Project: Cardholders – Part 2

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:

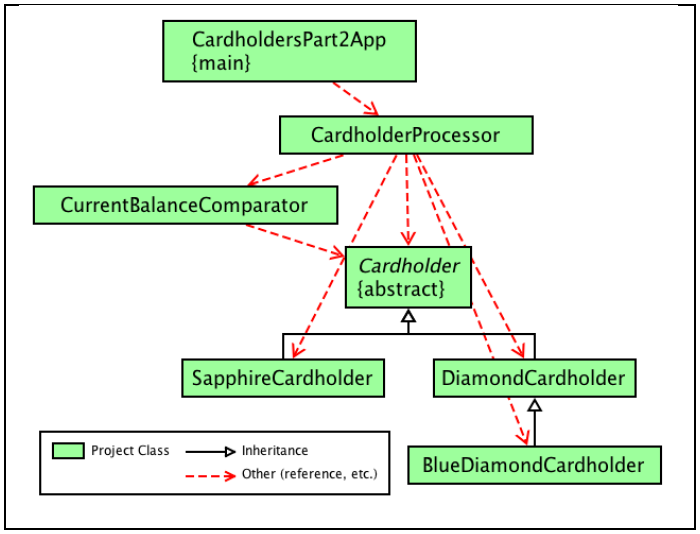
Files from Cardholders - Part 1

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

New Files in Cardholders - Part 2

- CurrentBalanceComparator
- CardholderProcessor.java, CardholderProcessorTest.java
- CardholdersPart2App.java, CardholdersPart2AppTest.java

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



Overview: Cardholders - Part 2 is the second of a three-part software project to process the monthly purchases made by Sapphire, Diamond, and Blue Diamond. In Part 2, you will modify the Cardholder class and you will create three additional classes and their associated test files:

(1) CardholderProcessor which includes a method to read in a cardholder data file and methods to generate several reports;

(2) CurrentBalanceComparator which implements the Comparator interface for Cardholder;

(3) CardholdersPart2App which includes the main method for the program. You should create new folder for Part 2 and copy your Part 1 source and test files to it. You should create a jGRASP project and add the new class and test files as they are created. As you develop and debug your program, you may find it helpful to use the "viewer canvas" feature in conjunction with the debugger or interactions.

- **Cardholder, SapphireCardholder, DiamondCardholder, BlueDiamondCardholder**

  Requirements and Design: In addition to the specifications in Part 1, the Cardholder class should implement the Comparable interface for type Cardholder. In the `compareTo` method, the comparison is based on the name field, which is a String. [Hint: In your `compareTo` method, you should consider invoking the `compareToIgnoreCase` method on name since it's a String.] Otherwise, there are no changes to the classes from Part 1.

- **CardholderProcessor.java**

  Requirements: The CardholderProcessor class provides methods for reading in the data file and generating the monthly reports.

  Design: The CardholderProcessor class has fields, a constructor, and methods as outlined below.

  (1) **Fields:** 1) An array of Cardholder objects and 2) an array of String elements to hold invalid records read from the data file. [The second array will be used in Part 3.] Note that there are no fields for the number elements in each array. In this project, the size of the array should be the same as the number of elements in the array. These two fields should be private.
  (2) **Constructor:** The constructor has no parameters and initializes the Cardholder array and String array in the fields to arrays of length 0.
  (3) **Methods:** Usually a class provides methods to access and modify each of its instance variables (i.e., getters and setters) along with any other required methods. The methods for CardholderProcessor are described below.
    - `getCardholdersArray` returns an array of type Cardholder representing the Cardholder array field.
    - `getInvalidRecordsArray` returns an array of type String representing the invalid records array field.
    - `addCardholder` has no return value, accepts a Cardholder object, increases the capacity of the Cardholder array by one, and adds the Cardholder in the last position of the Cardholder array.
    - `addInvalidRecord` has no return value, accepts a String, increases the capacity of the invalidRecords array by one, and adds the String in the last position of the invalidRecords array. This method will be used in Project 11, but it still needs to be tested in this project.
    - `readCardholderFile` has no return value, accepts the data file name as a String, and throws FileNotFoundException. This method creates a Scanner object to read in the file one line at a time. When a line is read, a separate Scanner object on the line should be created to read the values in that line. The data in each line is separated by a semicolon so the delimiter should be set to semicolon by invoking the `useDelimiter(";")` method on the Scanner object for the line. For each line read in, the appropriate Cardholder object is created and added to the Cardholder array field. The data file has semicolon-delimited text records as follows: category, account number, name, previous balance, and payment, followed by one or more purchases. To read in purchases, consider using a while loop which reads a purchase, converts it to a double, then call the `addPurchases` method for the Cardholder object.

