**Java Programming OOP – Summative assessment\_2**

**Airport Flight Simulation**

Assignment set up: A scenario is provided for candidates in the form of a company specification for a service they require.

This assignment is made up of three tasks

* Task A - provides a detailed design specification that should be followed by candidates when developing their program.
* Task B - requires the candidate to test the program and provide documentation.
* Task C - provides criteria that should be followed by candidates when producing their work.

**Scenario**

**Sky Ocean Ltd** is a flight training agency is located in a private airport. You are asked to help the company to develop a run-time monitoring software systems in order to facilitate monitoring the positions and status of each aircraft in sky and ground at any time. Flight operators should be able to change the current status of aircrafts at any time, following their operational procedures.

The application should be also serving the company’s management team to administrate the aircrafts by adding, removing or upgrading them remotely.

**Operational procedures:**

* Sky Ocean Ltd operates aircrafts only in one airport
* The airport has capacity to handle 12 aircrafts. There are 12 docking position specifying by their unique spot id allocated for park each aircraft in the airport.
* The company owns a different types of aircrafts, such as airplane, helicopter, jet and so on
* Each aircraft uses a specific type of fuel (table1)
* The Airport has only one runway. The runway can be found in a status of “*Busy*” or “*Free*” at any time.
* Each aircraft must be taken a sequence of compulsory actions to operate a flight in a correct order. Theses sequences are as follows:
  + **“*Fuelling*”**
  + **“*DockOut*”**
  + **“*Accelerating*”**
  + **“*TakeOff*”**
  + **“*Ascending*”**
  + **“*Flying*”**
  + **“*Deaccelerating”***
  + **“*Descending*”**
  + **“*Landing*”**
  + **“*Deaccelerating”***
  + **“*DockIn*”**
* Each aircraft is in one of these status at any time.
* An aircraft cannot be switched to “*TakeOff”* status before reach to a recommended speed while is in “*Accelerating”* status (table 1)
* An aircraft needs to reach to a specific altitude in “*Ascending*” status in order to be switched to “*Flying*” status (table 1)
* Each aircraft can only be allowed to stay in “*fly*” status as far as has ¼ of its full tank fuel. Needs to be switched to “*descending*” status immediately (table 1)
* An aircraft needs to reach to a specific altitude in order to be switched to “Landing” status (table 5)
* An aircraft cannot be switched to “*DockIn”* status before slowing down to a specific speed while it’s in “*Deaccelerating”* status (table 1)
* An Aircraft can only be sat to “*takeOff*” or landing status when the runway in is “*Free*” status
* Calculate the fuel consumption only when the aircraft is in *“fly”* mode.
* Each aircraft has a capacity to carry 200 gallons.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Aircraft | Fuel consumption  (miles per gallon (US)) | Speed to take off (miles) | Altitude to flying (foot) | Altitude to land (foot) | Speed to land (miles) | Speed to DockIn (miles) |
| SingleEngine Airplane | 25 | 150 | 2000 | 500 | 100 | 10 |
| TwinEngine Airplane | 10 | 100 | 5000 | 700 | 150 | 10 |
| Helicopter | 1.95 | 0 | 50 | 100 | 50 | 0 |
| Jet | 0.2 | 200 | 10000 | 500 | 100 | 10 |

Table 1 – Sample of aircrafts’ associated figures (can be enhanced by your own research)

Use the instruction below is suggested to build appropriate constructs such as abstracts/Interface or solid classes and Enum type in order to implement the required functionalities in java OOP boundary.

Feel free to make any assumption to add/remove fields/methods to improve the usability of the application. However, make sure that your design and implementations supports readability and reusability of the software. Implementing generic functionalities which serve similar needs with the same purpose are highly encouraged.

**Task A**

Candidates can use the following detailed specification to fulfil the company's requirements:

In this task you are required to create and test a number of components. You can find a list of suggestions in blow. However, make your decision and feel free to make any assumption with which the functionalities and reusability can be improved.

**Flyable (interface):**

boolean getFul(fuel per gallon)

boolean dockOut()

boolean accelerate(speed per mile)

boolean takeOff();

boolean ascend(altitude per foot)

boolean fly(distance per mile)

boolean descend(altitude per foot)

boolean deaccelerate(speed per mile)

boolean land()

boolean dockIn(position enum)

**abstract class Aircraft:**

Attributes: - id, model, name, age, airCrew, fuelType, fuelTankCapacit, airCraftStatus, ………

Constructor/s (if applicable)

Appropriate methods

**More abstract classes can be created which are all extend from the aircraft, such as:**

* **abstract class Airplane extends Aircraft**
  + appropriate attributes, Constructor/s (if applicable) and methods.
* **abstract Helicopter extends Aircraft** 
  + appropriate attributes, Constructor/s (if applicable) and methods.
* **abstract Jet extends Aircraft**
  + appropriate attributes, Constructor/s (if applicable) and methods.

