# PROG2004 Assignment 2

Weight: **40%** of your final mark

Due: **Midnight on # of Week 6**

## Specifications

Your task is to complete various exercises in **IntelliJ**, using the **Java language (OpenJDK 19)**, and to submit these via the MySCU link created for this purpose.

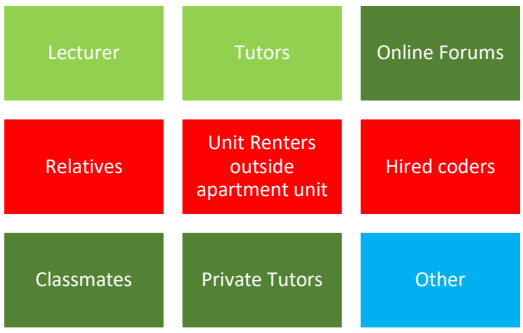
Marking criteria includes:

- Code compiles with OpenJDK 19
- Use of correct coding style, including the use of comments
- Accuracy of coding
- Use of suitable coding structures
- Correct submission and naming conventions of assessment items as required

## Getting Help

This assignment is to be completed individually. It is the opportunity to gain an understanding of the concepts of object-oriented programming and coding syntax. It is important that you master these concepts yourself. You are permitted to work from the examples in the study guide or textbook but you must acknowledge assistance from other textbooks or classmates. In particular, you **must not** use online material or help from others, as this would prevent you from mastering these concepts.

**Who can you get help from?** Use this diagram to determine from whom you may seek help with your program.




Encouraged

Attribution Required

Ask tutor

Not acceptable

## Getting started

In this assignment we will write the code that could form part of **a simple flight management system** for an airline company to manage their flight transactions.

To get started

- Create a new Java project called **username-assignment2** in IntelliJ using OpenJDK version 19 as the project SDK. In my case the project would be called **nsugia11-assignment2**.
- In the src directory create five classes called:
    - Person
    - Aircrew
    - Passenger
    - Flight
    - AssessmentTwo
- Copy the code for each class at the end of the assignment (see page 7-11) into the classes you just created.

Please note:

- Your assignment will be marked using IntelliJ. If your project files for your assignment were created in another IDE or do not open in IntelliJ, then your assignment **will not be marked**.
- Your assignment must compile using **OpenJDK version 19** or it will not be marked.

## Module 3 - Advanced collections

The following parts of the assessment covers the content in Module 3.

### Part 1 - Lists

The Flight class is missing the ability to store a collection of passengers who travel on a plane. For this part of the assignment:

1. Using a *LinkedList*, update the *Flight class* so that a flight can store a collection of passengers (i.e. data type Passenger class) who travel on a plane.

In addition to adding a *LinkedList*, you need to add the following methods to the Flight class that work with the LinkedList:

1. A method called *addPassenger* to add a passenger into a flight. This method will also check if there is an available seat in the flight before inserting a passenger. If there is

no available seat, the passenger will not be included and a message saying “The flight is fully booked. No available seat anymore” will be shown to the terminal.
2. A method called *removePassenger* to remove a passenger from a flight.
3. A method called *numberOfPassengers* that *returns* the number of passengers in the flight.
4. A method called *printPassengers* that prints the details of all passengers travelling in the flight (you must use an *Iterator* or you will get no marks).
5. A method called *allPassengers* that returns the LinkedList.

**Demonstration**

In the *partOne method* in the *AssessmentTwo class*:

1. Create a new Flight object.
2. Add a minimum of 5 passengers into the flight using the *addPassenger* method.
3. Remove a passenger using the *removePassenger* method.
4. Print the number of passengers in the flight to the terminal using the *numberOfPassengers* method.
5. Print all passenger details in the flight using the *printPassengers* method.

## Part 2 – Collections Class

At the moment there is no way to sort the passengers who travel in a flight. For this part of the assignment:

1. Create a class (you can choose the name) that *implements the Comparator interface*. When you implement the *compare method from the Comparator interface* you must use a *minimum of two* of the instance variables in your comparison.
2. Now create a method in the *Flight class* called *sortPassengers* that sorts the LinkedList using the *sort(List list, Comparator c) method in the Collections class*.

**Demonstration**

In the *partTwo method* in the *AssessmentTwo class*:

1. Create a new flight.
2. Add a minimum of 5 passengers into the flight using the *addPassengers* method.
3. Print all passenger details using the *printPassengers* method to show the order of the LinkedList before it is sorted.
4. Sort the LinkedList using the *sortPassengers* method.
5. Print the passengers again using the *printPassengers* method to show that the LinkedList has been sorted.

