CST8130: Data Structures
Midterm: Part B

Thursday October 30, 2014

Course Professor: Rex Woollard

Name: _____

1. This midterm test will be conducted during the regular 2-hour lecture.
2. You will have no aids in completing this test.
3. The midterm is split into two parts: A and B.
4. Part A is multiple choice. All answers are to be entered on a separate answer sheet. At the end of the test period, you can retain the Part A question set. If you record your answers on the question set, you can check your answers with the solution set which will be posted after the test. Part A will be worth 12 marks.
5. Part B involves writing Java program code to solve a problem. Part B will be worth 15 marks for the program code and 3 marks for the memory map (18 in total). Write your answer for Part B on separate blank sheets of paper. Your solution will be stapled to this test paper when complete.
6. You will begin with Part A. When you have completed Part A, submit it to me. At that point, you can choose to take an unsupervised break before receiving Part B.
7. After the completion of Part B, you may ask for the return of the Part A original exam paper so that you can compare your answers with the answers that will be posted on my website. I will retain the separate answer sheet.

Programming Problem (15 marks)

You will be implementing a simplified airline information system. You will be provided with the following file as a source of input. Notice that there are 8 flights in the sample data: *PORTER1234*, *AIRCAN3332*, etc. The file could hold many more than 8 flights. The flight information includes the departure city and the arrival city, along with a count of the number of passengers.

The first flight (*PORTER1234*) has 5 passengers.

The second flight (*AIRCAN3332*) has 2 passengers.

You will need to create a *Flight* class that stores the basic flight information shown here. The *Flight* class will also have some kind of collection to manage the *Passenger* objects associated with the flight.

Each *Passenger* object will consist of a name and the fare that they paid for the flight.

Not the following simplifications:

- Person's name has an underbar character (_) to tie the first to the last name. Therefore, you can read the name from the *Scanner* with a single call to *next()*.
- The city name may also use an underbar to tie two parts together (such as *New_York*).
- The *Scanner* object is already opened in *main()*. That single *Scanner* object can be used consistently throughout.
- No need to include any *import* statements.

```
PORTER1234 Ottawa Halifax 5
  Rex_Woollard 182.23
  Bob_Bangor 838.23
  Betty-Jo_Biolovski 139.75
  Jim_Hill 224.93
  Will_Smith 222.23
AIRCAN3332 Halifax Toronto 2
  Bill_Jones 538.23
  Frank_Wills 844.22
PORTER8383 Toronto New_York 3
  Nancy_Black 139.75
  Conrad_Black 999.99
  Brian_Mulroney 9992.32
PORTER3383 New_York Toronto 0
WESTJET1111 Toronto Calgary 0
PORTER3234 Halifax Ottawa 1
  Sarah_Smith 135.99
AIRCAN2332 Toronto Halifax 0
WESTJET3111 Calgary Toronto 1
  Eric_Wilson 194.33
```

flights.txt

I've supplied class *FlightTest*. Your solution must work with my *FlightTest* class. You do not need to re-write this class on your test paper. When you look at this class, you will see the classes that are directly referenced. There is no direct reference to the *Passenger* or *Flight* classes, but they will be required to support your solution.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

public class FlightTest {

    public static void main(String[] args) throws FileNotFoundException {
        FlightDataBase flightDataBase = new FlightDataBase();
        flightDataBase.load(new Scanner(new File("Flights.txt")));

        System.out.println("Flights BEFORE Sorting");
        flightDataBase.display();

        System.out.println("\nFlights AFTER Sorting on Passenger Fare");
        flightDataBase.sortPassengers(Passenger.CompareFare.instance);
        flightDataBase.display();

        System.out.println("\nFlights AFTER Sorting on Passenger Name");
        flightDataBase.sortPassengers(Passenger.CompareName.instance);
        flightDataBase.display();
    }
}
```

Memory Map (3 marks)

Draw a memory map the convinces me that you understand how you are organizing your data structures. Include enough real sample data to make the diagram clear.

The following output will result from the execution of this program:

```
Flights BEFORE Sorting
Flight Number:PORTER1234  Depart:Ottawa   Arrive:Halifax  Number of Passengers:5
  1: Name:Rex_Woollard Fare:182.23
  2: Name:Bob_Bangor Fare:838.23
  3: Name:Betty-Jo_Biolovski Fare:139.75
  4: Name:Jim_Hill Fare:224.93
  5: Name:Will_Smith Fare:222.23
Flight Number:AIRCAN3332  Depart:Halifax  Arrive:Toronto  Number of Passengers:2
  1: Name:Bill_Jones Fare:538.23
  2: Name:Frank_Wills Fare:844.22
Flight Number:PORTER8383  Depart:Toronto  Arrive:New_York Number of Passengers:3
  1: Name:Nancy_Black Fare:139.75
  2: Name:Conrad_Black Fare:999.99
  3: Name:Brian_Mulroney Fare:9992.32
Flight Number:PORTER3383  Depart:New_York Arrive:Toronto  Number of Passengers:0
Flight Number:WESTJET1111 Depart:Toronto  Arrive:Calgary  Number of Passengers:0
Flight Number:PORTER3234  Depart:Halifax  Arrive:Ottawa   Number of Passengers:1
  1: Name:Sarah_Smith Fare:135.99
Flight Number:AIRCAN2332  Depart:Toronto  Arrive:Halifax  Number of Passengers:0
Flight Number:WESTJET3111 Depart:Calgary  Arrive:Toronto  Number of Passengers:1
  1: Name:Eric_Wilson Fare:194.33

