# **Ryanair**

# **Airline Reservation System**

**Rayan Khales**

**Vanier College**

**420-SF2-RE Data Structures and Object-Oriented Programming**

**May 11, 2025**

**Table of Contents**

## 1. Project Description

The Airline Reservation System (ARS) is a program that helps manage flight bookings for both passengers and admins. Passengers can search for flights, book tickets, and make changes to their bookings, like canceling or updating them. Admins check and confirm bookings, keep track of passenger details, and manage all the information about flights, pilots, crew members, and booking reports.

## 2. Program Features

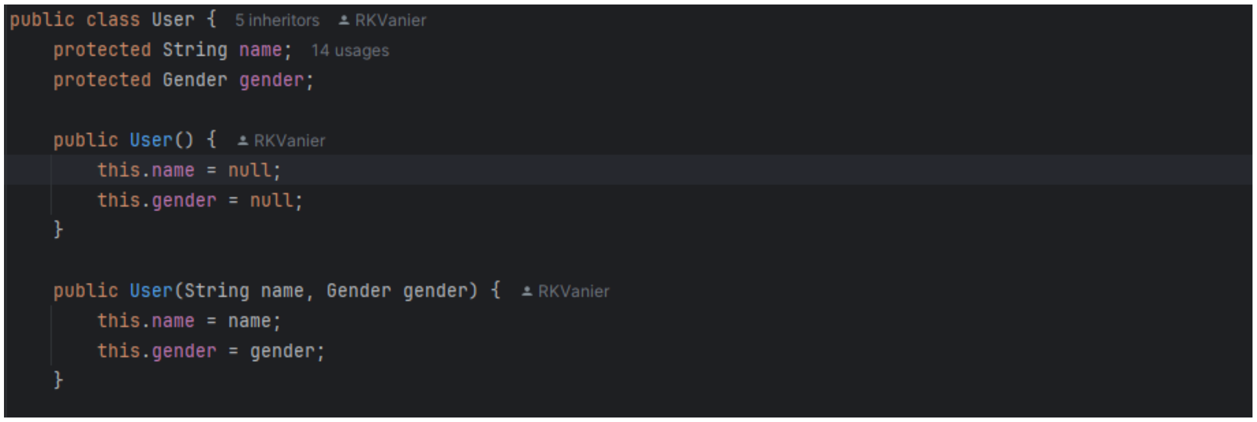
- **User Hierarchy**:
  - User (abstract) → Passenger, Admin, CrewMember
  - CrewMember → Pilot, FlightAttendant
- **Flight Hierarchy**:
  - Flight (abstract) → DomesticFlight, InternationalFlight
- **Booking Management**:
  - Booking requests can be made by Passenger and approved by Admin.
- **Interfaces**:
  - Interface Bookable (with abstract method requestBooking())
- **Comparable and Comparator**:

- CrewMember, Flight Attendent and Pilot implements Comparable
- Custom Comparator<CrewMember> is used to sort by name, hours, etc.

- **Polymorphism**:
  - compareTo() behavior varies in Pilot, FlightAttendant.
- **Text I/O**:
  - Booking information is written to a file (BookingSystem.generateReport()).
- **Unit Testing**:
  - All user-defined methods tested.

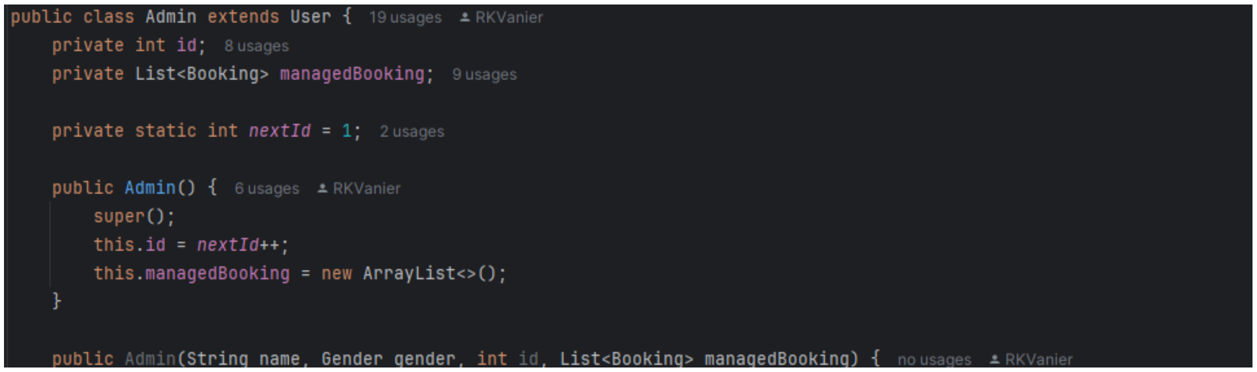
## 3. Screenshots & Execution

The parent class in the hierarchy:

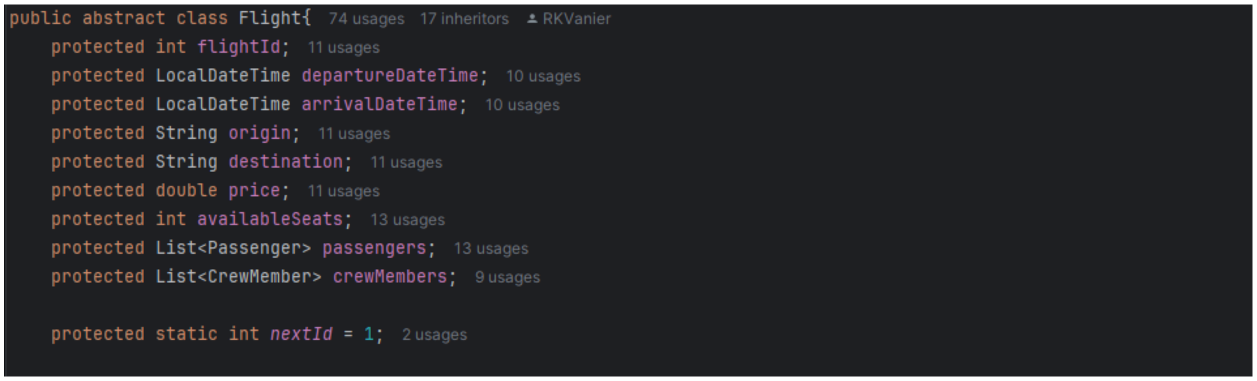


```
public class User {  5 inheritors  RKVanier
    protected String name;  14 usages
    protected Gender gender;

    public User() {  RKVanier
        this.name = null;
        this.gender = null;
    }

    public User(String name, Gender gender) {  RKVanier
        this.name = name;
        this.gender = gender;
    }
```

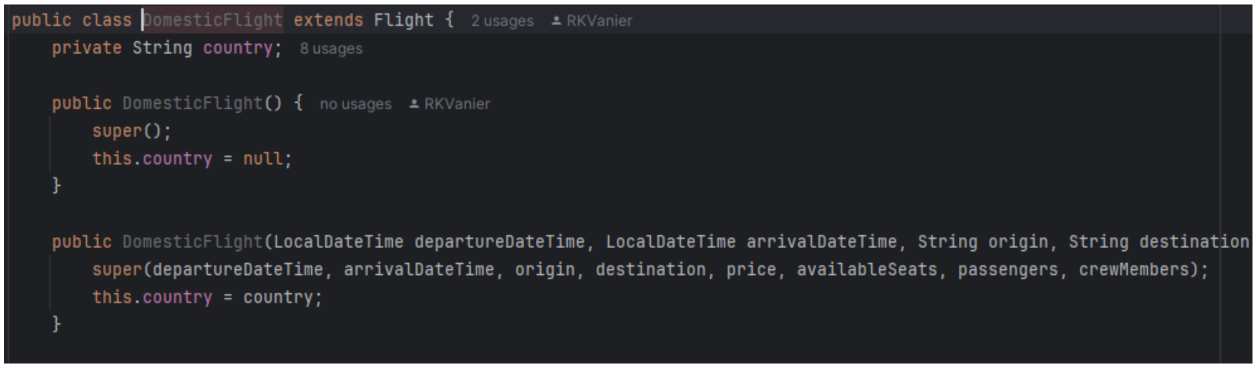
The child's class:

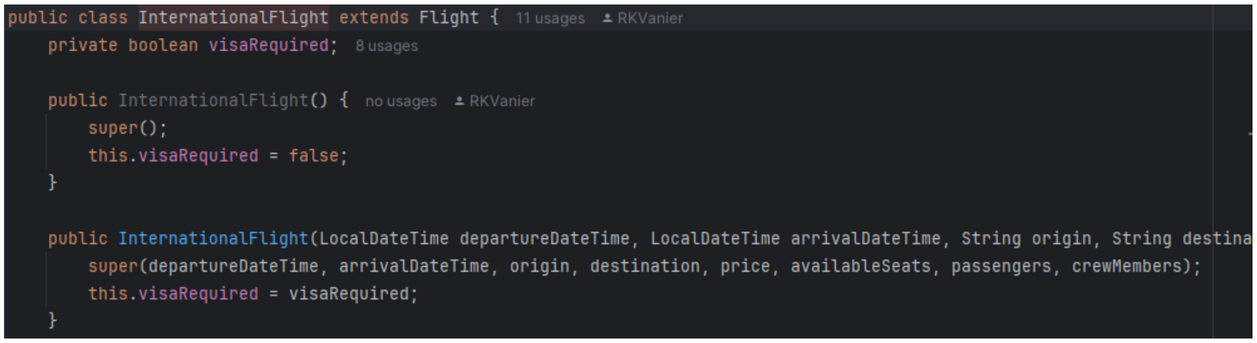

```
public class Admin extends User {  19 usages  RKVanier
    private int id;  8 usages
    private List<Booking> managedBooking;  9 usages

    private static int nextId = 1;  2 usages

    public Admin() {  6 usages  RKVanier
        super();
        this.id = nextId++;
        this.managedBooking = new ArrayList<>();
    }

    public Admin(String name, Gender gender, int id, List<Booking> managedBooking) {  no usages  RKVanier
```

The parent class:

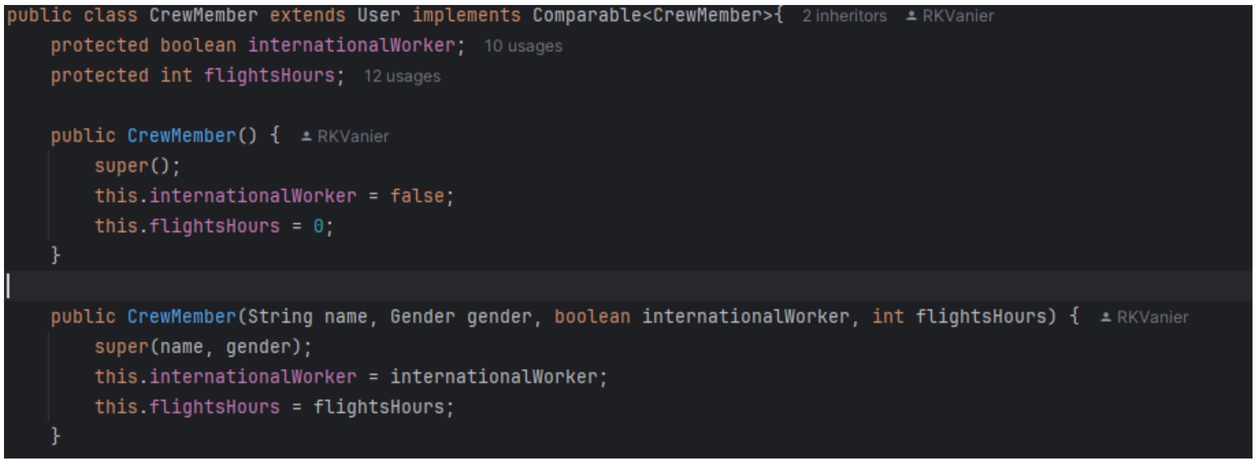

```
public abstract class Flight{  74 usages  17 inheritors  RKVanier
    protected int flightId;  11 usages
    protected LocalDateTime departureDateTime;  10 usages
    protected LocalDateTime arrivalDateTime;  10 usages
    protected String origin;  11 usages
    protected String destination;  11 usages
    protected double price;  11 usages
    protected int availableSeats;  13 usages
    protected List<Passenger> passengers;  13 usages
    protected List<CrewMember> crewMembers;  9 usages

    protected static int nextId = 1;  2 usages
```

The child class:


```
public class DomesticFlight extends Flight {  2 usages  RKVanier
    private String country;  8 usages

    public DomesticFlight() {  no usages  RKVanier
        super();
        this.country = null;
    }

    public DomesticFlight(LocalDateTime departureDateTime, LocalDateTime arrivalDateTime, String origin, String destination
        super(departureDateTime, arrivalDateTime, origin, destination, price, availableSeats, passengers, crewMembers);
        this.country = country;
    }
```