Remember, SapphireCardholder, DiamondCardholder, and BlueDiamondCardholder objects are all Cardholder objects. The category codes are 1 for Sapphire Cardholder, 2 for Diamond Cardholder, and 3 for Blue Diamond Cardholder. Below are examples data records:

```
1;10001;Smith, Sam;1200.0;200.0;34.5;100.0;63.50;350.0
2;10002;Jones, Pat;1200.0;0.0;34.5;100.0;63.50;300.0
3;10003;King, Kelly;1200.0;0.0;34.5;100.0;63.50;300.0;100.0
3;10004;Jenkins, Jordan;1200.0;0.0;5000.0;1000.0;4000.0
```

- `generateReport` processes the Cardholder array using the original order from the file to produce the Monthly Rewards Club Report and then returns the report as String. See example result in output for CardholdersPart2App beginning on page 5.
- `generateReportByName` sorts the Cardholder array by its natural ordering, and processes the Cardholder array to produce the Monthly Cardholder Report (by Name), then returns the report as a String. See example result in output for CardholdersPart2App beginning on page 5.
- `generateReportByCurrentBalance` sorts the Cardholder array by current balance, and processes the Cardholder array to produce the Monthly Cardholder Report (by Current Balance) and then returns the report as String. See example result in output for CardholdersPart2App beginning on page 5.

Code and Test: See examples of file reading and sorting (using Arrays.sort) in the class notes. The natural sorting order for Cardholder objects is determined by the compareTo method when the Comparable interface is implemented.

```
Arrays.sort(cardholders);
```

The sorting order based on a cardholder's current balance is determined by the CurrentBalanceComparator class which implements the Comparator interface (described below).

```
Arrays.sort(cardholders, new CurrentBalanceComparator());
```

In your JUnit test methods for the generate reports methods above, you may want to use the following assertion to avoid having to match the return result exactly (where actual_result is the result of the method call and expected_result is part of what you think it should contain.

```
Assert.assertTrue(actual_result.contains(expected_result));
```

- **CurrentBalanceComparator.java**

Requirements and Design: The CurrentBalanceComparator class implements the Comparator interface for Cardholder objects. Hence, it implements the method `compare(Cardholder c1, Cardholder c2)` that defines the ordering from highest to lowest based on the cardholder's current balance. See examples in class notes.

- **CardholdersPart2App.java**

Requirements: The CardholdersPart2App class contains the main method for running the program.

Design: The CardholdersPart2App class is the driver class and has a main method described below.

- `main` accepts a file name as a command line argument, creates a CardholderProcessor object, and then invokes its methods to read the file and process the cardholder records and then to generate and print the three reports as shown in the example output beginning on page 5. If no command line argument is provided, the program should indicate this and end as shown in the first example output on page 5. An example data file can be downloaded from the assignment page in Canvas.

Code and Test: In your JUnit test file for the CardholdersPart2App class, you should have at least two test methods for the main method. One test method should invoke CardholdersPart2App.main(args) where args is an empty String array, and the other test method should invoke CardholdersPart2App.main(args) where args[0] is the String representing the data file name. Depending on how you implemented the main method, these two methods should cover the code in main. As for the assertion in the test method, since INTEREST_RATE is a public class variable in Cardholder, you could assert that `Cardholder.INTEREST_RATE` equals `0.01` in each test methods.

