FLIGHT BOOKING SYSTEM

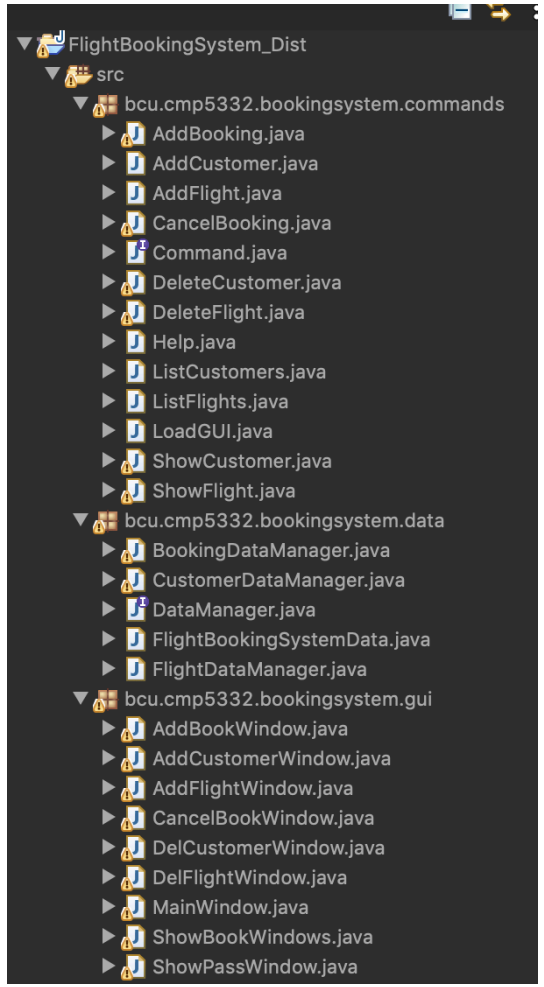
JASPREET SINGH & ROHEEL AHMED

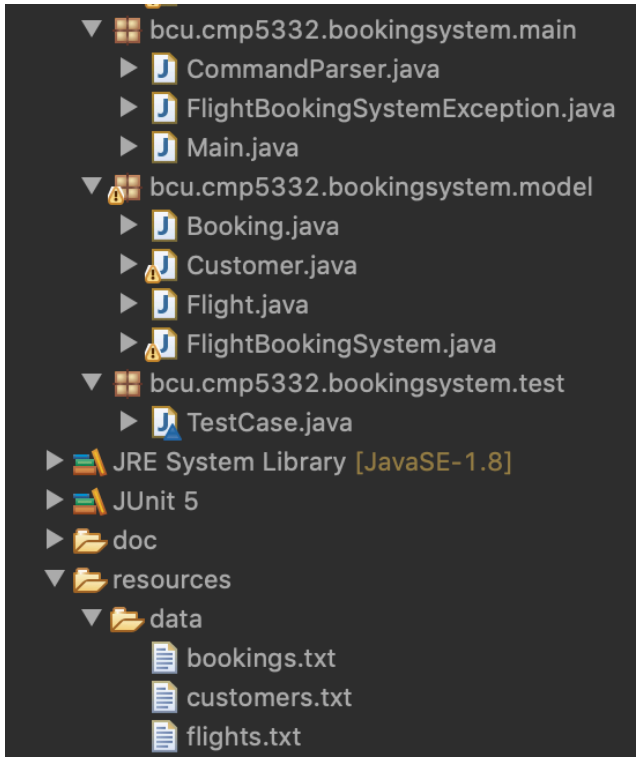
BIRMINGHAM CITY UNIVERSITY

CMP5332-A-S1

2020/2021

FOLDER STRUCTURE AND FILES





- Packgages organized in base of their own type or functionality in the program itself.
- Creation of various classes to implement functionalities.
- Slightly changes to the classes given at start.
- Creation of new ”Test” package.
- Creation of Java-Doc

ADD CUSTOMERS CODE

Here it's the code to add a customer to the system with all it's various information and with the print show option we can see all the information we put in about the customer itself as ID, name, phone No, mail and number of bookings as well as which these are and when they were done

```
> showcustomer 1
Customer #1
Name: jaspreet
Phone: 012345678
Email: jaspreet@gmail.com
----------------
3 booking(s)
*Booking date: 2021-01-15 fo
*Booking date: 2021-01-15 fo
*Booking date: 2021-01-15 fo
>
```

```java
public class AddCustomer implements Command {
    private final String name;
    private final String phone;
    private final String email;
    private boolean isDeleted = false;
    /**
     * AddCustomer constructor
     *
     * @param name of the customer (string)
     * @param phone of the customer (string)
     * @param email of the customer (string)
     */
    public AddCustomer(String name, String phone, String email) {
        this.name = name;
        this.phone = phone;
        this.email = email;
    }
```

```java
@Override
public void execute(FlightBookingSystem flightBookingSystem) throws FlightBookingSystemException {
    //Give new customer the next ID
    int maxId = 0;
    if (flightBookingSystem.getCustomers().size() > 0) {
        int lastIndex = flightBookingSystem.getCustomers().size() - 1;
        maxId = flightBookingSystem.getCustomers().get(lastIndex).getId();
    }
    //Create customer
    Customer customer = new Customer(++maxId, name, phone,email,isDeleted);
    //Add customer to system
    flightBookingSystem.addCustomer(customer);
    System.out.println("Customer #" + customer.getId() + " added.");

    try {
        //Save the status of the system
        FlightBookingSystemData.store(flightBookingSystem);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

LIST CUSTOMER CODE

Here it prints all the customers of the system with their # as CustomerID and name other than print all the customers present in the system, even those who have been deleted/hided

```
> listcustomers
Customer #1 - jaspreet
Customer #2 - roheel
Customer #3 - joe
Customer #5 - amy
Customer #6 - rocky
6 customers(s)>
```

```java
@Override
public void execute(FlightBookingSystem flightBookingSystem) throws FlightBookingSystemException {
    //Get customer list
        List<Customer> customers = flightBookingSystem.getCustomers();
        //
        for (Customer customer : customers) {
            if(customer.getisDeleted() == false) {
                System.out.println(customer.getDetailsShort());
            }
        }
        System.out.print(customers.size() + " customers(s)");
    }
```

ISSUE AND CANCEL BOOKINGS

Both methods use the same data to complete the command while the add uses them to create the booking while adding a localdate data too the delete use them to locate the exact booking to delete

