Air traffic control system

Assignment brief:

Time 3 hr (+1 hr to upload) (closed book).

You have been given some code that is currently being developed as part of an **Air Traffic Control** application. Not all the requirements have been implemented. It is your task to implement these, test your **solution** classes and raise the coding standards of all the code.

Create a **project solution** (named **p2<your student id> e.g. p26048201).** Create a package named **p2**. Add **Aircraft.java** to the solution**. Ensure your name and student number are placed in the Javadoc comments of all the classes you create.**

Part 1 – 40%

Using your knowledge of OOP you should update the code based on the following:

1. The system is expected to support many ***derivative*** (sub classes) of *Aircraft* such as **Airline**, *Helicopter*, *Glider, Drone* etc. There will be no need to ever instantiate an *Aircraft* class in the system, it this therefore *Abstract*. You are responsible for developing the ***Airline*** class. This is the only subclass you need to develop. It should have all the properties and methods of the *Aircraft* class but also include a new attribute, **Engine Type** which should be limited to the values ***jet*** or ***propeller***.

2. **Business rules for class attributes –**
   - **Current Speed -** minimum value is 0 and maximum is 800 (inclusive). This is in mph.
   - **Distance to Airfield –** minimum value is 1 and maximum is 20000 (inclusive). This is in miles.
   - **Aircraft Code –** exactly 6 characters (any characters including letters, numbers or symbols but first character should be an uppercase **A.** No trailing whitespace characters.)
   - **Engine Type** - which should be limited to the values **jet** or **propeller**.
   - Return an appropriate exception with an appropriate exception message i.e. "*INVALID CODE*", "*INVALID SPEED*" etc. if an attempt is made to set outside the range or allowable values.

3. Some but not all of the Aircraft class derivatives (sub classes) are expected to support *a method that will calculate the estimated time (in minutes) any aircraft subclass will take to reach the airfield.* This will require each class that needs this functionality to have a method named **timeToAirfield.** The Airline class requires this method and should return the time (in minutes) value based on the following formula.

time = distance to airfield / current speed * 60

[CONTINUED OVER]

Part 2 - Airtraffic Control class - 30%

Create an ***Airtraffic Control*** class to support the system for searching and other utility methods. Each method should be ***static*** and accept and return an ***ArrayList*** of **appropriate type**. Using your knowledge of OOP you should update the code based on the following:

1. Create a **searchBySpeed** method i.e. search for all objects in the parameter argument ArrayList that are *moving (current speed) within a specified range* e.g. between 100 and 300 (range inclusive of both values). You should return an ArrayList containing any that satisfy the search criteria.

2. Create a **searchForAllByEngineType** method i.e. search for all objects in the parameter argument ArrayList that match a specified engine type. You should return an ArrayList containing any that satisfy the search criteria.

3. Create a **searchForLandingList** method i.e. search for all objects in the parameter argument ArrayList that are within a given time (inclusive) of the airfield and have a specified engine type. You should return an ArrayList containing any that satisfy the search criteria.

Part 3 – Testing – 30%

1. Unit Test the application.

When complete compress (zip) the entire ***Eclipse solution*** and upload to **Assignments** (P2 assessment) on CANVAS. Remember to record and then upload a short commentary walk-through of your code with your solution. Keep the separate screen recording safe (no need to upload at this point).

Now : check the uploads to ensure you have submitted the correct files.

[END]

Sport stats system

Assignment brief:

Time 3 hr (+1 hr to upload)

You have been given some code that is currently being developed as part of a **Sport Stats** application specifically analysing recent **Six Nations Rugby Player** stats. Not all the requirements have been implemented. It is your task to implement these and raise the coding standards of all the code.

Create a **project solution** (named **<Your Name><Student Number>p3 e.g. AidanMcGowan3048614P3).** Create a package named **p3**. Add **StartApp.java** to the solution and the **playerstats.csv. Ensure your name and student number are placed in the Javadoc comments of all the classes you create.** The StartApp has been partially written with a menu.

The application will run (start) from the StartApp.java, initially reading in the data from the **playerstats.csv** file and then perform a number of menu driven operations.

Part 1 – Data mapping, storage and read from file - 50%

Using your knowledge of OOP you should add/update the code based on the following:

1. Analyse the data in the **playerstats.csv** and create a class (**Player.java**), include the following re-codes…

    I. **Country code** should be recoded as: 1 = ENG, 2 = FRA, 3 = IRE, 4 = ITA, 5 = SCO, 6 = WAL

    II. Split and record the **name** separately (two instance vars) as **first name** and **last name** such that "Owen Farrell" would be recorded in the Player.java class as First name *Owen* and Last name is *Farrell*.

    III. Include a double field (instance var) in the Player.java class to store the **percentage of games won** for each player based on the number of games won **(Games Won) divided** by the total games played **(Total Matches) * 100.**

2. Conduct a simple unit test for the **Player** class. (Note there are no other validation or business rules for the Player class).

3. In the **StartApp.java** class read and store the data in an appropriate JCF container. Note there will be NO duplicate data entries in the dataset.

[CONTINUED OVER]

Part 2 – Functions – 50%

Having read the data from the csv file complete the menu driven functions as outlined below. An example of the expected format is shown for each function. Note if you were unable to recode the input data as specified in Part 1 for Name and Country use the uncoded raw data (as appropriate).

1. Display all players to screen. Example output…

```
All players

Country         :SCO
First name      :Sam
Last name       :Johnson
Position        :Back
Club            :Glasgow
Total games     :4
Wins            :1
Height          :1.80
Points          :10
Influence       :66
Percent wins    :25.0

Country         :WAL
First name      :Taulupe
Last name       :Faletau
Position        :Forward
Club            :Bath
Total games     :31
Wins            :22
Height          :1.88
Points          :15
Influence       :66
Percent wins    :70.96774193548387
```

etc…

2. Display all players from Ireland. Example output…

```
All players from Ireland
Jacob Stockdale
Josh vanderFlier
James Ryan
Ultan Dillane
Tadhg Furlong
```

etc…

3. Display the highest point scorer (Points scored) in the format, first name, last name, country, points scored. Example output…

   e.g. Keith Wood IRE 356

   (note: not actual answer based on csv data provided)

4. Display all players ordered by height (tallest first). Example output…

```
2.11    Devin Toner
2.03    James Ryan
2.03    Adam Beard
2.01    Paul Willemse
2.01    Romain Taofifenua
2.01    Jake Ball
2.01    Grant Gilchrist
```

etc…

[CONTINUED OVER]

5. Display each **club** (in alphabetical order with the cumulative number of games played in the six nations (Total Matches) from each player from that club. Example output…

```
Bath : 70
Benetton : 187
Bordeaux : 25
Brive : 5
Cardiff Blues : 22
Clermont : 5
Connacht : 18
```

etc…

6. Capitalise the Last name for all players. Then in a new **Thread** export/write to a new file (*playerstats_updated.csv*) in the format: Last name, First name and Country. Note, include the header as shown. Example csv output …

```
Last name,First name,Country
JOHNSON,Sam,SCO
FALETAU,Taulupe,WAL
ADAMS,Josh,WAL
MAITLAND,Sean,SCO
```

etc…

When complete compress (zip) the entire ***Eclipse solution*** and upload to **Assignments** (P3 assessment) on CANVAS. Remember to record and then upload a short commentary walk-through of your code with your solution (upload that too). Keep the separate screen recording safe (no need to upload at this point, I will be sampling these).

Now : check the uploads to ensure you have submitted the correct files (in the correct area).

[END]