In the first test method, you can invoke main with no command line argument as follows:

```
// If you are checking for args.length == 0
// in CardholdersPart2App, the following should exercise
// the code for true.
String[] args1 = {};  // an empty String[]
CardholdersPart2App.main(args1);
```

In the second test method, you can invoke main as follows with the file name as the first (and only) command line argument:

```
String[] args2 = {"cardholder_data_1.txt"};
// args2[0] is the file name
CardholdersPart2App.main(args2);
```

If Web-CAT complains the default constructor for `CardholdersPart2App` has not been covered, you should include the following line of code in one of your test methods.

```
// to exercise the default constructor
CardholdersPart2App app = new CardholdersPart2App();
```

Example Output

Running the `CardholdersPart2App` without a file name as a command line argument:

```
 ----jGRASP exec: java CardholdersPart2App
File name expected as command line argument.
Program ending.

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

Running the `CardholdersPart2App` with the file name `cardholder_data_1.txt` as a command line argument:

```
 ----jGRASP exec: java CardholdersPart2App cardholder_data_1.txt
----------------------------
Monthly Cardholder Report
----------------------------
Sapphire Cardholder
AcctNo/Name: 10001 Smith, Sam
Previous Balance: $1,200.00
Payment: ($200.00)
Interest: $10.00
New Purchases: $548.00
Current Balance: $1,558.00
Minimum Payment: $46.74
Purchase Points: 548

Diamond Cardholder
AcctNo/Name: 10002 Jones, Pat
Previous Balance: $1,200.00
Payment: ($0.00)
Interest: $12.00
New Purchases: $473.10
Current Balance: $1,685.10
Minimum Payment: $50.55
Purchase Points: 1,419
(includes 5.0% discount rate applied to New Purchases)

Blue Diamond Cardholder
AcctNo/Name: 10003 King, Kelly
Previous Balance: $1,200.00
Payment: ($0.00)
Interest: $12.00
New Purchases: $538.20
Current Balance: $1,750.20
Minimum Payment: $52.51
Purchase Points: 2,690
(includes 10.0% discount rate applied to New Purchases)

Blue Diamond Cardholder
AcctNo/Name: 10004 Jenkins, Jordan
Previous Balance: $1,200.00
Payment: ($0.00)
Interest: $12.00
New Purchases: $9,000.00
Current Balance: $10,212.00
Minimum Payment: $306.36
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: $1,200.00
Payment: ($0.00)
Interest: $12.00
New Purchases: $9,000.00
Current Balance: $10,212.00
Minimum Payment: $306.36
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: ($0.00)
Interest: $12.00
New Purchases: $473.10
Current Balance: $1,685.10
Minimum Payment: $50.55
Purchase Points: 1,419
(includes 5.0% discount rate applied to New Purchases)

Blue Diamond Cardholder
AcctNo/Name: 10003 King, Kelly
Previous Balance: $1,200.00
Payment: ($0.00)
Interest: $12.00
New Purchases: $538.20
Current Balance: $1,750.20
Minimum Payment: $52.51
Purchase Points: 2,690
(includes 10.0% discount rate applied to New Purchases)

Sapphire Cardholder
AcctNo/Name: 10001 Smith, Sam
Previous Balance: $1,200.00
Payment: ($200.00)
Interest: $10.00
New Purchases: $548.00
Current Balance: $1,558.00
Minimum Payment: $46.74
Purchase Points: 548

--------------------------------------
Monthly Cardholder Report (by Current Balance)
--------------------------------------
Blue Diamond Cardholder
AcctNo/Name: 10004 Jenkins, Jordan
Previous Balance: $1,200.00
Payment: ($0.00)
Interest: $12.00
New Purchases: $9,000.00
Current Balance: $10,212.00
Minimum Payment: $306.36
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: $1,200.00
Payment: ($0.00)
Interest: $12.00
New Purchases: $538.20
Current Balance: $1,750.20
Minimum Payment: $52.51
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: ($0.00)
Interest: $12.00
New Purchases: $473.10
Current Balance: $1,685.10
Minimum Payment: $50.55
Purchase Points: 1,419
(includes 5.0% discount rate applied to New Purchases)

Sapphire Cardholder
AcctNo/Name: 10001 Smith, Sam
Previous Balance: $1,200.00
Payment: ($200.00)
Interest: $10.00
New Purchases: $548.00
Current Balance: $1,558.00
Minimum Payment: $46.74
Purchase Points: 548


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