```
private int customerID;
private int flightID;
```

```
this.customerID = customerID;
this.flightID = flightID;
```

```
@Override
public void execute(FlightBookingSystem flightBookingSystem) throws FlightBookingSystemException {

    //Get flight using ID
    Flight flight = FlightBookingSystem.getFlightByID(flightID);
    //Check if flight full
    if(flight.getNumberPassengers() < flight.getFlightCapacity()) {
        //Get customer using ID
        Customer customer = FlightBookingSystem.getCustomerByID(customerID);
        //Get local date
        LocalDate bookingDate = flightBookingSystem.getSystemDate();
        //create new booking
        Booking booking = new Booking(customer.getId(), flight.getId(), bookingDate);
        //Add the booking to the customer's info
        customer.addBooking(booking);
        //Add customer to the flight's passengers
        flight.addPassenger(customer);
        System.out.println(customer.getName()+" is now passenger of flight "+flight.getFlightNumber());
        System.out.println(" Number of passengers: "+flight.getNumberPassengers());
        System.out.println(" Max capacity: "+flight.getFlightCapacity());
    }else {
        throw new FlightBookingSystemException("Sorry, flight "+flight.getFlightNumber()+" is full!");
    }
    try {
        //Save the status of the system
        FlightBookingSystemData.store(flightBookingSystem);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
@Override
public void execute(FlightBookingSystem flightBookingSystem) throws FlightBookingSystemException {
    //Get customer using ID
    Customer customer = flightBookingSystem.getCustomerByID(customerID);
    //Get flight by ID
    Flight flight = FlightBookingSystem.getFlightByID(flightID);
    //Get customer's bookings
    List<Booking> bookingList = customer.getBookings();
    //Select Booking to delete
    for(Booking booking: bookingList) {
        //If found
        if(booking.getFlightId()==flightID) {
            //Cancel booking
            customer.cancelBooking(booking);
            //Cancel customer from passengers
            flight.cancelPassenger(customer);
            System.out.println(customer.getName()+
                    " is not longer passenger of flight "+flight.getFlightNumber());
            deleted = true;
            break;
        }
    }
    //If not found
    if (deleted == false) {
        throw new FlightBookingSystemException("This booking does not exist!!!");
    }
    try {
        //Save the status of the system
        FlightBookingSystemData.store(flightBookingSystem);
    } catch (IOException e) {
        e.printStackTrace();
        }
    }
```

SAVE DATA

```java
@Override
public void storeData(FlightBookingSystem fbs) throws IOException {
    // TODO: implementation here
    try (PrintWriter out = new PrintWriter(new FileWriter(RESOURCE))) {
        for (Customer customer : fbs.getCustomers()) {
            out.print(customer.getId() + SEPARATOR);
            out.print(customer.getName() + SEPARATOR);
            out.print(customer.getPhone() + SEPARATOR);
            out.print(customer.getEmail() + SEPARATOR);
            out.print(customer.getisDeleted() + SEPARATOR);
            out.println();
        }
    }
}
```

These store data scripts save the various data about the customers, the flights and the bookings and save them into their own txt file

```java
@Override
public void storeData(FlightBookingSystem fbs) throws IOException {
    try (PrintWriter out = new PrintWriter(new FileWriter(RESOURCE))) {
        for (Flight flight : fbs.getFlights()) {
            out.print(flight.getId() + SEPARATOR);
            out.print(flight.getFlightNumber() + SEPARATOR);
            out.print(flight.getOrigin() + SEPARATOR);
            out.print(flight.getDestination() + SEPARATOR);
            out.print(flight.getDepartureDate() + SEPARATOR);
            out.print(flight.getFlightCapacity() + SEPARATOR);
            out.print(flight.getTicketPrice() + SEPARATOR);
            out.print(flight.getisDeleted() + SEPARATOR);
            out.println();
        }
    }
}
```

```java
@Override
public void storeData(FlightBookingSystem fbs) throws IOException {
    // TODO: implementation here
    try (PrintWriter out = new PrintWriter(new FileWriter(RESOURCE))) {
        for (Customer customer : fbs.getCustomers()) {
            for(Booking booking: customer.getBookings()) {
                out.print(booking.getCustomerId() + SEPARATOR);
                out.print(booking.getFlightId() + SEPARATOR);
                out.print(booking.getBookingDate() + SEPARATOR);
                out.print(booking.getBookingPrice() + SEPARATOR);
                out.println();
            }
        }
    }
}
```

LOAD DATA

```java
@Override
public void loadData(FlightBookingSystem fbs) throws IOException, FlightBookingSystemException {
    // TODO: implementation here
    try (Scanner sc = new Scanner(new File(RESOURCE))) {
        int line_idx = 1;
        while (sc.hasNextLine()) {
            String line = sc.nextLine();
            String[] properties = line.split(SEPARATOR, -1);
            try {
                int id = Integer.parseInt(properties[0]);
                String name = properties[1];
                String phone = properties[2];
                String email = properties[3];
                Boolean isDeleted = Boolean.parseBoolean(properties[4]);
                Customer customer = new Customer(id, name, phone,email,isDeleted);
                fbs.addCustomer(customer);
            } catch (NumberFormatException ex) {
                throw new FlightBookingSystemException("Unable to parse book id "
                        + properties[0] + " on line " + line_idx
                    + "\nError: " + ex);
            }
            line_idx++;
        }
    }
}
```

```java
@Override
public void loadData(FlightBookingSystem fbs) throws IOException, FlightBookingSystemException {
    try (Scanner sc = new Scanner(new File(RESOURCE))) {
        int line_idx = 1;
        while (sc.hasNextLine()) {
            String line = sc.nextLine();
            String[] properties = line.split(SEPARATOR, -1);
            try {
                int id = Integer.parseInt(properties[0]);
                String flightNumber = properties[1];
                String origin = properties[2];
                String destination = properties[3];
                LocalDate departureDate = LocalDate.parse(properties[4]);
                int capacity = Integer.parseInt(properties[5]);
                double ticketPrice = Double.parseDouble(properties[6]);
                boolean isDeleted = Boolean.parseBoolean(properties[7]);
                Flight flight = new Flight(id, flightNumber, origin, destination, departureDate, capacity, ticketPrice, isDeleted);
                fbs.addFlight(flight);
            } catch (NumberFormatException ex) {
                throw new FlightBookingSystemException("Unable to parse book id " + properties[0] + " on line " + line_idx
                    + "\nError: " + ex);
            }
            line_idx++;
        }
    }
}
```

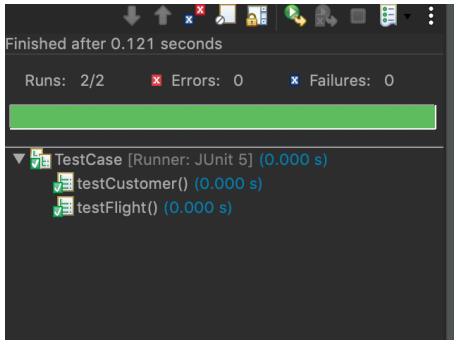
These Load data scripts load the various data about the customers, the flights and the bookings and read their information from their own txt file into the system

