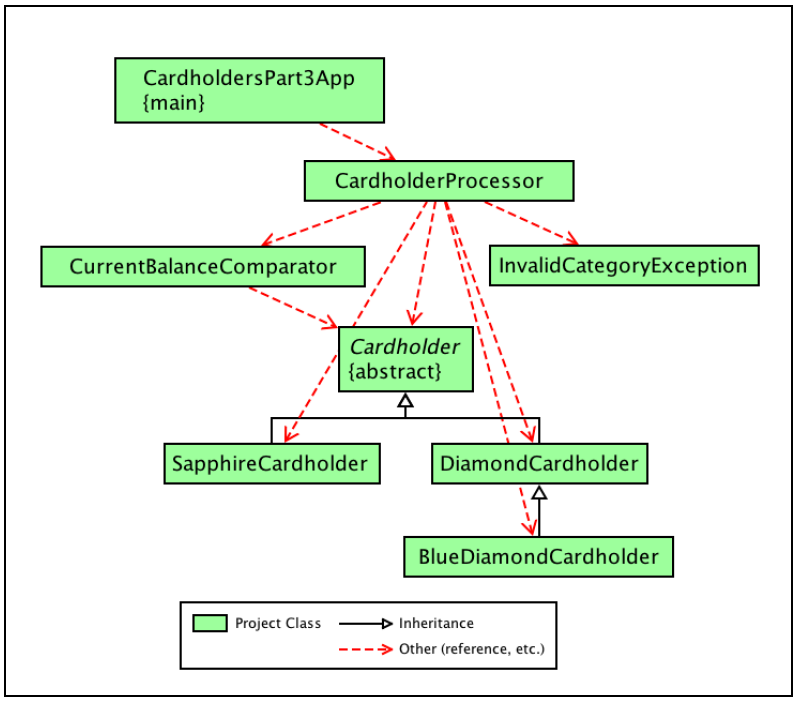
- CurrentBalanceComparator
- CardholderProcessor.java (add exception handling), CardholderProcessorTest.java

New Files in Cardholders - Part 3

- CardholdersPart3App.java java (add exception handling to Part2App), CardholdersPart3AppTest.java
- InvalidCategoryException

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

Overview: Cardholders - Part 3 is the third of a three-part software project to process the monthly purchases made by Sapphire, Diamond, and Blue Diamond cardholders. The completed class hierarchy is shown in the UML class diagram at right. Part 3 of the project focuses on handling exceptions that are thrown as a result of erroneous input from the command line or the data file. The main method in the CardholdersPart3App class, which reads in the file name from the command line, will need handle a FileNotFoundException that may result from



attempting to open the file (e.g., if the file does not exist). Also, the readCardholderFile method in CardholderProcessor will need to handle exceptions that occur while processing the data file, including a new exception called InvalidCategoryException.

Cardholder, SapphireCardholder, DiamondCardholder, BlueDiamondCardholder, CurrentBalanceComparator

Requirements and Design: There are no changes to these classes from Part 2.

CardholderProcessor.java

Requirements: The CardholderProcessor class provides methods for reading in the data file and generating the monthly report. The readCardholderFile method should redesigned to handle exceptions in the data. A "good" line of data results in a new element added the Cardholder array, and "bad" line of data results in the line + exception message being added the to the invalid records array. A new report method produces the Invalid Records Report.

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

- **Fields:** no change from Part 2.
- **Constructor:** no change from Part 2.
- **Methods:** The readCardholderFile 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.
  - `readCardholderFile` has no return value, accepts the data file name as a String, and throws FileNotFoundException. If a FileNotFoundException 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 Cardholder object of the appropriate category (subclass) is added to the Cardholder array field. However, when an exception occurs as a result from erroneous data in a line read from the file, it should be caught and handled as follows. The exception message should be concatenated to the line 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 below) and the NumberFormatException. Note that the InvalidCategoryException must be explicitly thrown by your code if the category is not 1, 2, or 3; whereas, 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.
  - `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 for CardholdersPart3App that begins on page 4 and ends on page 6.

Code and Test: See examples of exception handling in the text and the class notes. Download `cardholder_data_3_exceptions.txt` from the assignment page in Canvas to test your program. Your JUnit test methods should force the exceptions described above to thrown and caught.

Since the readCardholderFile method will propagate the FileNotFoundException if the file is not found when the Scanner is created to read the file, your test method could catch this exception to check that it was thrown.

