BSc (Hons) in Computing in Software Development

BSc (Hons) in Computing in Games Development

Stage 2 – December 2021

Practical Examination for CA2 (20%)

Time allowed: 100 minutes

Answer ALL four questions.

(Answer questions 1, 2 & 3 in sequence. Q4 is independent.)

Questions do NOT carry equal marks.

OPEN BOOK Exam

Communicating with others is NOT permitted

Instructions

Please find the GitHub repo link called "**oop-ca2-Dec21-AirplaneManager-STARTER**", in the Assessments section of Moodle, and clone (open from VCS) the project in IntelliJ.

The project contains a number of starter Java classes.

Class **Airplane** defines an Airplane class (Abstract type) with two fields (*id* and *type*), a constructor, getters and setters and a *toString*() method. You will need to refer to this class, but **do not change** it.

The main App class contains a *start*() method in which you can write code to create and manipulate Airplane objects via an ***AirplaneManager*** class.

Create a private GitHub Repo on your GitHub account, and push the project to that Repo. In GitHub, share the Repo with your lecturer.

Please remember to **save your work** at regular intervals, and push to the Remote Repo regularly. If you get stuck on a question, comment out your attempted solution and move on.

At the end of the exam, please compress your work into one ZIP folder:
[File > Export > "Project to Zipped file …"] and upload to the CA2 Upload link in Moodle. Check and confirm that it has been uploaded to Moodle.

Also, PUSH your project to your Github repo and email the Repo URL to your lecturer.

Question 1 (10 marks)

Write code to implement the following in the ***CargoAirplane*** class:

1. Define two integer fields:
    a. MAX_LOAD_KILOS {maximum load plane can carry}
    b. currentLoad {current load on the plane}
2. Make this class a **sub-class** of the Airplane class (i.e. inherit from Airplane)
3. Define a **constructor** that accepts two arguments (*type, maxLoad*) and initialize the relevant fields with these values.
4. Define a *toString*() method that returns **all** fields.

In the **App** class:

5. Uncomment the code in *start*() that creates two *CargoAirplane* objects, and print out both objects to check that your *CargoAirplane* class is working.

Question 2 (10 marks)

Write code to implement the following in the ***PassengerAirplane*** class:

1. Define two fields:
    a. MAX_NUM_PASSENGERS of integer type
    b. **passengerList** – an ArrayList of names of type String
2. Make this class a **sub-class** of the Airplane class (i.e. inherit from Airplane)
3. Complete the constructor that accepts two arguments (*type, maxNumPassengers*) and initialize the relevant fields with these values.
4. Define a *toString*() method that returns **all** fields
5. Complete the *addPassenger*() method to accept a String *name*, and adds that name to the *passengerList*. The maximum number of passengers must not be exceeded.

In the App class:

6. Uncomment the code that creates the two *PassengerAirplane* objects.
7. Display both of the *PassengerAirplane* objects.

Question 3 (60 marks)

Write code to implement the following in the ***AirplaneManager*** and other classes:

1. Write a method ***add****(Airplane airplane)* that takes an Airplane as a parameter and adds that Airplane to the *airplaneList*.
2. Write code in the App *start*() method that will add the four Airplane objects, which have already been created, to the *AirplaneManager*.
3. Write a method ***displayAllAirplanes***() in *AirplaneManager* that will display all Airplanes. Call the method to confirm that it works.
4. Write a method ***displayAllPassengerAirplanes***() in *AirplaneManager* that will display **only** the *PassengerAirplanes*. Call the method to confirm that it works.
5. Write a method ***getAllCargoAirplanes***() that will **return** a **List** containing only *CargoAirplanes*. Call the method and print the values that it returns.
6. Write a method ***addPassengerNameToAirplane****(airplaneId, passengerName)* that will add the passenger name to the list in the airplane specified by id. Return true if added, and false otherwise. Call method to confirm that it works.
7. Write a method ***containsAirplane****(Airplane plane),* that takes an *Airplane* as an argument and returns *true* if the *Airplane* is in the AirplaneManager list, or *false* if not found. Airplanes are considered equal **only** if their **id** field values are equal. Make an appropriate call to the method, and output a result accounting for all possible return values.
8. Write a method ***findAirplaneByPassengerName****(passengerName)* that accepts a passenger name (String) and returns the *Airplane* object that contains the passenger or *null* if not found.
9. Write a method ***displayAllAirplanesInOrderOfType****(argument)* that will **sort** the list of planes in "type" order (alphabetic order), and display in that order. Call the method, passing in the required parameter.

Question 4 (20 marks)

The class ***CityDistanceManager*** stores a String array of City names, and a 2D array of distances between those cities. The index positions in the cities names array correspond to the index positions in the 2D array. Distances (kms) between each of five Irish cities are stored in a 2-D array of *int* called ***distances***. The names of the 5 cities are stored in the ***cities*** array.

The method ***printCitiesData()*** - given in the code - displays this data neatly in a table.

You are required to add two additional features to this class by completing the code for the two method stubs provided. Make use of additional methods as required. You may ignore invalid user input (i.e. assume that the user only enters city names that exist in the array).

In the class **CityDistanceManager** and in class App:

1. Write the method ***findDistanceBetween****(String city1, String city2)* that takes two city names, searches for the distance between the two cities, and returns that distance. Check that the call to the method works correctly.

2. Write a **JUnit** test method ***testFindDistanceBetween***() to test the above method.

3. Write the method ***findClosestCityTo****(String searchCity)* that takes the name of a city (the base city), and finds the distance to the nearest city, and returns the name of that city. Check that the call to the method works correctly.

END