```java
@Override
public void loadData(FlightBookingSystem fbs) throws IOException, FlightBookingSystemException {
    // TODO: implementation here
    try (Scanner sc = new Scanner(new File(RESOURCE))) {
        int line_idx = 1;
        while (sc.hasNextLine()) {
            String line = sc.nextLine();
            String[] properties = line.split(SEPARATOR, -1);
            try {
                int customerID = Integer.parseInt(properties[0]);
                int flightID = Integer.parseInt(properties[1]);;
                LocalDate bookingDate = LocalDate.parse(properties[2]);
                double bookingPrice = Double.parseDouble((properties[3]));
                Booking booking = new Booking(customerID,flightID,bookingDate);
                booking.setBookingPrice(bookingPrice);
                Customer customer = fbs.getCustomerByID(customerID);
                Flight flight = FlightBookingSystem.getFlightByID(flightID);
                customer.addBooking(booking);
                flight.addPassenger(customer);
            } catch (NumberFormatException ex) {
                throw new FlightBookingSystemException("Unable to parse book id "
                        + properties[0] + " on line " + line_idx
                    + "\nError: " + ex);
            }
            line_idx++;
        }
    }
}
```

TEST UNIT – MAIL - CAPACITY

```java
public class Flight {
    private int id;
    private String flightNumber;
    private String origin;
    private String destination;
    private LocalDate departureDate;
    private int capacity;
    private double ticketPrice;
    int  numberPassengers = 0;
    private Boolean isDeleted;
    private final Set<Customer> passengers;
    public Flight(int id, String flightNumber, Str
        this.id = id;
        this.flightNumber = flightNumber;
        this.origin = origin;
        this.destination = destination;
        this.departureDate = departureDate;
        this.capacity = capacity;
        this.ticketPrice = ticketPrice;
        passengers = new HashSet<>();
        this.isDeleted = isDeleted;
    }
```

Here the mail and capacity data inserted into the customer and flight istances of the system and checked using a simple test unit



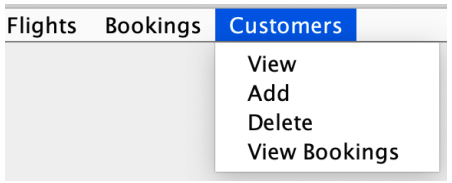

```java
package bcu.cmp5332.bookingsystem.test;

import static org.junit.Assert.assertEquals;

