CSCI 470/680 Assignment 1

Write a console-based Java application to help an airline frequent flyer find good ways of redeeming his/her accumulated mileage into airline tickets. Assume all trips depart from ORD. Different destinations need different mileage for redeeming tickets. If the person doesn't travel in a busy season, they may take advantage of the “supersaver” program that can get a ticket with fewer accumulated miles.

Input

A list of destination cities and related ticket redemption information should be read from a file, which is formatted with one city per line, with five fields separated by ";" as shown below:

destination; *normal mileage* needed for an economy class ticket (this information is proportional to the actual travel distance); *supersaver mileage* needed for an economy class ticket; *additional mileage* needed for upgrading to first class; and the *months of departure* when the supersaver mileage can be used.

For example:

```
Hyderabad;30000;20000;15000;2-4
Sidney;50000;30000;20000;5-6
Paris;20000;15000;10000;2-4
New York;10000;5000;8000;10-11
```

The name of the file (such as "`miles.txt`") should be passed to the program as a command line argument.

Miles Redemption Algorithm

1. Try to get tickets that travel the farthest.
2. Use supersaver whenever possible.
3. Try to get as many as different tickets as possible (at most 1 ticket is needed for 1 destination)
4. Use the remaining mileage for upgrade, if possible (try to upgrade the longest trip first).

Output

Your program should first print out a list of the destination cities for the user. It will then prompt the user for input and print out a list of tickets that can be redeemed.

Here is some sample output. Values entered by the user are shown in blue.

```
-------------------------------------------
List of destination cities you can travel to:

Hyderabad
New York
Paris
Sidney
-------------------------------------------
```

```
Please input your total accumulated miles: 49600

Please input your month of departure (1-12): 4

Your accumulated miles can be used to redeem the following tickets:

* A trip to Hyderabad, economy class
* A trip to Paris, economy class
* A trip to New York, economy class

Your remaining miles: 4600

Do you want to continue (y/n)? n
```

Another test case: With a total of 69000 miles and a planned departure in July, you can get an economy class ticket to Sydney and a first class ticket to New York, with 1000 miles remaining.

Recommended Classes

Write a class called `Destination`. This class will encapsulate information about a destination such as the name of the destination city, the normal miles required for a ticket, the additional miles for upgrading, supersaver miles, the start month of the supersaver program, and the end month of the supersaver program. Private instance variables and public accessor methods should be written for this information. A constructor that takes all of this information as arguments is also needed.

Also write a class `MilesRedeemer` to encapsulate the logic for redeeming mileage. This class should have private instance variables for an `ArrayList` of `Destination` objects, and an integer to represent the remaining miles after the user's miles have been redeemed. Initialize the remaining miles to 0.

You should also create methods in `MilesRedeemer` such as:

```
public void readDestinations(Scanner fileScanner)
public String[] getCityNames()
public ArrayList<String> redeemMiles(int miles, int month)
public int getRemainingMiles()
```

For the first method, we use a `Scanner` object as the input parameter for flexibility and reusability. For example, we could reuse the method to read files from a web URL. This method should use the `Scanner` object to read and parse the destination data into an `ArrayList` of `Destination` objects, which should then be sorted by their normal mileage (see below).

The second method should loop through the `ArrayList` of `Destination` objects and create an array of `String` objects from the city names. This array can be sorted in ascending order and returned (to be printed out by the main program).

For `redeemMiles()`, `miles` is the total available miles, and `month` is the desired month of departure. To avoid writing one huge method, you can (and probably should) have the `redeemMiles()` method call some other methods to accomplish subtasks as part of the larger overall algorithm. This method should return an `ArrayList` of `String` objects containing

descriptions of redeemed tickets to be printed out by the main program. It should also save the miles remaining after the tickets have been redeemed.

The last method should return the saved remaining miles.

You should also write a main class, e.g. `MilageRedemptionApp`, that will have the `main()` method with the loop for user interaction. The `main()` method will get the command line argument, use the filename supplied to create a `Scanner` object, create an instance of `MilesRedeemer`, tell it to read in the destination data, get the city names from `MilesRedeemer` and print them out, read the data input by the user using another `Scanner`, call `redeemMiles()` to get the list of tickets that may be redeemed, and `getRemainingMiles()` to get the miles remaining. All of the normal I/O and interaction with the user should be done in methods of your main class, not in the `MilesRedeemer` class.

Remember that only one class per source code module can be public. So if all classes for this project are placed in one source code file, only the `MilageRedemptionApp` class that contains the `main()` method should be public. If each class is in its own separate source code file, each class should be designated as public.

A hint: if an users answers `'y'` or `'Y'`, an additional method call to `scanner.nextLine()` will be needed before the program proceeds to the next iteration. This is for the scanner to pass the current line.

File IO and String Parsing

The `Scanner` class can be used to read the lines from the file. As lines are read from the input file, the lines need to be parsed (broken into fields) using `";"` and `"-"` as the delimiters. Instances of `Destination` will be created and added to the list. There are a variety of different ways to parse a `String` into fields. One easy way is to use the `split()` method of the `String` class, which allows you to supply the desired delimiters as a regular expression.

Sorting

The cities in the input text file might be listed in any order. The `redeemMiles()` method may need to sort the destinations in descending order based on their distances.

The class `java.util.ArrayList` has a `sort()` method that takes a `Comparator`, an object of a class that defines the ordering using a method named `compare()`. For example:

```
class MileageComparator implements Comparator<Destination> {

   public int compare(Destination d1, Destination d2) {
      return (d2.getNormalMiles() - d1.getNormalMiles());
   }
}
```

`Comparator` is a *functional interface*, so you may supply a lambda expression as the argument to the `sort()` method instead.

The array of city name strings created and returned by `getCityNames()` can also be sorted using a version of `Arrays.sort()`. There's no need to write a comparator in this case; the “natural ordering” of the strings used by default will be sufficient.

Input Verification and Exception Handling

The `main()` method should verify the number of command line arguments before taking in the file name.

If the program encounters problems in opening the file or input type errors during reading/parsing the file, the program should print out an error message, then exit.

If an input type mismatch is thrown while reading console input, the program should print out an error message and then proceed directly to the question `"Do you want to continue (y/n)?"`.

How to Submit the Assignment

Submit the `.java` files for the assignment through Blackboard. You may upload them individually or as a zip file if you prefer.

If you work in a group, only one submission is needed for the entire group. The same student should submit the assignments for the group throughout the semester.