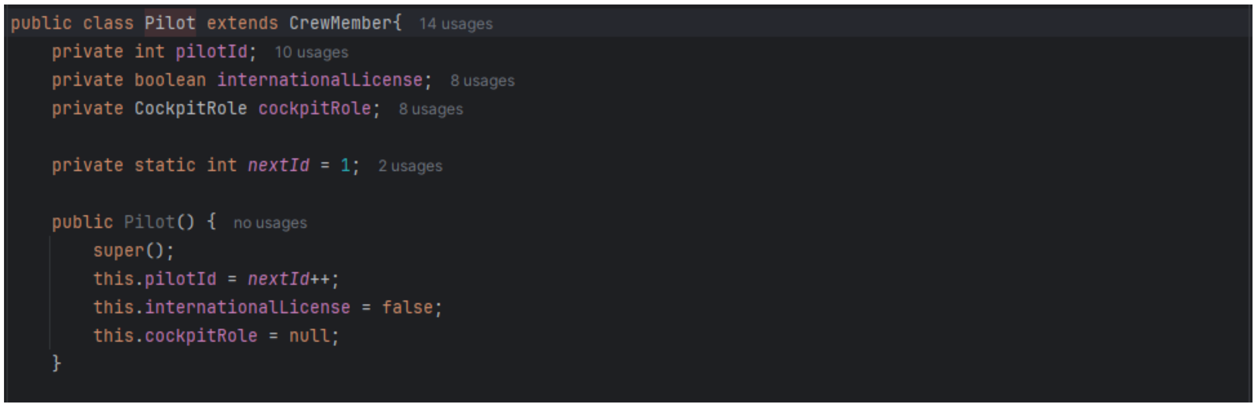

```
public class InternationalFlight extends Flight {  11 usages  RKVanier
    private boolean visaRequired;  8 usages

    public InternationalFlight() {  no usages  RKVanier
        super();
        this.visaRequired = false;
    }

    public InternationalFlight(LocalDateTime departureDateTime, LocalDateTime arrivalDateTime, String origin, String destina
        super(departureDateTime, arrivalDateTime, origin, destination, price, availableSeats, passengers, crewMembers);
        this.visaRequired = visaRequired;
    }
```

The class is both a child and parent:

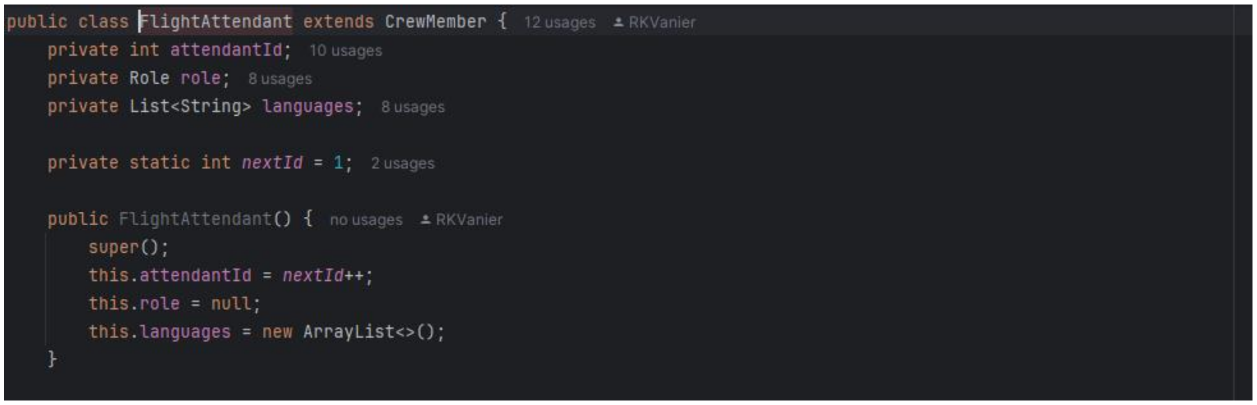


```java
public class CrewMember extends User implements Comparable<CrewMember>{  2 inheritors  RKVanier
    protected boolean internationalWorker;  10 usages
    protected int flightsHours;  12 usages

    public CrewMember() {  RKVanier
        super();
        this.internationalWorker = false;
        this.flightsHours = 0;
    }

    public CrewMember(String name, Gender gender, boolean internationalWorker, int flightsHours) {  RKVanier
        super(name, gender);
        this.internationalWorker = internationalWorker;
        this.flightsHours = flightsHours;
    }
```

Its child:

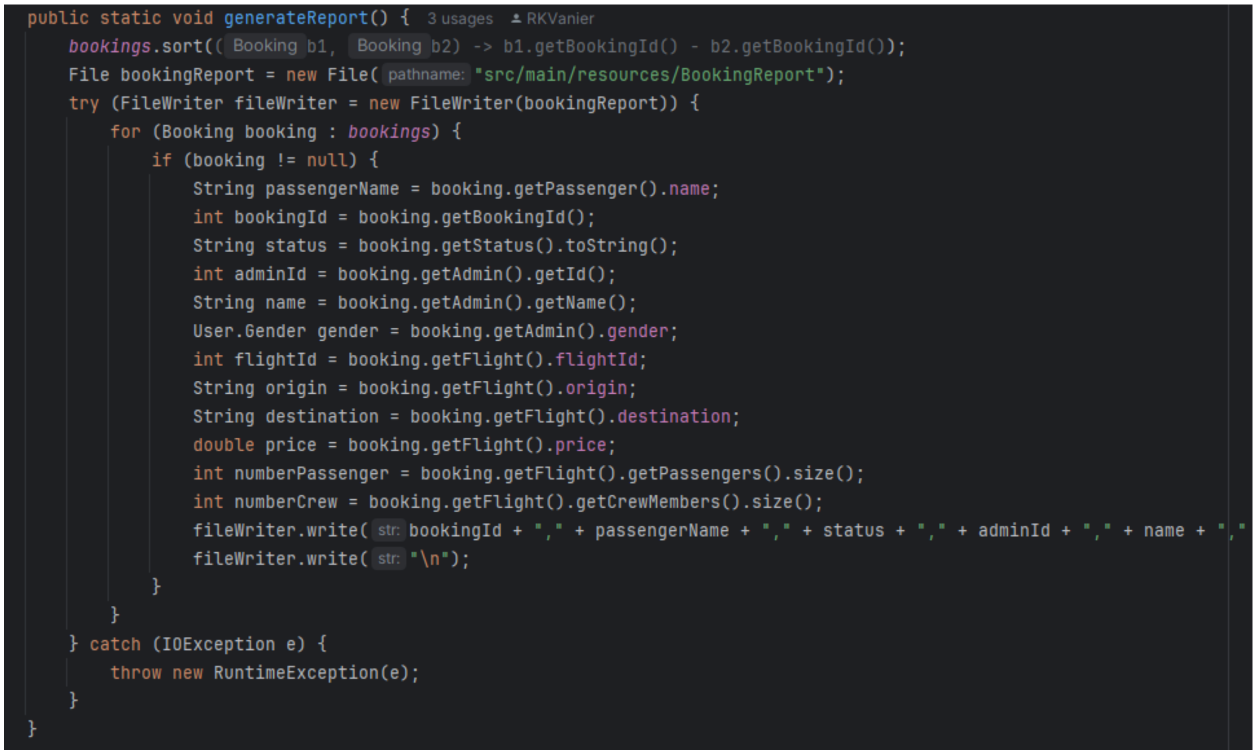


```java
public class Pilot extends CrewMember{  14 usages
    private int pilotId;  10 usages
    private boolean internationalLicense;  8 usages
    private CockpitRole cockpitRole;  8 usages

    private static int nextId = 1;  2 usages

    public Pilot() {  no usages
        super();
        this.pilotId = nextId++;
        this.internationalLicense = false;
        this.cockpitRole = null;
    }
```



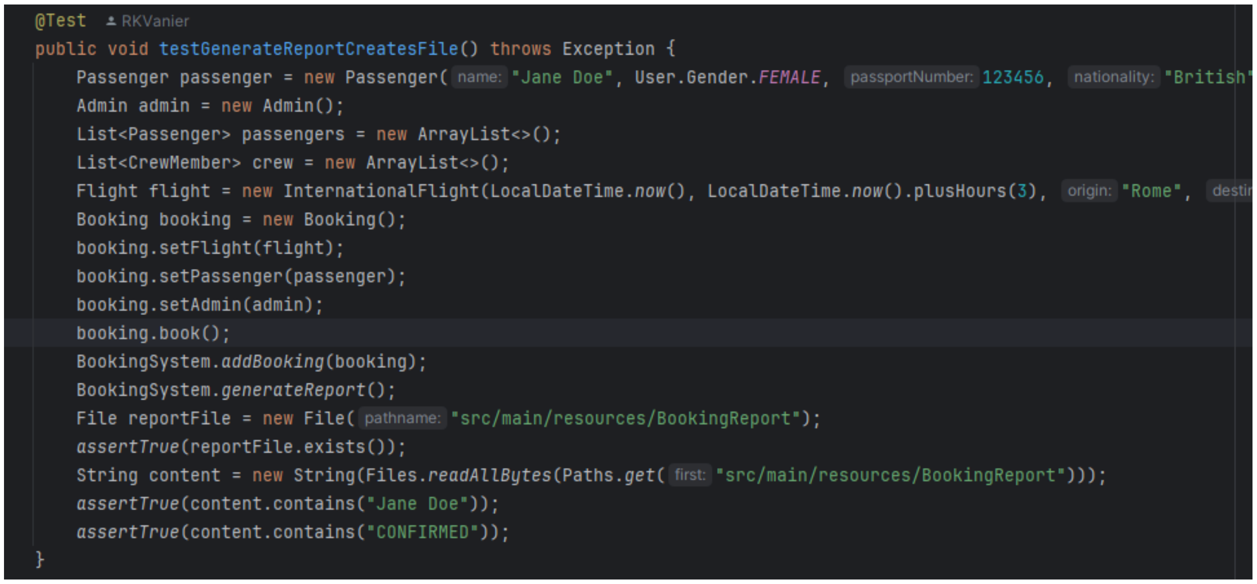
```java
public class FlightAttendant extends CrewMember {  12 usages  RKVanier
    private int attendantId;  10 usages
    private Role role;  8 usages
    private List<String> languages;  8 usages

    private static int nextId = 1;  2 usages

    public FlightAttendant() {  no usages  RKVanier
        super();
        this.attendantId = nextId++;
        this.role = null;
        this.languages = new ArrayList<>();
    }
```

