COMP 1030

Lab #5

Control Structures

Introduction

During this assignment you will build two java classes. The first class will contain the required state and behaviour for the object **but NO main method**. The second class will contain simply the main method to give the JRE an entry point into the program, a line to instantiate a new object based upon the first class and a few lines to exercise the functionality of the first class.

Additionally – you will also be capturing information from the user during this assignment. To do so you will need to use the java.util.Scanner class. The Scanner object has behaviour that allows you to capture information from the keyboard such as nextFloat() – to capture floats, nextInt() – to capture ints, nextLine()-to capture Strings. Below is sample code for guidance.

```
import java.util.Scanner;
public class HelloWorld{

     public static void main(String []args)
     {
        Scanner in = new Scanner(System.in);
        System.out.print("Deposit Amount?");
        float num1 = in.nextFloat();
        System.out.println("deposit amount was " + num1);
        in.close();
     }
}
```

When writing your code, keep these guidelines in mind:

- Start each class with the proper javadoc comment header. The first line of that comment should be the purpose of the class not just its name.
- Provide a comment for **each** speciality method.
- Follow the layout for your class as illustrated below:

    Javadoc comment header
    Import statements (if required)
    Class declaration
        State (instance variables/data)
        Constructor(s) (if required)
        Behaviour(s) (method(s))
    Close class declaration

- Use whitespace and indentations to make your code more readable and easy to debug.
- Be sure to clearly understand your work – do not simply copy code from someone else.

Instructions: Use the BlueJ IDE to complete this assignment.

Challenge 1

Write a java class that models an Airline Reservation and adheres to the following criteria:

- 4 fields: firstName, lastName, flightNumber, seatNumber
- A constructor that takes no arguments.
- A constructor that takes four arguments
- A setter and getter for each field that simply sets and gets the field value.
- A method which returns a "seating class" for each seat number (use a switch statement)
  - 1-2: First Class
  - 3-4: Second Class
  - 5-6 Third Class
  - 7-8: Fourth Class

Challenge 2

- Write a method which concatenates the first and last name then checks the name against a "nofly list " which contains three names: Jack Blue, Jack Green and Jill White. If the name matches (regardless of case differences) the method will return the code "9999".

Challenge 3

- Write a method which will concatenate all object state into string, then will return a portion of that string based upon the first and last number of the seat number assuming that the seat number is a two digit number in which the first digit is smaller than the second and that the second number is no greater than the total number of characters in the concatenated string. (Do not test for this just assume it will be the case when you test)

Challenge 4

- Create a second class called AirlineReservationTestHarness which contains a **main method** to test the first class:
  - **Note: Any interaction with the user must be accompanied with an appropriate message.**
  - Instantiate an AirlineReservation object and pre-populate the four fields with random data you make up.
  - Print out the object state, including the seating class and a portion of that string based upon the first and last number of the seat number.
  - If the passenger is on the no fly list print the word "ALERT" on the screen 7 times, with each word appearing on the screen approximately every second.
  - Instantiate a second AirlineReservation object with no data.
  - Ask the user for the data required to fill each field of the record.
  - Print out the data, including the seating class, and a portion of that string based upon the first and last number of the seat number
  - **Note: Any interaction with the user must be accompanied with an appropriate message.**

Things to consider for success:

- Follow the layout for your class as illustrated above.
- Write a section at a time and compile after each section so you do not have a volume of compiler errors to deal with.
- Comment all speciality methods.
- Use indents and whitespace appropriately to make your code readable.
- Be sure that you actually understand the code you have written, simply copying someone else's code will not aid in your understanding of the java language.
- Stay focused and work diligently, collaborate with others if you are stuck.