# 48024 Programming 2 Assignment 1 Guide

## Aim:

This guide is intended to help students to get started on the assignment. You may skip to the section that you need help with.

The assignment is designed to be completed week by week.

The first message is that there are no tricks in the assignment. If you apply the processes and patterns you have learnt, you should have little trouble completing at least the passing requirements of the assignment.

## Weekend 5

At the end of Week 5, you should be able to complete the following:

1. Declare the constructors – you may need to partially complete constructors that initialize Lists.
2. Declare the toString() functions.
3. Implement the main menu.

The Agency class exists as the entry point of the program, it also contains the main menu of the program which should delegate tasks down to the respective classes. For example, if the user chooses the “Explore Flights” option, then your code should access the respective menu inside the Flights class, which manages all things flight-related.

Everything you need to know to complete the above steps is covered in the Study 4 module, and the tutor demo for Lab 5.

If you are not sure how to do some of the tasks above, it is recommended that you open sample code as a reference.

The sample code contains examples of everything above. There is sample code included in the Study 4 module download link (code-classes.jar). You can also watch the tutor demonstration video in Demo 5.

***Classes and fields***

To be able to successfully create a Trip, you will first need to develop the classes it relies on. A Trip has a collection of destinations (abstracted to the Destinations class) and a collection of flights (abstracted to the Flights class); both of which rely on the singular version of that class (Destination and Flight, respectively). of that class (Destination and Flight, respectively). It is highly recommended you work from the bottom up. For example, developing the Flight and Destination class, followed by the Flights and Destinations class, and then working on the Trip class.

**Destination**

**Constructor**: Takes as parameters the destination name as a String and the name of the country as a String. Should initialize the matching member variables.

**toString():** Should print out the information of the flight in the following format:

{destination name} in {country}

Example: Eiffel Tower in France

**Flight**

**Constructor**: Takes as parameters the airline name as a String, the flight number as an integer, the takeoff country as a String, the landing country as a String and the flight cost as a double. Should initialize the matching member variables.

**toString():** Should print out the information of the flight in the following format:

{airline} Flight {flight number} from {takeoff country} to {landing country} for the price of ${cost of the flight}

Example: American Airlines Flight 677 from Argentina to Spain for the price of $260.44

You should be familiar with how to write constructors and toString() functions from Week 5. While you may not be familiar with Lists just yet, they can be temporarily ignored. If you don't remember the Constructor patterns, you should consult your Patterns Book (you wrote them down right?).

**The Main Menu**

This is the menu pattern. This is also the first menu that is shown when the user runs your application. Following the standard pattern, you'll put this menu into a method named use(). Also following the standard pattern, you'll need a main() method to start your program, and this main method should call the use() method to show the menu.

The point that's sometimes difficult to understand here is that the main() method is a static method, while the use() method is an instance method. Therefore, the main() method cannot simply just call the use() method otherwise you will get an error such as "non-static method cannot be invoked from a static context".

What you need to understand is that because the use() method is an instance method inside Agency, you need to first create an instance of the Agency class, and then you can call the use() method on that instance. This is what the Study 5 example shows you how to do. If you follow the pattern as shown, your main method will create a new instance before invoking the instance method.

Sometime within Weeks 5 and 6 you will need to complete the other menus as well.

## Week 6

In Week 6, you should watch the Study 5 module on lists and complete at least the following features related to lists:

1. Instantiating and initializing the lists in the constructors. Remember that you may choose any class that implements the List interface (you will need to import java.util for most of them). I personally recommend ArrayList, but for this assignment, you shouldn't notice any difference.

2. Add a Destination to Destinations. – note that you must also keep the combination of country and name unique. You will need to check that a destination does not already exist with that combination later.

3. Add a Flight to Flights – note that you must also keep the combination of takeoff and landing unique. Your will need to check that a flight does not already exist with that combination later.

4. Remove a Destination from Destinations.

5. Remove a Flight from Flights.

These goals are moderately challenging to implement. Make sure you plan carefully before diving into the coding – it may be helpful to review the processes you know for designing OO applications.

### Agency

**Constructor**: No parameters. Should initialize the member variables.

**login():** Prompt the user for a username and password and cross check the input with valid credentials. This should be done by lookup pattern for checking every administrator in Administrators for matching credentials.

**main():** Main method for the program.

### Destinations

**Constructor**: Takes an instance of Agency as a parameter. This should be passed down from the Agency class that called the constructor. Should initialize the destinations list.

**use():** Main menu for the “Explore Destinations” option.

**Add Destination:** Prompts the user for all necessary fields to create a new Destination object and adds it to the destinations list.

**Remove Destination:** Prompts the user for all necessary fields to identify a destination from the destinations list, should it exist.

Check required: The destinations must exist inside the flights list. If input is wrong, ask the user to re-enter the values

**destination (String name, String country):** returns the destination object matching that combination of name and country, should it exist.

**display by country:** Prompts the user for a country. This method should do the same as display() but only list destinations that are located in the same country as the inputted country.

**display():** Prints all destinations by calling their toString() function, one per line. This should include the destination header and footer, found in the Utils class.

### Administrators

**Constructor**: Should Initialize the administrators list.

You can input sample Administrator objects into the administrators list.

**Flights**

**Constructor**: Takes an instance of Agency as a parameter. This should be passed down from the Agency class that called the constructor. Should initialize the flights list.

**use():** Main menu for the “Explore Flights” option.

**Add Flight:** Prompts the user for all necessary fields to create a new Flight object and adds it to the flights list.

Check required: The combination of takeoff and landing must be unique. Your code must check that a flight does not already exist with that combination. If the input is wrong, ask the user to re-enter the values.

**Remove Flight:** Prompts the user for all necessary fields to identify a flight from the flights list, should it exist.

Check required: The flight must exist inside the flights list. If the input is wrong, ask the user to re-enter the values.

**flight(String takeoff, String landing):** returns the flight object matching that combination of takeoff and landing, should it exist.

**display():** Prints all flights by calling their toString() function, one per line. This should include the flight header and footer, found in the Utils class.

## Week7

If you remain on track, you should be starting to move to the more advanced List manipulation:

1. Book a trip.
2. Displaying the Trip.
3. Add connecting flights.
4. Handling Non-existing and Existing check.

**Trip**

**Constructor**: Takes an instance of Agency as a parameter. This should be passed down from the Agency class that called the constructor. Should initialize the agency, the flights list, and the destinations list.

**use():** Main menu for the “Book a Trip” option.

**addConnectingFlights():** This method assumes a finalized list of destinations. For any 2 consecutive destinations, your code must find a flight ***from the agency’s flight list*** that connects the destination countries and add the flight to the Trip’s flight list. Your code must do this for all destinations in the Trip’s destinations list.

Checks required: If the flights list is already populated, clear that list before repopulating it.

Checks required: If there are less than 2 destinations in the destinations list, inform the user and return to the Trip main menu.

**display():** Prints all the destinations and flights related to the trip, line by line. Provided you did your addConnectingFlights() method properly, the “chronological order” part is done for you. You can print a destination and a flight at the same index in their respective arrays. Keep in mind the size of the arrays will be different.

## StuVac

If you have kept up, you should be close to fully complete the assignment. If you are attempting this, you should be at the point where manipulating Lists no longer poses difficulty - your brain power here will be needed to consider how to get the information out reliably.