Text out code:



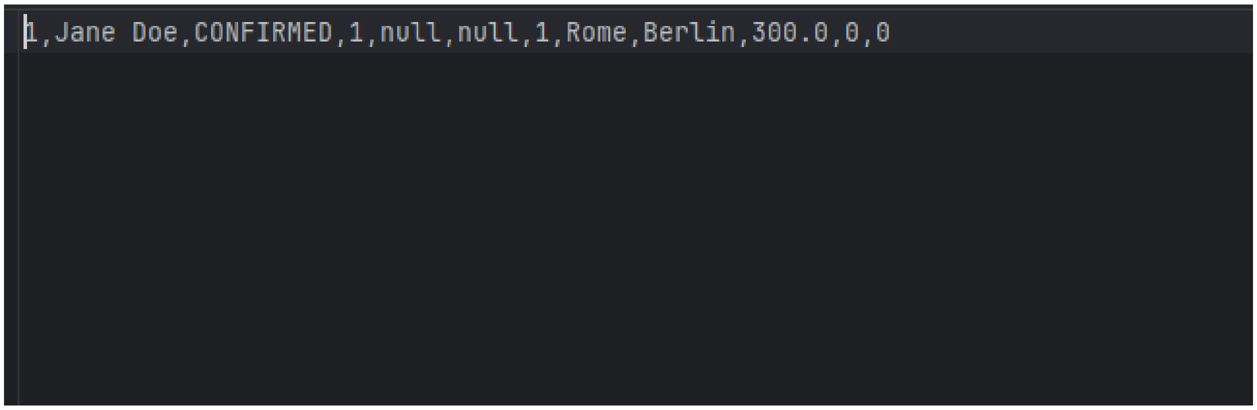
```
public static void generateReport() {  3 usages  RKVanier
    bookings.sort(( Booking b1,  Booking b2) -> b1.getBookingId() - b2.getBookingId());
    File bookingReport = new File( pathname: "src/main/resources/BookingReport");
    try (FileWriter fileWriter = new FileWriter(bookingReport)) {
        for (Booking booking : bookings) {
            if (booking != null) {
                String passengerName = booking.getPassenger().name;
                int bookingId = booking.getBookingId();
                String status = booking.getStatus().toString();
                int adminId = booking.getAdmin().getId();
                String name = booking.getAdmin().getName();
                User.Gender gender = booking.getAdmin().gender;
                int flightId = booking.getFlight().flightId;
                String origin = booking.getFlight().origin;
                String destination = booking.getFlight().destination;
                double price = booking.getFlight().price;
                int numberPassenger = booking.getFlight().getPassengers().size();
                int numberCrew = booking.getFlight().getCrewMembers().size();
                fileWriter.write( str: bookingId + "," + passengerName + "," + status + "," + adminId + "," + name + ","
                fileWriter.write( str: "\n");
            }
        }
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Testing of the code



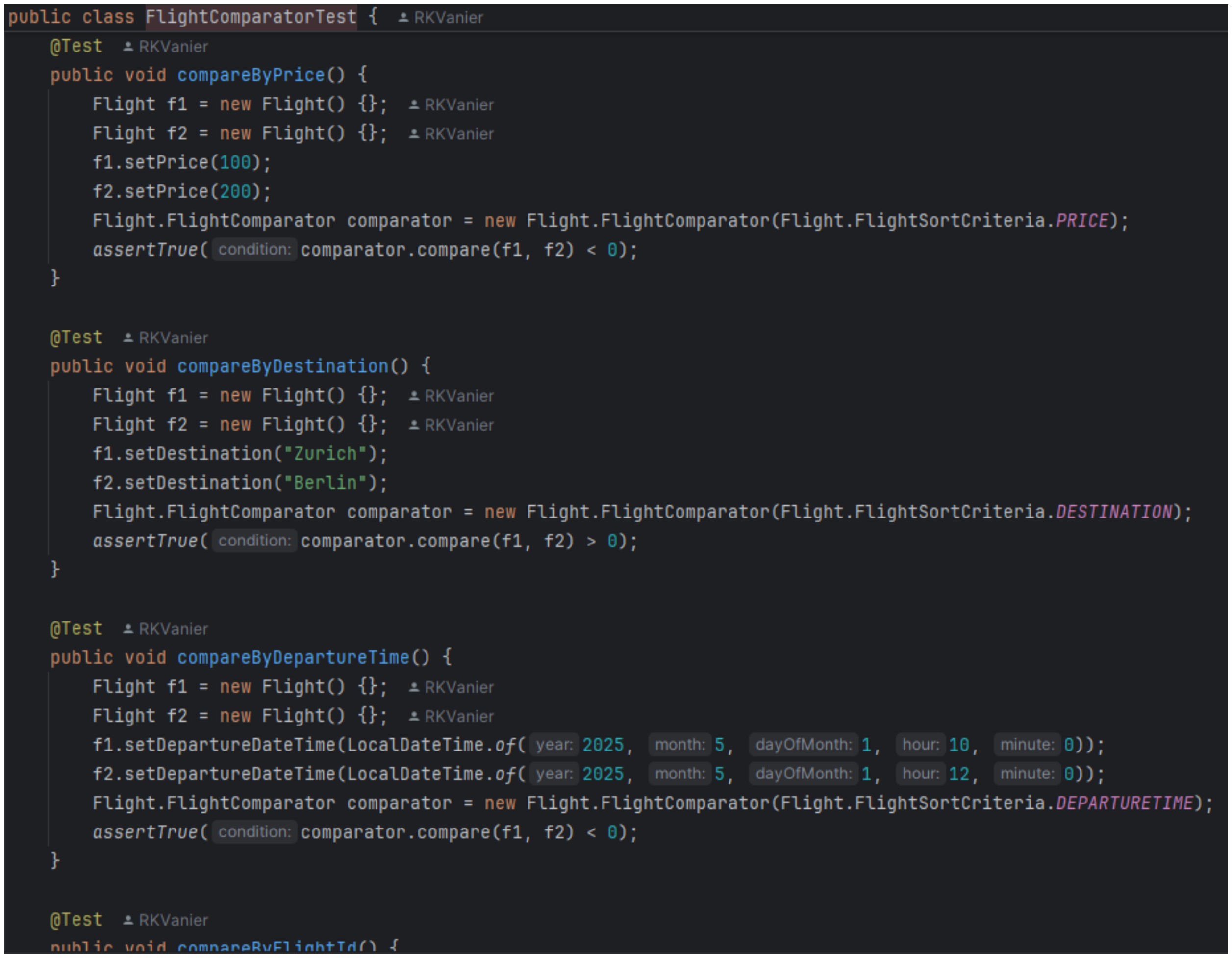
```
@Test  RKVanier
public void testGenerateReportCreatesFile() throws Exception {