InvalidCategoryException.java

Requirements and Design: The InvalidCategoryException class defines a new subclass of the Exception class. The constructor invokes the super constructor with the message: `"*** invalid category ***"`. See examples of creating user defined in from text and class notes.

CardholdersPart3App.java

Requirements: The CardholdersPart3App class contains the main method for running the program. In addition to the specifications in Part 2, the main method should be modified as indicated below.

Design: The CardholdersPart3App class is the driver class and has a main method described below.

- `main` accepts a file name as a command line argument, then within a try block, creates a CardholderProcessor object, and then invokes its methods to read the file and process the reward records and then to generate and print the four reports as shown in the example output beginning on page 4. If no command line argument is provided, the program should indicate this and end as shown in the first example output on page 4. If an FileNotFoundException is thrown in the readCardholderFile method in the CardholderProcessor class, it should be caught in the catch block of the try statement. The catch block should print a message ("*** Attempted to read file: " along with the exception's message). For example, if the user entered "nofile.txt" as the command line argument and this file does not exit, then the Run I/O in jGRASP would look like the following (this is also repeated in the example output beginning on page 4). Note that since the main method is catching FileNotFoundException, it no longer needs the throws clause in its declaration.

```
----jGRASP exec: java CardholdersPart3App nofile.txt
*** Attempted to read file: nofile.txt (No such file or directory)

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

Code and Test: See examples of exception handling in the text and the class notes. Download `cardholder_data_3_exceptions.txt` from the assignment page in Canvas to test your program. One of your JUnit test methods should call your main method with an argument this is not a valid file name to ensure that your catch block is covered. See "Code and Test" for CardholdersPart2App in Part 2 to see how to invoke your main method.

Example Output

Three separate runs are shown below: (1) one with no command line argument, (2) one with an invalid file name as command line argument, and (3) one with valid file name as command line argument.

```
 ----jGRASP exec: java CardholdersPart3App
File name expected as command line argument.
Program ending.

 ----jGRASP: operation complete.

 ----jGRASP exec: java CardholdersPart3App nofile.txt
 *** Attempted to read file: nofile.txt (No such file or directory)

 ----jGRASP: operation complete.

 ----jGRASP exec: java CardholdersPart3App cardholder_data_3_exceptions.txt
----------------------------
Monthly Cardholder Report
----------------------------
Diamond Cardholder
AcctNo/Name: 10002 Jones, Pat
Previous Balance: $1,200.00
Payment: ($100.00)
Interest: $11.00
New Purchases: $473.10
Current Balance: $1,584.10
Minimum Payment: $47.52
Purchase Points: 1,419
(includes 5.0% discount rate applied to New Purchases)

Blue Diamond Cardholder
AcctNo/Name: 10003 King, Kelly
Previous Balance: $1,300.00
Payment: ($150.00)
Interest: $11.50
New Purchases: $538.20
Current Balance: $1,699.70
Minimum Payment: $50.99
Purchase Points: 2,690
(includes 10.0% discount rate applied to New Purchases)

Sapphire Cardholder
AcctNo/Name: 10001 Smith, Sam
Previous Balance: $1,100.00
Payment: ($200.00)
Interest: $9.00
New Purchases: $548.00
Current Balance: $1,457.00
Minimum Payment: $43.71
Purchase Points: 548

Blue Diamond Cardholder
AcctNo/Name: 10004 Jenkins, Jordan
Previous Balance: $1,400.00
Payment: ($1,400.00)
Interest: $0.00
```

```
New Purchases: $9,000.00
Current Balance: $9,000.00
Minimum Payment: $270.00
Purchase Points: 47,500
(includes 10.0% discount rate applied to New Purchases)
(includes 2,500 bonus points added to Purchase Points)

--------------------------------------
Monthly Cardholder Report (by Name)
--------------------------------------
Blue Diamond Cardholder
AcctNo/Name: 10004 Jenkins, Jordan
Previous Balance: $1,400.00
Payment: ($1,400.00)
Interest: $0.00
New Purchases: $9,000.00
Current Balance: $9,000.00
Minimum Payment: $270.00
Purchase Points: 47,500
(includes 10.0% discount rate applied to New Purchases)
(includes 2,500 bonus points added to Purchase Points)