**Number of solid classes which extend from the Airplane abstract class:**

**Class SingleEngineAirplan extends Airplane implements Flyable (Solid class)**

* + appropriate attributes, Constructor/s (if applicable) and methods.
  + Implementing Flyable interface methods

**Class TwinEngineAirplan extends Airplane implements Flyable (Solid class)**

* + appropriate attributes, Constructor/s (if applicable) and methods.
  + Implementing Flyable interface methods

**Number of solid classes which extend from the Helicopter abstract class:**

**Class Apache extends Helicopter implements Flyable (Solid class)**

* + appropriate attributes, Constructor/s (if applicable) and methods.
  + Implementing Flyable interface methods

**Class SirkorSky extends Helicopter implements Flyable (Solid class)**

* + appropriate attributes, Constructor/s (if applicable) and methods.
  + Implementing Flyable interface methods

**Number of solid classes which extend from the Jet abstract class:**

**Class HondaJet extends Jet implements Flyable (Solid class)**

* + appropriate attributes, Constructor/s (if applicable) and methods.
  + Implementing Flyable interface methods

**Class GolfStream extends Jet implements Flyable (Solid class)**

* + appropriate attributes, Constructor/s (if applicable) and methods.
  + Implementing Flyable interface methods

**Class Airport**

Attributes: - name, location

An array (preferable ArrayList) which holds objects of aircrafts (polymorphism)

Constructor/s (if applicable)

Appropriate methods

**A number of Enums construct to keep the following details:**

AircraftStatus: - “*Fuelling*”, “DockOut”, “*Accelerating*” ,“*TakeOff*”, “*Ascending*”, “*Flying*”, “*Descending*”, “*Landing*”, “*Deaccelerating”* , “*DockIn*”

RunWayStatus: - “*Occupied*”, “*Free*”

DockingPositon:- “001A”,”002B”,”003C”,”004D”,”005E”

FuelTypes: - “Unleaded-Kerosene“, “Avgas-82-UL”, “Avgas-100”, “Avgas-100-LL”

**AirCraftSimulator (a java file with a main method)**: -

The Aircraft Simulatoractivities will be processed through a runtime menus:

* Main Menu: - offers 5 options as illustrates in figure 1, serve the operator to find, operate and monitor the aircrafts
* Admin menu: - is also offers 4 options as illustrates in figure 2, however this menu only serves the operator to administrate the aircrafts by adding, removing or editing them.

When the program terminates (exit option is chosen) status of all aircrafts with logged in an external file.

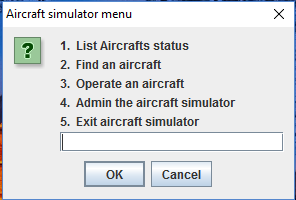


Figure. 1 Main menu

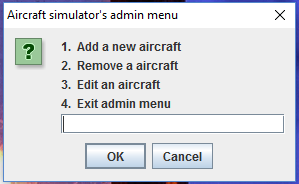


Figure 2. Admin flight simulator’s menu

**Task B**

In this task you are required to test the classes you have created using your Junit test class and provide evidence for the results. You are required to prepare a test plan, test log and a list of cases with expected and actual results.

You will also implement the Class JUnitTestAircraftSimulator using assert class’s methods testing the expecting figures in your test log.

**Task C**

Candidates should follow the criteria below when producing their work:

* The program conforms to the design specification.
* List of assumptions amended to the program specification
* The code is commented.
* Meaningful names are used for classes, methods and attributes using consistent naming conventions.

**Notes:**

Candidates should produce and submit the following programs:

Class AirCraft,

Class Airplan,

Class SIngleEngineAirplan, Class TwinEngineAirplan, Class Helicopter, Class Jet

Class AirCraftSimulator,

Interface Flyable,

Enums:

AircaftsStatus, RunwayStatus, DockingPositions, Fuel Types

Class JUnitTestAircraftSimulators – using assert class - expected results, actual results

Test data, test paln & test log, AirCraftsStatus Log file

Ensure that your name is on the top of each program.