## Part 3 – HashMap class

The Flight class is also missing the ability to store aircrews who operate the flight and what their role in the flight is. For this part of the assignment:

1. Using a *HashMap*, update the *Flight class* so that a flight can store aircrews (i.e. Aircrew objects) who operate a flight and their associated role (i.e. a String). The role should be used as the key and the aircrew object should be used as the value.

In addition to adding a *HashMap*, you need to add the following methods to the Flight class that work with the HashMap:

1. A method called *addAircrew* to add an aircrew and their role to the HashMap.
2. A method called *removeAircrew* to remove an aircrew and their role from the HashMap.
3. A method called *printAircrew* that prints the details (key and all aircrew details) for all allcrews operate the flight (you cannot just print the HashMap, you must loop through it or you will get no marks).

**Demonstration**

In the *partThree method* in the *AssessmentTwo class*:

1. Create a new flight.
2. Add a minimum of 5 aircrews into the flight using the *addAircrew* method.
3. Remove an aircrew member using the *removeAircrew* method.
4. Print all aircrew details in the flight using the *printAircrew* method.

**Hint**

Remember that in a HashMap the key must be unique. You can add a number on the end of the key to facilitate this. For example:

- One aircrew may have the key *Pilot*
- One aircrew may have the key *Co-Pilot*
- One aircrew may have the key *FlightAttendantOne*
- One aircrew may have the key *FlightAttendantTwo*
- One aircrew may have the key *FlightAttendantThree*

# Module 4 – Advanced exception handling

The following parts of the assessment covers the content in Module 4.

## Part 4 – Implementing exception handling

At the moment we have not implemented any exception handling in our program. For this part of the assignment:

1. Where applicable, make sure that your setters confirm that the values they are writing to your instance variables are valid. If they are not, throw an IllegalArgumentException and print an appropriate error message.
2. Add any other exception handling that you feel is appropriate to your program.

**Demonstration**

In the *partFour method* in the *AssessmentTwo class*:

1. Create an object and:
    a. Pass a valid value to one of your setters that you modified to confirm that it does not throw an IllegalArgumentException.
    b. Pass a non-valid value to one of your setters that you modified to confirm that it throws an IllegalArgumentException.

# Module 5 – Input/output

The following part of the assessment covers the content in Module 5.

An important part of many programs is the ability to back up data to a file then restore it as needed. In this section of the assignment we will add this ability to our program.

**Hint for exporting and importing data**

A common way to store data in a file that needs to be imported at a later date is to use comma separated values (csv).

This means that we store a record on a single line and we separate values using a comma (,).

For example, imagine a passenger has the following information:

- name – Nehemia Sugianto
- PNR – KQKRCR ← (6 letters to identify a passenger uniquely)
- seat – A10 ← (3 digits). If the seat is not assigned, '000' will be the default value
- luggagecapacity – 10kg ← in kg

You could store the passenger in the file on a single line like:

**Nehemia Sugianto,KQKRCR, A10,10**

When you read the file each line in the file will contain the details for a single passenger.

You can then use the *split() method* from the *String class* to split the line into the individual values, and then use the values to create a new Student and add it to the LinkedList.

## Part 5 – Writing to a file

The Flight class is missing the ability to back up the passengers who travel in a flight and the aircrews who control the flight. For this part of the assignment:

1. Add a method to the *Flight class* called *exportPassengerManifest*.
2. The *exportPassengerManifest* method should write the details of all of the passengers that travel in a flight (i.e. stored in the LinkedList) to a file. The details for each passenger should be written on their own line.
3. You must make sure to add all appropriate exception handling and error messages.

**Demonstration**

In the *partFive method* in the *AssessmentTwo class*:

1. Create a new flight.
2. Add a minimum of 5 passengers into the flight using the *addPassenger* method.
3. Generate a passenger maniest to a file by calling the *exportPassengerManifest* method.

## Part 6 – Reading from a file

The Flight class is also missing the ability to restore the passengers who travel in a flight. For this part of the assignment:

1. Add a method to the *Flight class* called *importPassengerManifest*.
2. The *importPassengerManifest* method should read the file that was created by the *importPassengerManifest* method above.
3. When reading the file, you need to add all of the passenger into the flight (i.e. add them to the LinkedList).
4. You must make sure to add all appropriate exception handling and error messages.