    Passenger passenger = new Passenger( name: "Jane Doe", User.Gender.FEMALE,  passportNumber: 123456,  nationality: "British"
    Admin admin = new Admin();
    List<Passenger> passengers = new ArrayList<>();
    List<CrewMember> crew = new ArrayList<>();
    Flight flight = new InternationalFlight(LocalDateTime.now(), LocalDateTime.now().plusHours(3),  origin: "Rome",  destir
    Booking booking = new Booking();
    booking.setFlight(flight);
    booking.setPassenger(passenger);
    booking.setAdmin(admin);
    booking.book();
    BookingSystem.addBooking(booking);
    BookingSystem.generateReport();
    File reportFile = new File( pathname: "src/main/resources/BookingReport");
    assertTrue(reportFile.exists());
    String content = new String(Files.readAllBytes(Paths.get( first: "src/main/resources/BookingReport")));
    assertTrue(content.contains("Jane Doe"));
    assertTrue(content.contains("CONFIRMED"));
}
```

In the file:



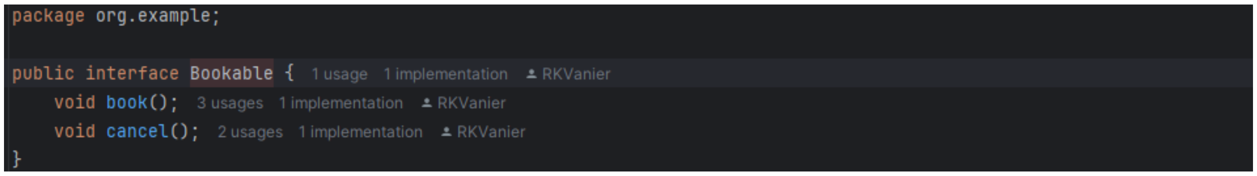
```
1,Jane Doe,CONFIRMED,1,null,null,1,Rome,Berlin,300.0,0,0
```

Testing of the comparator code that passed:



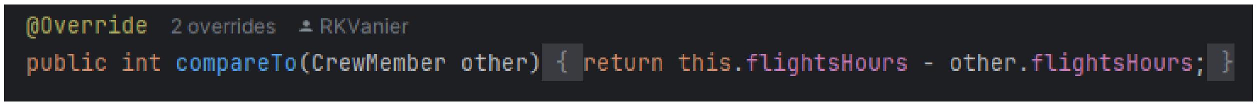
```
public class FlightComparatorTest {  RKVanier
    @Test  RKVanier
    public void compareByPrice() {
        Flight f1 = new Flight() {};  RKVanier
        Flight f2 = new Flight() {};  RKVanier
        f1.setPrice(100);
        f2.setPrice(200);
        Flight.FlightComparator comparator = new Flight.FlightComparator(Flight.FlightSortCriteria.PRICE);
        assertTrue( condition: comparator.compare(f1, f2) < 0);
    }

