BSc (Hons) in Computing in Software Development
BSc (Hons) in Computing in Games Development
Stage 2, Semester 2

Object Oriented Programming
Practical Lab Examination
Repeat CA2 Weighting: 20%

Instructions

Students are allowed to access locally:

- code previously written by them
- cheat sheets
- other code that they have gathered prior to the examination.

No communications of any form between candidates is permitted during the examination.

Date: 25th February 2022

Type your **name** and **class group** at the top of each source file. Answer all questions.

Download the ZIP file from Moodle. It contains the 5 files which you will be required to modify: PartA.java, Flight.java, Permanent.java, Contract.java, MainApp.java.

Part A

A 3 * 3 Matrix that is filled with the distinct integers 1,2,3,4,5,6,7,8,9 is called a MAGIC square if the sum of the elements in each row, in each column and in the two diagonals is the same value. For example, in the following square they all add up to 15:

2	7	6
9	5	1
4	3	8

In the file PartA.java, write a boolean method *checkIfMagicSquare*() that takes a 2D int array as argument and returns true if it represents a magic square, false otherwise.

Assume:

The array has 3 rows and 3 columns.
Each of the numbers [1,9] occur in the array.
Each cell contains a distinct integer.

Part B

Description

FlightsAway airline is developing a system to store details on staff allocation (both permanent and contract staff) to flights. Download the initial starting code for the solution to this problem, examine the code and make the improvement/modifications as suggested below.

The structure developed to date has 3 classes – Flight, Permanent, Contract - with attributes as follows:

Class Name	Attributes
Flight	id, origin, destination
Permanent	id, name, bonus, base-airport, overnight-allowance
Contract	id, name, bonus, base-airport, top-up bonus, number of flight-hours

All staff are either: Permanent or Contract.
Staff will have their bonus updated using different calculations depending on whether they are permanent or contract (see q6).

Text File: “staff.txt”

Functional Requirements

Write the code required to implement the following functionality for the entities outlined above, and write appropriate method calls to test your implementation at each stage:

1. Design an appropriate inheritance hierarchy that uses a class called **Staff** to act as a superclass of the **Permanent** and **Contract** classes. Implement the Staff class as an *abstract* class and make appropriate modifications to the Permanent and Contract classes. Include appropriate constructors, getters, setters, toString, equals and hashCode methods.

Examine the Flight class and make the following modifications. Test your methods by calling them from the MainApp.

2. Modify the Flight class to include an additional field, called **staffList**, to store the list of staff assigned to a flight (both Contract and Permanent staff). (Remember to instantiate the list).

3. Write a method *addStaff*(...) in the Flight class that allows a Staff object to be added to the **flightList**. Duplicate staff names are not permitted.

4. Write a method *displayStaff*() that displays all staff details from a flight. Neat formatting of output is not required.

5. Write a method *displayNamesWithFlightHours*() that displays only the names (i.e. not all fields)) of all staff but also shows the flight hours for any Contract staff on the flight.

6. All staff on a flight (both Permanent and Contract) will have their **bonus** increased if their base airport is different from the flight destination. In this case:
   - Permanent staff get an increase in their bonus equal to the overnight-allowance amount.
   - Contract staff get an increase in their bonus equal to their top-up amount, but only if in addition their number of flight hours is greater than 100.

   Write a method *updateAllStaffBonus*() in the Flight class to implement this logic, making use of an appropriate *updateBonus*() method in each of the staff classes.

7. In a clearly labelled comment in Flight, write an explanation of polymorphism and indicate where in your code (for Q6) polymorphic behaviour is used.

8. Write a method *sortStaffByName*() to sort the staffList in alphabetic order of staff name within base airport, using a **Comparator** called StaffNameComparator.

9. Write a method *readStaffFromFile*() that reads data from the file "staff.txt", and populates *staffList*. The "staff.txt" file is given. (In the text file, the code "P" indicates a permanent staff member and "C" indicates a contract staff member).

10. Write appropriate exception handlers for all file processing operations.

11. Change the first line of code in the Staff class to:
    "public class Staff implements Comparable"
    (a) In a comment in Staff, briefly explain the meaning of this piece of code and what else needs to change for it to work.
    (b) Write any code that needs to be written as a consequence of this line.

When you have finished writing code please zip all source files into a zip file called *yourFirstName_yourLastName_CO2_CA2*.zip and submit on Moodle. Please do not leave your PC until you have checked with your lecturer that your submission is visible on Moodle.