Note: If you cannot add the passengers in the flight (i.e. add them to the LinkedList) you will still get marks for reading the file.

**Demonstration**

In the *partSix method* in the *AssessmentTwo class*:

1. Create a new flight.
2. Import the file you created in the previous part of the assignment.

3. Print the number of passengers in a flight to the terminal using the *numberOfPassengers* method to confirm that the correct number of passengers were imported.
4. Print all passenger details in the flight using the *printPassengers* method to confirm that the details of each passnger were imported correctly.

## Module 6 – Concurrency

The following part of the assessment covers the content in Module 6

### Part 7 – lock() and unlock() methods

We are using a LinkedList to store the passengers who travel in a flight. However, a LinkedList is not thread safe. This means that if multiple threads were performing operations on the passengers travel in a particular flight we could encounter issues. For this part of the assignment:

1. Use the *lock() and unlock() methods* to protect any critical sections of code in the *Passenger class* that perform any operations on the LinkedList that stores the passengers who travel in the flight.
2. You must make sure to add all appropriate exception handling and error messages.

Note: You may watch an additional video (see Video 1) on Blackboard that demonstrates how to use lock() and unlock() methods

## Submission

Zip your project into a file called username_A2.zip  For example, mine would be nsugia11_A2.zip. Please note that the extension of the file must be zip. Any other compression e.g. .rar will NOT be accepted.

Submit the zip file via the Assignment 2 link on MySCU by the due date. Please leave enough time for it to upload, so do not submit at the last minute!

**See the Starting Code below!**

## Starting Code

### AssessmentTwo class

```
public class AssessmentTwo {

    public static void main(String[] args) {
    }

    public void partOne(){
    }

    public void partTwo(){
    }

    public void partThree(){
    }

    public void partFour(){
    }

    public void partFive(){
    }

    public void partSix(){
    }
}
```

### Person class

```
public abstract class Person {

    private String name;
    private String email;
    private int age;

    public Person() {
    }

    public Person(String name, String email, int age) {
        this.name = name;
        this.email = email;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
```

```
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
}
```

## Aircrew class

```
public class Aircrew extends Person {

    private String position;  // can be "pilot", "flight attendant"

    public Aircrew (String name, String email, int age, String position){
        super(name, email, age);
        this.position= position;
    }

    public String getPosition() {
        return position;
    }

    public void setPosition(String position) {
        this.position = position;
    }
}
```

## Passenger class

```
public class Passenger extends Person {

    private String pnr;
    private String seat;
    private int luggageCapacity;

    public Passenger(String name, String email, int age, String pnr, String
    seat, int luggageCapacity) {
        super(name, email, age);
        this.pnr = pnr;
        this.seat = seat;
        this.luggageCapacity = luggageCapacity;
    }

    public String getPNR() {
        return pnr;
    }

    public void setPNR(String pnr) {
        this.pnr = pnr;
    }

    public String getSeat () {
        return seat;
    }
```

```
    public void setSeat(String seat) {
        this.seat= seat;
    }

    public int getLuggageCapacity() {
        return luggageCapacity;
    }

    public void setLuggageCapacity(int luggageCapacity) {
        this.luggageCapacity = luggageCapacity;
    }
}
```

## Flight class

```
import java.util.LinkedList;

public class Flight {

    private String flightCode;
    private String origin;
    private String destination;
    private String aircraftType;
    private int numberOfSeats; // number of available seats

    public Flight(String flightCode, String origin, String destination,
    String aircraftType, int numberOfSeats) {
        this.flightCode = flightCode;
        this.origin = origin;
        this.destination = destination;
        this.aircraftType = aircraftType;
        this.numberOfSeats = numberOfSeats;
    }

    public String getFlightCode() {
        return flightCode;
    }

    public void setFlightCode(String flightCode) {
        this.flightCode = flightCode;
    }

    public String getOrigin() {
        return origin;
    }

    public void setOrigin(String origin) {
        this.origin = origin;
    }

    public String getDestination() {
        return destination;
    }

    public void setDestination(String destination) {
        this.destination = destination;
    }

    public String getAircraftType() {
        return aircraftType;
```

```
    }

    public void setAircraftType(String aircraftType) {
        this.aircraftType = aircraftType;
    }

    public int getNumberOfSeats() {
        return numberOfSeats;
    }

    public void setNumberOfSeats(int numberOfSeats) {
        this.numberOfSeats = numberOfSeats;
    }

}
```