class TestCase {
    private final Flight flight = new Flight(1 , "MM0001" , "Madrid" , "Manchester" , LocalDate.parse("2021-02-26"), 70 , 110.0 ,false);
    private final Customer customer  = new Customer(1 , "Joe" , "019995678" , "Joe@gmail.com" , false);
    Double price  =  110.0;
    @Test
    public void testFlight() throws Exception {
        assertEquals(70, flight.getFlightCapacity());
        assertEquals(price, flight.getTicketPrice());
    }
    @Test
    public void testCustomer() throws Exception {
        assertEquals("Joe@gmail.com" , customer.getEmail());
    }
}
```

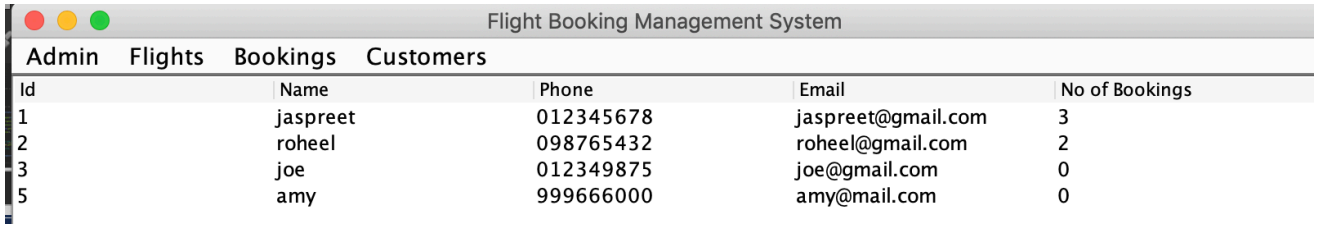
```java
public class Customer {
    private int id;
    private String name;
    private String phone;
    private String email;
    private final List<Booking> bookings;
    private Boolean isDeleted;
    // TODO: implement constructor here
    public Customer (int id,String name, String phone, String email, Boolean isDeleted) {
        this.id = id;
        this.name = name;
        this.phone = phone;
        this.email = email;
        bookings = new ArrayList<>();
        this.isDeleted = isDeleted;
    }
```

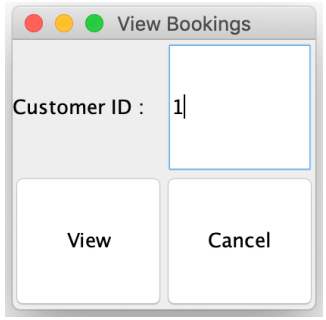
```java
public class AddCustomer implements Command {
    private final String name;
    private final String phone;
    private final String email;
    private boolean isDeleted = false;
    /**
     * AddCustomer constructor
     *
     * @param name of the customer (string)
     * @param phone of the customer (string)
     * @param email of the customer (string)
     */
    public AddCustomer(String name, String phone, String email) {
        this.name = name;
        this.phone = phone;
        this.email = email;
    }
```

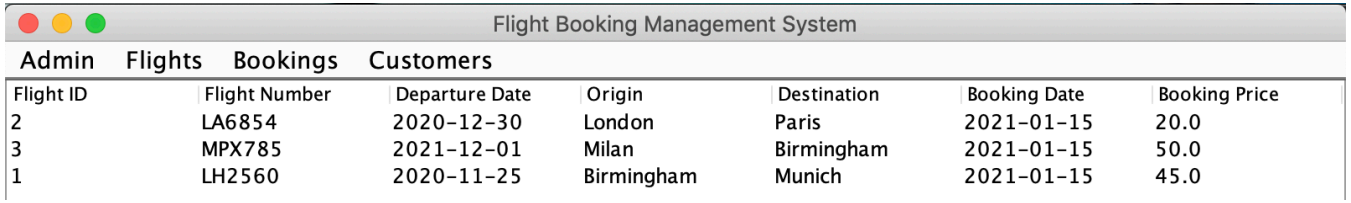
GUI - VIEW BOOKINGS/CUSTOMERS/PASSENGERS LIST

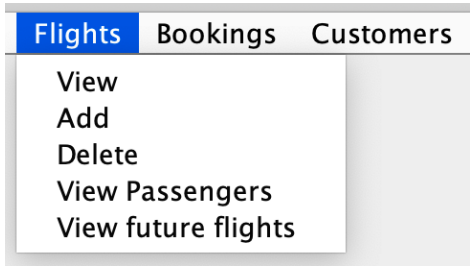
With the various popups menu options it's possible to view the customers list, the flight's passengers and customer's bookings after inserting the flight/customer ID as an input in the window apperating on screen

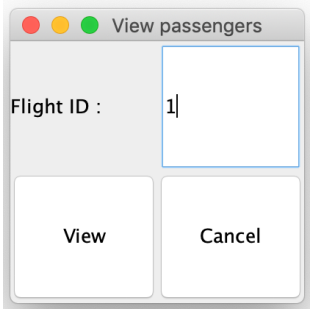


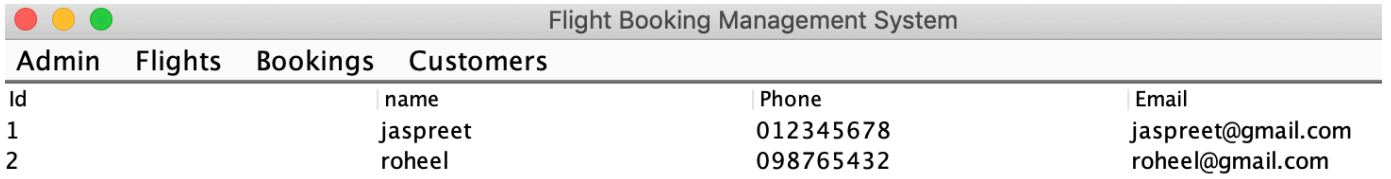




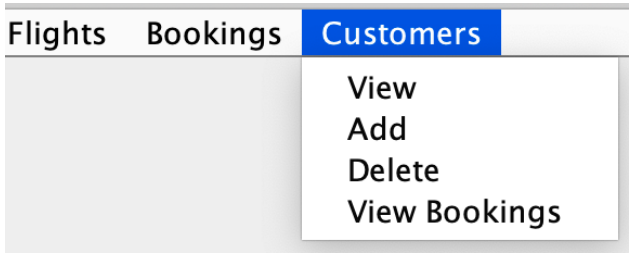


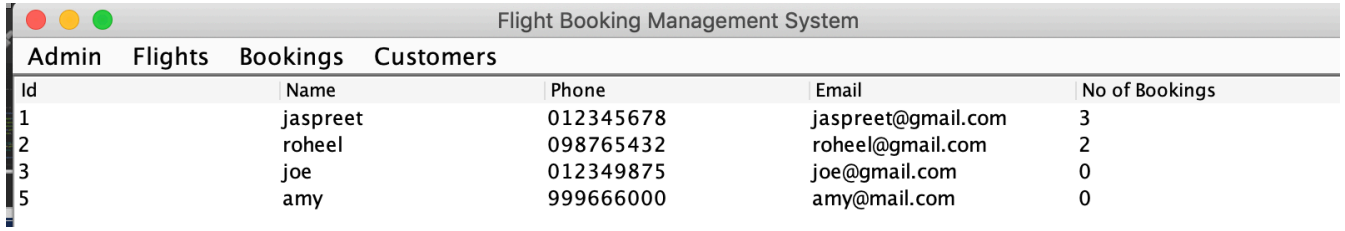




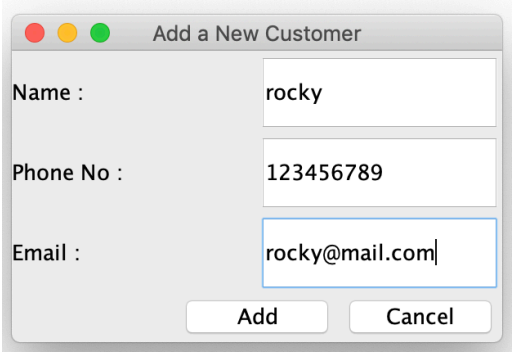


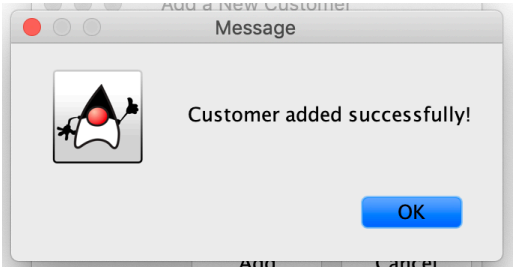
GUI - ADDING NEW CUSTOMER

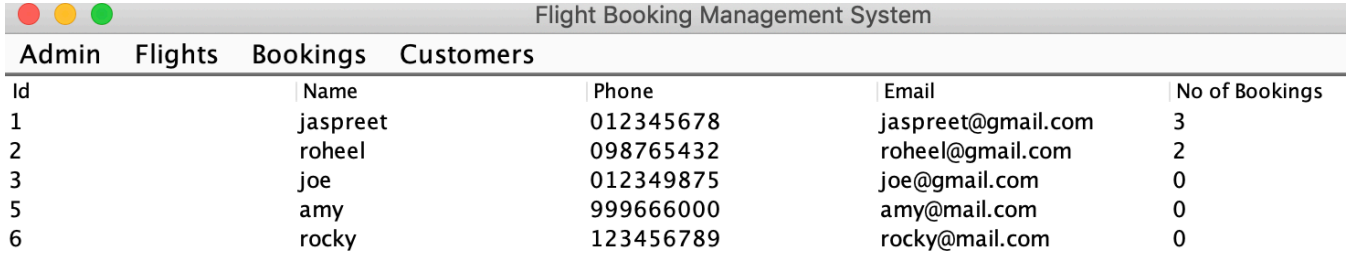




Here using ther popup menu options we can insert a new customer after putting the customer's information into the right textboxes inside the windown that will appear on screen







CODE OF GUI ELEMENTS