Flights AFTER Sorting on Passenger Fare
Flight Number:PORTER1234  Depart:Ottawa   Arrive:Halifax  Number of Passengers:5
  1: Name:Betty-Jo_Biolovski Fare:139.75
  2: Name:Rex_Woollard Fare:182.23
  3: Name:Will_Smith Fare:222.23
  4: Name:Jim_Hill Fare:224.93
  5: Name:Bob_Bangor Fare:838.23
Flight Number:AIRCAN3332  Depart:Halifax  Arrive:Toronto  Number of Passengers:2
  1: Name:Bill_Jones Fare:538.23
  2: Name:Frank_Wills Fare:844.22
Flight Number:PORTER8383  Depart:Toronto  Arrive:New_York Number of Passengers:3
  1: Name:Nancy_Black Fare:139.75
  2: Name:Conrad_Black Fare:999.99
  3: Name:Brian_Mulroney Fare:9992.32
Flight Number:PORTER3383  Depart:New_York Arrive:Toronto  Number of Passengers:0
Flight Number:WESTJET1111 Depart:Toronto  Arrive:Calgary  Number of Passengers:0
Flight Number:PORTER3234  Depart:Halifax  Arrive:Ottawa   Number of Passengers:1
  1: Name:Sarah_Smith Fare:135.99
Flight Number:AIRCAN2332  Depart:Toronto  Arrive:Halifax  Number of Passengers:0
Flight Number:WESTJET3111 Depart:Calgary  Arrive:Toronto  Number of Passengers:1
  1: Name:Eric_Wilson Fare:194.33

Flights AFTER Sorting on Passenger Name
Flight Number:PORTER1234  Depart:Ottawa   Arrive:Halifax  Number of Passengers:5
  1: Name:Betty-Jo_Biolovski Fare:139.75
  2: Name:Bob_Bangor Fare:838.23
  3: Name:Jim_Hill Fare:224.93
  4: Name:Rex_Woollard Fare:182.23
  5: Name:Will_Smith Fare:222.23
Flight Number:AIRCAN3332  Depart:Halifax  Arrive:Toronto  Number of Passengers:2
  1: Name:Bill_Jones Fare:538.23
  2: Name:Frank_Wills Fare:844.22
Flight Number:PORTER8383  Depart:Toronto  Arrive:New_York Number of Passengers:3
  1: Name:Brian_Mulroney Fare:9992.32
  2: Name:Conrad_Black Fare:999.99
  3: Name:Nancy_Black Fare:139.75
Flight Number:PORTER3383  Depart:New_York Arrive:Toronto  Number of Passengers:0
Flight Number:WESTJET1111 Depart:Toronto  Arrive:Calgary  Number of Passengers:0
Flight Number:PORTER3234  Depart:Halifax  Arrive:Ottawa   Number of Passengers:1
  1: Name:Sarah_Smith Fare:135.99
Flight Number:AIRCAN2332  Depart:Toronto  Arrive:Halifax  Number of Passengers:0
Flight Number:WESTJET3111 Depart:Calgary  Arrive:Toronto  Number of Passengers:1
  1: Name:Eric_Wilson Fare:194.33
```

FlightDataBase.java

```
import java.util.*;

public class FlightDataBase {
    private List<Flight> collectionFlights = new ArrayList<>();

    public void load(Scanner input) {
        while (input.hasNext()) {
            collectionFlights.add(new Flight(input));
        }
    } // end load()

    public void sortPassengers(Comparator<Passenger> compare) {
        for (Flight flight : collectionFlights) {
            flight.sortPassengers(compare);
        }
    } // end sortPassengers()

    public void display() {
        for (Flight flight : collectionFlights)
            flight.display();
    } // end display()

    @Override
    public String toString() {
        return String.format("Total Flights:%d", collectionFlights.size());
    } // end toString()
}
```

Flight.java

```
import java.util.*;

public class Flight {
    private String flightNumber;
    private String cityDepart;
    private String cityArrive;
    private List<Passenger> listPassengers = new ArrayList<Passenger>();

    public Flight(Scanner input) {
        flightNumber = input.next();
        cityDepart = input.next();
        cityArrive = input.next();
        int numberOfPassengers = input.nextInt();
        for (int i=0; i<numberOfPassengers; ++i)
            listPassengers.add(new Passenger(input));
    } // end constructor()

    public void display() {
        System.out.println(toString());
        int count = 0;
        for (Passenger passenger : listPassengers)
            System.out.printf("  %d: %s\n", ++count, passenger);
    } // end display()

    public void sortPassengers(Comparator<Passenger> compare) {
        Collections.sort(listPassengers, compare);
    } // end sortPassengers()

    @Override
    public String toString() {
        return String.format("Flight Number:%-11s Depart:%-8s Arrive:%-8s Number of
Passengers:%d", flightNumber, cityDepart, cityArrive, listPassengers.size());
    } // end toString()
}
```

Passenger.java

```
import java.util.*;

public class Passenger {
    private String name;
    private double fare;

    public Passenger(Scanner input) {
        name = input.next();
        fare = input.nextDouble();
    }

    @Override
    public String toString() {
        return String.format("Name:%s Fare:%.2f", name, fare);
    }

    public static class CompareFare implements Comparator<Passenger> {
        public static final CompareFare instance = new CompareFare();
        @Override
        public int compare(Passenger lhs, Passenger rhs) {
            return lhs.fare<rhs.fare?-1:lhs.fare>rhs.fare?1:0;
        }
    }

    public static class CompareName implements Comparator<Passenger> {
        public static final CompareName instance = new CompareName();
        @Override
        public int compare(Passenger lhs, Passenger rhs) {
            return lhs.name.compareTo(rhs.name);
        }
    }
}
```