Diamond Cardholder
AcctNo/Name: 10002 Jones, Pat
Previous Balance: $1,200.00
Payment: ($100.00)
Interest: $11.00
New Purchases: $473.10
Current Balance: $1,584.10
Minimum Payment: $47.52
Purchase Points: 1,419
(includes 5.0% discount rate applied to New Purchases)

Blue Diamond Cardholder
AcctNo/Name: 10003 King, Kelly
Previous Balance: $1,300.00
Payment: ($150.00)
Interest: $11.50
New Purchases: $538.20
Current Balance: $1,699.70
Minimum Payment: $50.99
Purchase Points: 2,690
(includes 10.0% discount rate applied to New Purchases)

Sapphire Cardholder
AcctNo/Name: 10001 Smith, Sam
Previous Balance: $1,100.00
Payment: ($200.00)
Interest: $9.00
New Purchases: $548.00
Current Balance: $1,457.00
Minimum Payment: $43.71
Purchase Points: 548

--------------------------------------
Monthly Cardholder Report (by Current Balance)
--------------------------------------
Blue Diamond Cardholder
AcctNo/Name: 10004 Jenkins, Jordan
Previous Balance: $1,400.00
```

```
Payment: ($1,400.00)
Interest: $0.00
New Purchases: $9,000.00
Current Balance: $9,000.00
Minimum Payment: $270.00
Purchase Points: 47,500
(includes 10.0% discount rate applied to New Purchases)
(includes 2,500 bonus points added to Purchase Points)

Blue Diamond Cardholder
AcctNo/Name: 10003 King, Kelly
Previous Balance: $1,300.00
Payment: ($150.00)
Interest: $11.50
New Purchases: $538.20
Current Balance: $1,699.70
Minimum Payment: $50.99
Purchase Points: 2,690
(includes 10.0% discount rate applied to New Purchases)

Diamond Cardholder
AcctNo/Name: 10002 Jones, Pat
Previous Balance: $1,200.00
Payment: ($100.00)
Interest: $11.00
New Purchases: $473.10
Current Balance: $1,584.10
Minimum Payment: $47.52
Purchase Points: 1,419
(includes 5.0% discount rate applied to New Purchases)

Sapphire Cardholder
AcctNo/Name: 10001 Smith, Sam
Previous Balance: $1,100.00
Payment: ($200.00)
Interest: $9.00
New Purchases: $548.00
Current Balance: $1,457.00
Minimum Payment: $43.71
Purchase Points: 548

-----------------------
Invalid Records Report
-----------------------
1;10005;Smith, Pam;110p.0;200.0;34.5;100.0;63.50;350.0 *** invalid numeric value ***

4;00000;Williams,Pat;1000.0;0.0;34.5;100.0;63.50;300.0 *** invalid category ***

3;10008;Jenkins, Jordan;1400.0;1400.0;5000.0;1000.0+;4000.0 *** invalid numeric value ***

2;10006;Jones, Mat;1200.0;1-0.0;34.5;100.0;63.50;300.0 *** invalid numeric value ***

3;10007;King, Kilby;1300.0;150.0;34.5;$100.0;63.50;300.0;100.0 *** invalid numeric value ***


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

Notes

1. This project assumes that you are reading each double value as a String using next() and then parsing it into a double with Double.parseDouble(...) as shown in the following example.

   . . . Double.parseDouble(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 in is not a double.

   For this assignment, you should change your input to use Double.parseDouble(...) rather than nextDouble(), since Web-CAT is looking for NumberFormatException rather than java.util.InputMismatchException.

2. If you are using the JUnit Assert.assertArrayEquals method to check two Cardholder arrays for equality, then the `equals` and `hashCode` methods must be implemented in your Cardholder class; that is, Assert.assertArrayEquals calls equals(Object obj) on each object in the array, so Cardholder must have an equals method that overrides the one inherited from the Object class. If Cardholder does not override equals(Object obj), then the JUnit Assert.assertArrayEquals method will use the inherited equals(Object obj) method which means two Cardholder arrays will be equal only if they are aliases.

   Below is a simplified `equals` method and `hashCode` method you are free to use.

```
public boolean equals(Object obj) {
   if (!(obj instanceof Cardholder)) {
      return false;
   }
   else {
      Cardholder c = (Cardholder) obj;
      return (name.equalsIgnoreCase(c.getName()));
   }
}

public int hashCode() {
   return 0;
}
```