```java
//add a new customer element
private void addCustomer() {
    try {
        String name = nameText.getText();
        String phone = phoneNoText.getText();
        String email = emailText.getText();

        // create and execute the AddCustomer Command
        Command addCustomer = new AddCustomer(name, phone, email);
        addCustomer.execute(mw.getFlightBookingSystem());
        JOptionPane.showMessageDialog(this, "Customer added successfully!");
        // refresh the list of Customers
        mw.displayCustomers();
        // close the AddCustomerWindow
        this.setVisible(false);
    } catch (FlightBookingSystemException ex) {
        JOptionPane.showMessageDialog(this, ex, "Error", JOptionPane.ERROR_MESSAGE);
    }
}
```

```java
//find and view the passengers
private void viewPass() throws FlightBookingSystemException {
    try {
        int flightId = Integer.parseInt(flightID.getText());
        Flight flight = FlightBookingSystem.getFlightByID(flightId);
        if(!flight.getisDeleted()) {
        mw.displayPassengers(flight);
        }else {
            JOptionPane.showMessageDialog(this, "This Flight has been deleted from the system!");
        }
        this.setVisible(false);
    } catch (FlightBookingSystemException ex) {
        JOptionPane.showMessageDialog(this, ex, "Error", JOptionPane.ERROR_MESSAGE);
    }

}
```

```java
// displayBookings to display bookings in table
public void displayBookings(Customer customer) throws FlightBookingSystemException {
    List<Booking> bookingList = customer.getBookings();
    // headers for the table
    String[] columns = new String[]{"Flight ID", "Flight Number",
            "Departure Date", "Origin", "Destination",
            "Booking Date", "Booking Price"};

    Object[][] data = new Object[bookingList.size()][7];
    for (int i = 0; i < bookingList.size(); i++) {
        Booking booking = bookingList.get(i);
        Flight flight = FlightBookingSystem.getFlightByID(booking.getFlightId());
        data[i][0] = booking.getFlightId();
        data[i][1] = flight.getFlightNumber();
        data[i][2] = flight.getDepartureDate();
        data[i][3] = flight.getOrigin();
        data[i][4] = flight.getDestination();
        data[i][5] = booking.getBookingDate();
        data[i][6] = booking.getBookingPrice();

    }

    JTable table = new JTable(data, columns);
    this.getContentPane().removeAll();
    this.getContentPane().add(new JScrollPane(table));
    this.revalidate();
}
```

```java
// displayPassengers to display Passengers in table
public void displayPassengers(Flight flight) {
    List<Customer> passengerList = flight.getPassengers();
    String[] columns = new String[]{"Id", "name", "Phone", "Email"};
    Object[][] data = new Object[passengerList.size()][4];
    int j = 0;
    for (int i = 0; i < passengerList.size(); i++) {
        Customer passenger = passengerList.get(i);
        if(!passenger.getisDeleted()) {
            data[j][0] = passenger.getId();
            data[j][1] = passenger.getName();
            data[j][2] = passenger.getPhone();
            data[j][3] = passenger.getEmail();
            j++;
        }
    }

    JTable table = new JTable(data, columns);
    this.getContentPane().removeAll();
    this.getContentPane().add(new JScrollPane(table));
    this.revalidate();
}
```

```java
public void displayCustomers() {
    //Get customers from the system
    List<Customer> customersList = fbs.getCustomers();
    String[] columns = new String[]{"Id","Name", "Phone", "Email", "No of Bookings"};
    Object[][] data = new Object[customersList.size()][5];
    int j = 0;
    for (int i = 0; i < customersList.size(); i++) {
        Customer customer = customersList.get(i);
        //If customer not hidden/deleted
        if(!customer.getisDeleted()) {
            data[j][0] = customer.getId();
            data[j][1] = customer.getName();
            data[j][2] = customer.getPhone();
            data[j][3] = customer.getEmail();
            data[j][4] = customer.getBookSize();
            j++;

        }
    }

    JTable table = new JTable(data, columns);
    this.getContentPane().removeAll();
    this.getContentPane().add(new JScrollPane(table));
    this.revalidate();
}
```

REMOVE (HIDE) CUSTOMER / FLIGHT

```
public DeleteCustomer(int customerID) {
    this.customerID = customerID;
}
```

```
public DeleteFlight(int flightID) {
    this.flightID = flightID;
}
```

```
@Override
public void execute(FlightBookingSystem flightBookingSystem) throws FlightBookingSystemException {
     Customer customer = flightBookingSystem.getCustomerByID(customerID);
     customer.setisDeleted(true);
     System.out.println(customer.getName()+" is not longer a customer of the system.");
     try {
        FlightBookingSystemData.store(flightBookingSystem);
    } catch (IOException e) {
        e.printStackTrace();
        }
    }
```

