Please fill in your name and student ID in the table below.	
Student Name	*Bibek Sapkota & Solomon Silwal*
Student Number	23189618 & 23189650
Course and Year	BSc Computer Science 2022/23
Module Code	CMP5332
Module Title	Object Oriented Programming
Module Leader	Dr Abdel-Rahman Tawil
Assessment item:	Flight Booking System

Students are required to complete this software implementation checklist for their Flight Booking System developed in Java. You need to select only the features that were implemented in your code. You can select all/some features from any marking range as long as they have been implemented in the code submitted for this assessment. For example, you can select all/some features from the marking range 40%-49% and all/some features under the range 70%-79% etc. This implementation checklist should be submitted with the PowerPoint documentation for this coursework.

Important notice: This checklist will assist the tutors when marking your code, hence, you should only select the feature requirements that are implemented in your code. Even if some features are not working correctly, you can still select them as long as there is evidence in your code showing the implementation attempt. However, it is not acceptable for a student to claim the implementation of features that were not attempted/implemented in the code. <u>False claims are a clear indication that the student does not understand the submitted code, hence, the submission will be investigated further for plagiarism</u>, and the tutor marking the assessment may invite the student to explain all/parts of the submitted code.

Checklist Interactive Library System

Achieving a mark of 40% to maximum of 49%	
• Add new customers (passengers) to the system. System should store at least the following information for each customer: ID, Name, Phone Number and List of Bookings made.	☒ Yes
• List all customers stored in the system.	☒
• Issue bookings for customers. When a booking is issued for a customer, a Booking object must be created holding a reference to the outbound and return flights booked and to the customer that made the booking. This object should be added to the customer list of flight bookings. In addition, the Customer object should be added to the list of passengers in the Flight object.	☒
• Cancel bookings. A customer can decide to cancel a booking. The status of the booking should be updated to reflect cancelation and also the flight	☒

object should be updated to reduce the number of passengers for that particular flight.	
• Display details for a particular customer including details for bookings (flight number, origin, destination, date and price) they have made [showcustomer command].	☒
• Display details for a particular flight including details for passengers (name, phone number) [showflight command].	☒
• Save the status of the system to the backend storage (i.e. text file storage) when the system is closed. The flight booking system data should be stored in three different files (flights.txt, customers.txt and bookings.txt). A sample format to save the different properties for each object is given in the **Sample Prototype Application** section above. When the system starts it should load the status of the booking system from the text files to the memory.	☒
Achieving a mark of 50% to maximum of 59%	
• Add a ‘number of seats’ (capacity) property and a price property to the Flight class and make the appropriate changes to the program to ensure that this information can be captured when a new flight is created. Also ensure that this information will be stored to and correctly loaded from the file storage.	☒
• Add an email property to the Customer class and make the appropriate changes to the program to ensure that this information can be captured when a new customer is created. Also ensure that this information will be stored to and correctly loaded from the file storage.	☒
• Implement Unit Tests to validate and demonstrate that the above changes made to the Flight and Customer classes work as expected.	☒
Achieving a mark of 60% to maximum of 69%	
• Extend the prototype implementation for the GUI application provided to allow for the following basic functionalities: ○ Display a popup window that will show the list of customers (passengers) for a particular flight. ○ List all customers and their details including the number of bookings they made. ○ Display a popup window that will show the Booking details if a customer is selected from the above created list. ○ Display a popup window when the “Add” submenu of the “Customer” menu item is selected. The popup should display a form that allows the addition of a new customer to the system.	☒ ☒ ☒ ☒
• Extend the functionality of the flight system to allow for storing data to the file storage after the execution of commands that change the state of the system (e.g. “addFlight”, “updateBooking”). If the system fails to store the data on the file storage due to an error (e.g. file is already in use or corrupted), the program must inform the user and rollback any changes made to the system	☒

prior to the error. *Hint: You can change the file permission to "read-only" in order to test this functionality.*	
Achieving a mark of 70% to maximum of 79%	
• Remove (hide) existing flights from the system. When a flight is removed, it should not appear in the flight list view. Instead of completely deleting a flight, use a Boolean property in the Flight class to indicate whether a flight is deleted. Change the affected functions appropriately to return only the flights that are not deleted.	☒
• Remove (hide) existing customers from the system. When a customer is removed, it should not appear in the customers list view. Instead of completely deleting a customer, use a Boolean property in the Customer class to indicate that the customer is deleted. Change the affected functions appropriately to return only the customers that are not deleted.	☒
• Impose a limit on the maximum number of passengers that can be added to a flight using the capacity property of the Flight class. The system should not allow to make a booking for a flight that is in full capacity and a message should be displayed to the user.	☒
• Extend the implementation for the GUI application to add a Delete functionality for both flights and customers using the GUI.	☒
• Add Javadoc documentation only to the new methods created as part of this marking band.	☒
Achieving a mark of 80% and over	
• List only flights that are in the future and have not departed. To implement this functionality you need to use the systemDate to indicate whether a flight has departed and a booking is completed. You have the flexibility to decide on how to complete the implementation of this functionality.	☒
• Impose a cancellation/rebook fee when customers cancel/update their booking. The cancellation/rebook fee must be shown in the bookings in addition to the flights price.	☐
• Calculate a different (increased) price for each flight, based on how many days left for the flight to depart on the day of the booking (current systemDate) and the capacity (number of seats left) for the flight. This price should be displayed when listing all flights before a booking is made. In addition, after a booking is made, this price should be stored and displayed when showing the booking details for a customer. This means that two bookings for the same flight, made by different customers on different dates would display different prices.	☒
• Complete the implementation for the GUI application provided to allow for the following functionalities: o Implement the issue booking functionality to book a flight using the GUI. o Implement the update booking functionality to allow for updating a booking using the GUI. o Implement the cancel booking functionality to cancel a booking using the GUI.	 ☒ ☒ ☒