Computer Science 60-212 - Fall 2017

Assignment 4

Due: End of Friday, December 8, 2017

Readings

Chapters 1 to 12

Problem 1. (20 points)

Write a Java class `ComboLock` that works like the combination lock in a gym locker. The lock is constructed with a combination, three numbers between `0` and `39`. The `reset` method resets the dial so that it points to 0. The `turnLeft` and `turnRight` methods turn by a given number of ticks to the left or right. The `open` method attempts to open the lock. The lock opens if the user first turned it right to the first number in the combination, then left to the second, and then right to the third. The class and its methods' signatures are as follows:

```
public class ComboLock {
      . . .
      public ComboLock(int secret1, int secret2, int secret3) { . . . }
      public void reset() { . . . }
      public void turnLeft(int ticks) { . . . }
      public void turnRight(int ticks) { . . . }
      public boolean open() { . . . }
}
```

Also, provide a tester class, `ComboLockTester`, that tests the above class using various combinations entered by a user inside a loop. User will exit from the loop by entering three 0s as the combination.

Problem 2. (20 points)

Implement a superclass `Appointment` and subclasses `OneTime`, `Daily`, and `Monthly`. An appointment has a description (for example "see the dentist") and a date. Write a method `occursOn(int year, int month, int day)` that checks whether the appointment occurs on that date. For example, for a monthly appointment, you must check whether the day of the month matches. Then fill an array of `Apointment` objects with a mixture of appointments. Have the user enter date and print out all appointments that occurs on that date.

Also, provide a tester class, `AppintmentTester`, for the above classes, in which create various types of appointments and test the expected outputs.

Problem 3. (20 points)

Consider an interface

```
public interface NumberFormatter {
      String format(int n)
{
```

Provide four classes that implement this interface as follows: A `DefaultFormatter` formats an integer in the usual way. A `DecimalSeparatorFormatter` formats an integer with decimal separators; for example, one million as `1,000,000`. An `AccountingFormatter` formats negative numbers with parentheses; for example, `-1` as `(1)`. A `BaseFormatter` formats the number in base `n`, where `n` is any number between `2` and `16`, that is provided in the constructor.

Also, provide a tester class, `FormatTester`, that tests above classes using various integer values read from an input file, `Numbers.txt`, and creates an output file, `FormattedNumbers.txt`, and add the formatted numbers into it. For instance if the input file contains:

```
5  10000  1000000  36
```

Then output file, based on the methods you call for these numbers, can contain something like:

```
Default Format: 5 -10000 1000000 36
Decimal Format: 5 -10,000 1,000,000 36
Accounting Format: 5 (10000) 1000000 36
Base 8 Format: 5 -23420 3641100 44
Base 2 Format: 101 -10011100010000 11110100001001000000 100100
```

Problem 4.   (20 points)

The CSV (or comma-separated values) format is commonly used for tabular data. Each table row is a line, with columns separated by commas. Items may be enclosed in quotation marks, and they must be if they contain commas or quotation marks. Quotation marks inside quoted fields are doubled. Here is a lone with four fields:

```
1729, San Francisco, “Hello, World”, “He asked: “”Where are you going?”””
```

Implement a class `CVSReader` that reads a CSV file, and provide the following methods:

```
int numberOfRows()                // Returns the number of lines in the CSV file
int numberOfFields(int row)   // Returns the number of fields in a particular row
String field(int row, int column)        // Returns the field in a particular row and colum
```

Then, test your class with the tester class, provided, and with the two CSV files, provided.

Problem 5.   (20 points)

Airline seating. Write a program that assigns seats on an airplane. Assume th airplane has 20 seats in `first` class (5 rows of 4 seats each, separated by an aisle) and 90 seats in `economy` class (15 rows of 6 seats each, separated by aisle). Your program should take three commands: add passengers, show seating, and quit. When passengers are added, ask for the class (first or economy), the number of passengers travelling together (1 or 2 in first class; 1 to 3 in economy), and the seating preference (`aisle` or `window` in first class; `aisle`, `center`, or `window` in economy). Then try to find a match and assign the seats. If no match exists, print a message. Your solution should include a class `Airplane` that is not coupled with the Scanner or PrintStream classes. Follow the design process that was described in Chapter 12.