```
@Override
public void execute(FlightBookingSystem flightBookingSystem) throws FlightBookingSystemException {
     Flight flight = flightBookingSystem.getFlightByID(flightID);
     flight.setisDeleted(true);
     System.out.println(flight.getFlightNumber()+" is not longer a part of the system.");
     try {
        //Save the status of the system
        FlightBookingSystemData.store(flightBookingSystem);
    } catch (IOException e) {
        e.printStackTrace();
        }
    }
```

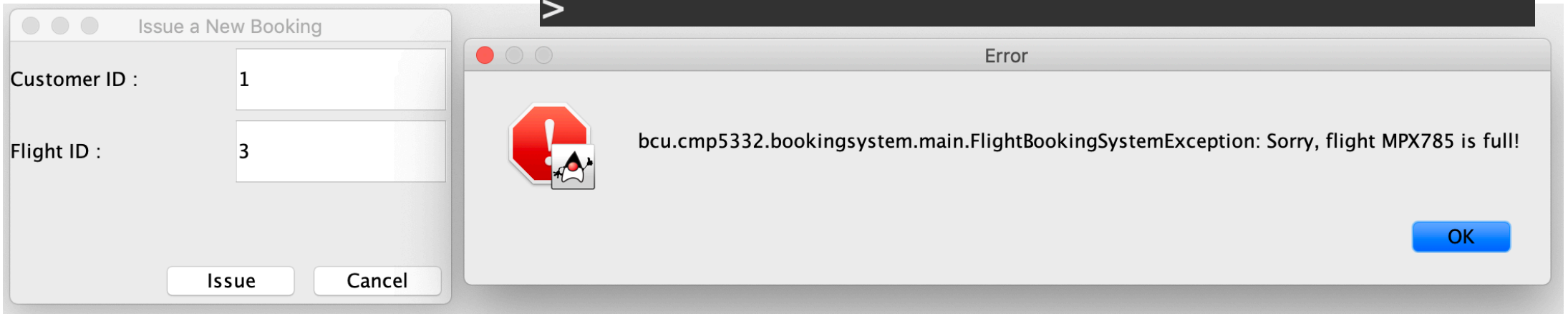
Here the code to delete or hide the customer / flight using a isDeleted boolean variable and setting it to true that will be used to indicate if the element will be listed when printing all the elements

IMPOSE LIMIT OF PASSENGERS ON FLIGHTS

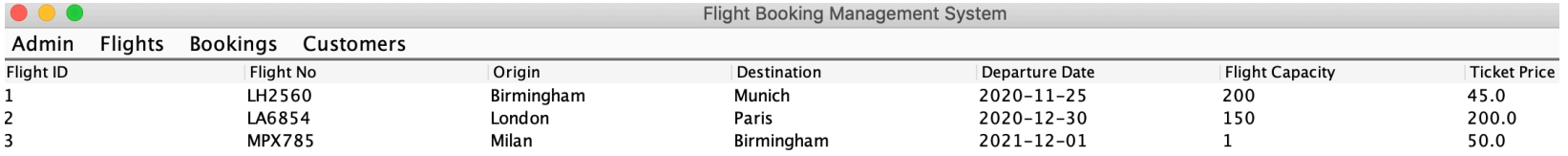
Here the code to impose a max limit on flights as capacity of the flight and when trying to add a new passenger to the flight, the proigram blocks the user to do so

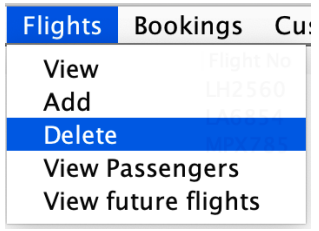
```
//Get flight using ID
Flight flight = FlightBookingSystem.getFlightByID(flightID);
//Check if flight full
if(flight.getNumberPassengers() < flight.getFlightCapacity())
```

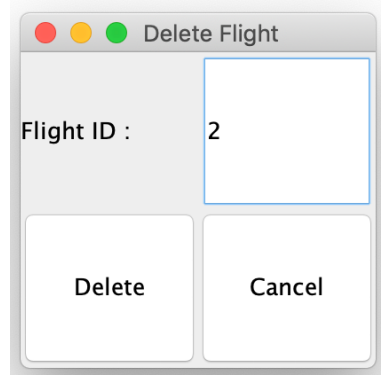
```
> addbooking 1 3
jaspreet is now passenger of flight MPX785
 Number of passengers: 1
 Max capacity: 1
> addbooking 2 3
Sorry, flight MPX785 is full!
>
```

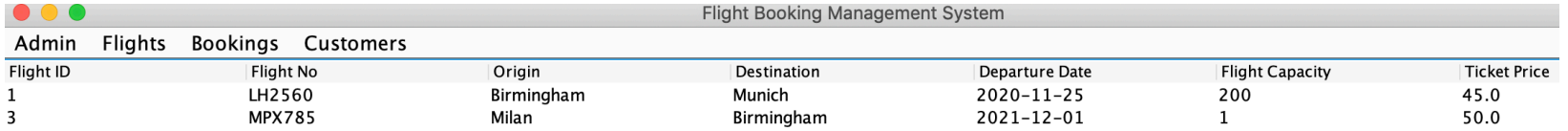


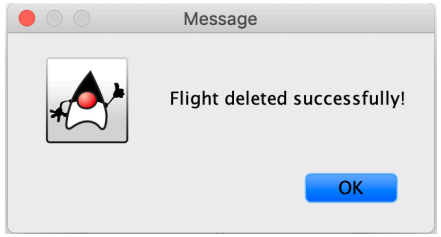
DELETE CUSTOMER / FLIGHT



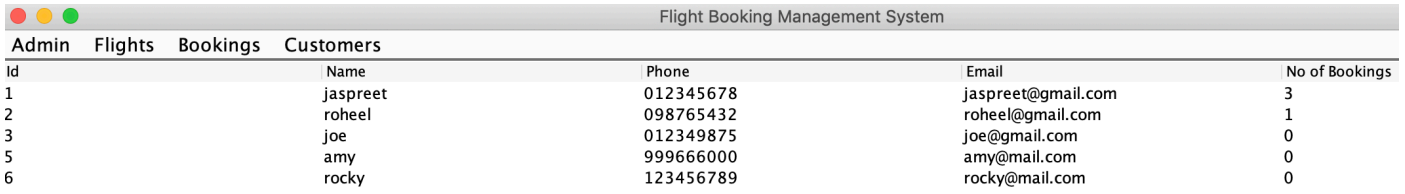


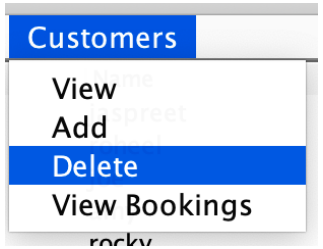


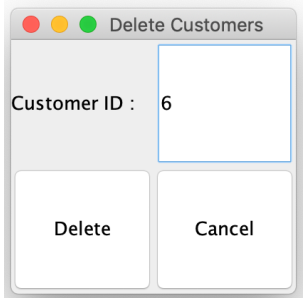


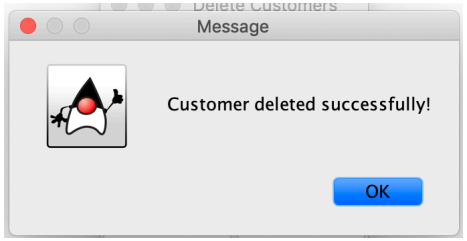


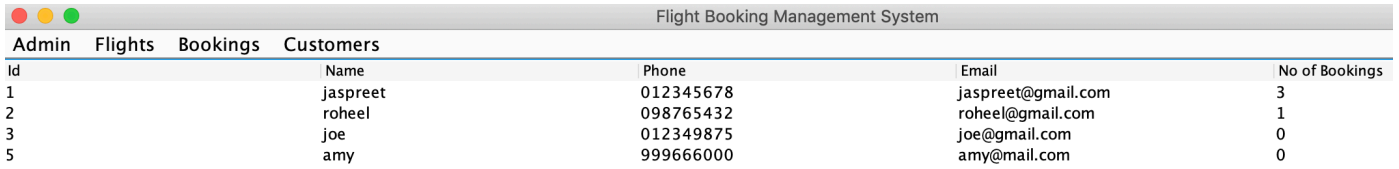
Here the code to delelte the customer and flight implemented in the GUI too





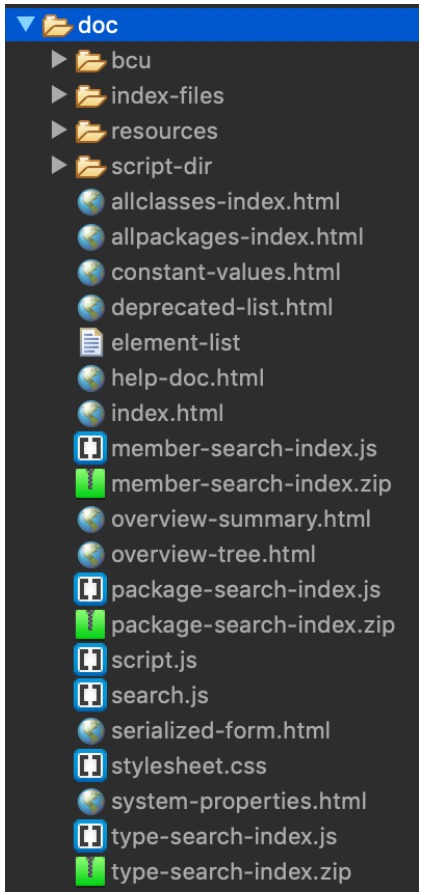


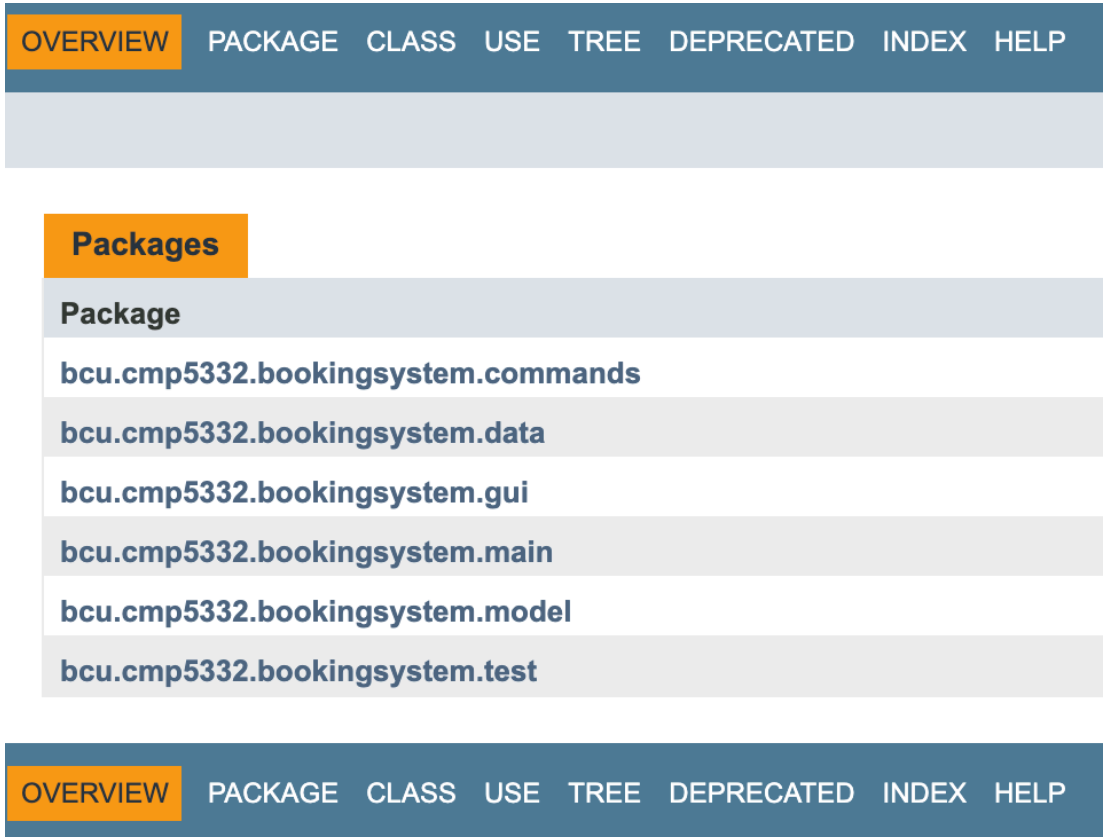


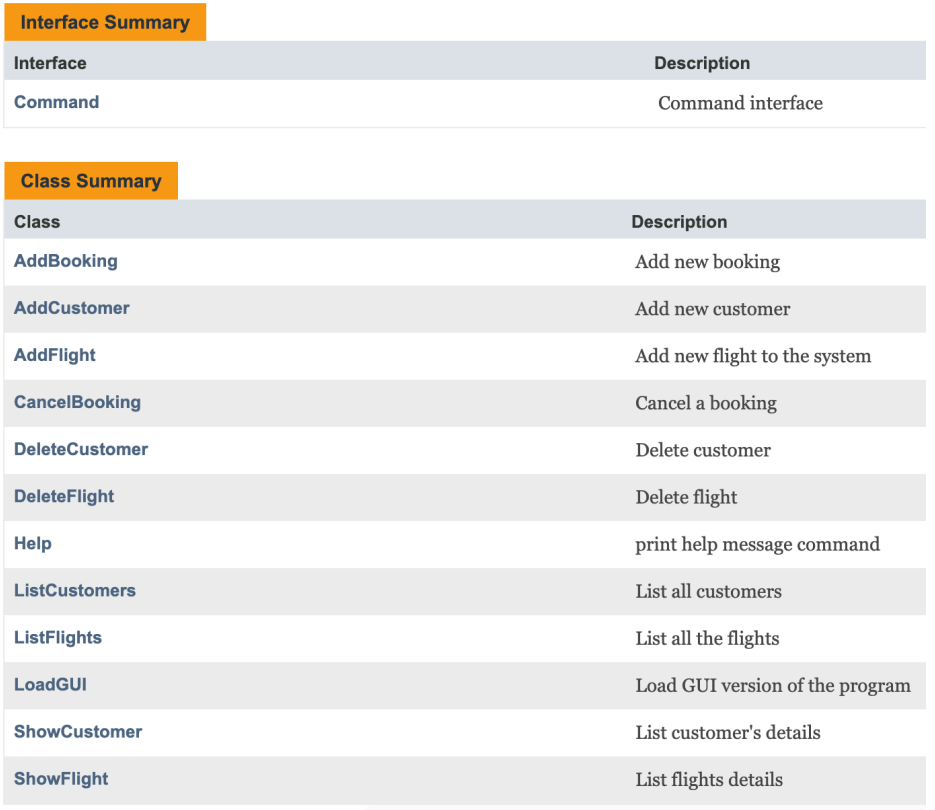


JAVA DOC

Here the creation of the javadoc files with all the classes, methods, etc.







VIEW ONLY FUTURE FLIGHTS

Flight Booking Management System

Admin Flights Bookings Customers

Flight ID	Flight No	Origin	Destination	Departure Date	Flight Capacity	Ticket Price
1	LH2560	Birmingham	Munich	2020-11-25	200	45.0
3	MPX785	Milan	Birmingham	2021-12-01	1	50.0

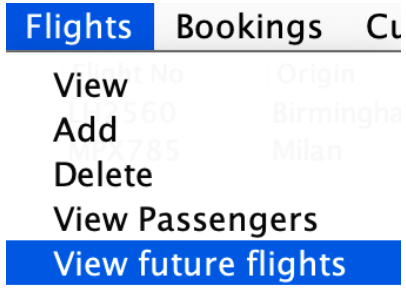
Flight Booking Management System

Admin Flights Bookings Customers

Flight ID	Flight No	Origin	Destination	Departure Date	Flight Capacity	Ticket Price
3	MPX785	Milan	Birmingham	2021-12-01	1	50.0

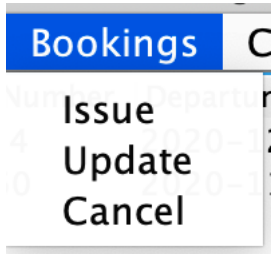
```
if(!flight.getisDeleted()) {
    //If flight departure date is after today
    if(flight.getDepartureDate().isAfter(LocalDate.now())) {
```

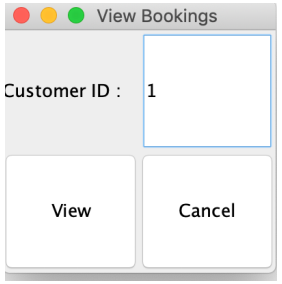
Here the code and GUI to show only the flights that are gonna fly in the future



ADD / CANCEL BOOKINGS USING GUI

Here the GUI implementation to add or delete a customer's booking

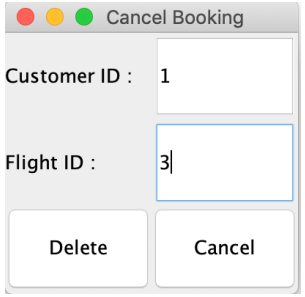


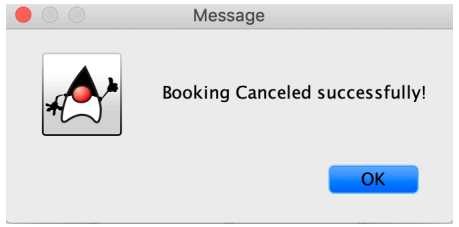


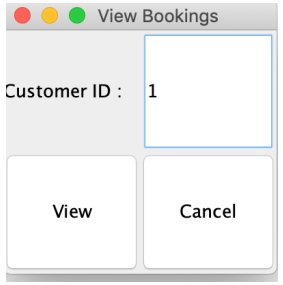
Flight Booking Management System

Admin Flights Bookings Customers

Flight ID	Flight Number	Departure Date	Origin	Destination	Booking Date	Booking Price
2	LA6854	2020-12-30	London	Paris	2021-01-15	20.0
1	LH2560	2020-11-25	Birmingham	Munich	2021-01-15	45.0
3	MPX785	2021-12-01	Milan	Birmingham	2021-01-16	50.0



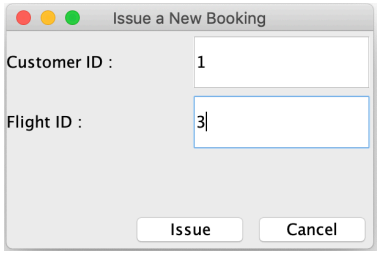


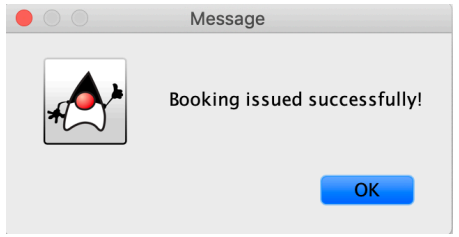


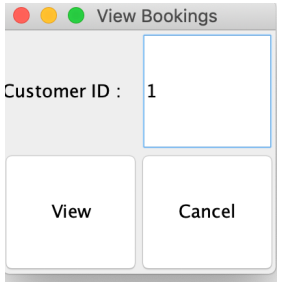
Flight Booking Management System

Admin Flights Bookings Customers

Flight ID	Flight Number	Departure Date	Origin	Destination	Booking Date	Booking Price
2	LA6854	2020-12-30	London	Paris	2021-01-15	20.0
1	LH2560	2020-11-25	Birmingham	Munich	2021-01-15	45.0







Flight Booking Management System

Admin Flights Bookings Customers

Flight ID	Flight Number	Departure Date	Origin	Destination	Booking Date	Booking Price
2	LA6854	2020-12-30	London	Paris	2021-01-15	20.0
1	LH2560	2020-11-25	Birmingham	Munich	2021-01-15	45.0
3	MPX785	2021-12-01	Milan	Birmingham	2021-01-16	50.0

END

JASPREET SINGH & ROHEEL AHMED

BIRMINGHAM CITY UNIVERSITY

CMP5332-A-S1

2020/2021