    @Test  RKVanier
    public void compareByDestination() {
        Flight f1 = new Flight() {};  RKVanier
        Flight f2 = new Flight() {};  RKVanier
        f1.setDestination("Zurich");
        f2.setDestination("Berlin");
        Flight.FlightComparator comparator = new Flight.FlightComparator(Flight.FlightSortCriteria.DESTINATION);
        assertTrue( condition: comparator.compare(f1, f2) > 0);
    }

    @Test  RKVanier
    public void compareByDepartureTime() {
        Flight f1 = new Flight() {};  RKVanier
        Flight f2 = new Flight() {};  RKVanier
        f1.setDepartureDateTime(LocalDateTime.of( year: 2025,  month: 5,  dayOfMonth: 1,  hour: 10,  minute: 0));
        f2.setDepartureDateTime(LocalDateTime.of( year: 2025,  month: 5,  dayOfMonth: 1,  hour: 12,  minute: 0));
        Flight.FlightComparator comparator = new Flight.FlightComparator(Flight.FlightSortCriteria.DEPARTURETIME);
        assertTrue( condition: comparator.compare(f1, f2) < 0);
    }

    @Test  RKVanier
    public void compareByFlightId() {
```

Interface for the booing methods:

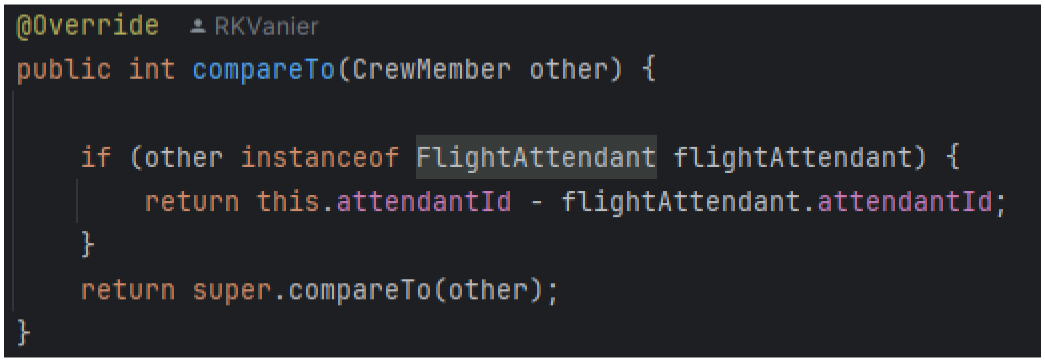


```
package org.example;

public interface Bookable {  1 usage  1 implementation  RKVanier
    void book();  3 usages  1 implementation  RKVanier
    void cancel();  2 usages  1 implementation  RKVanier
}
```

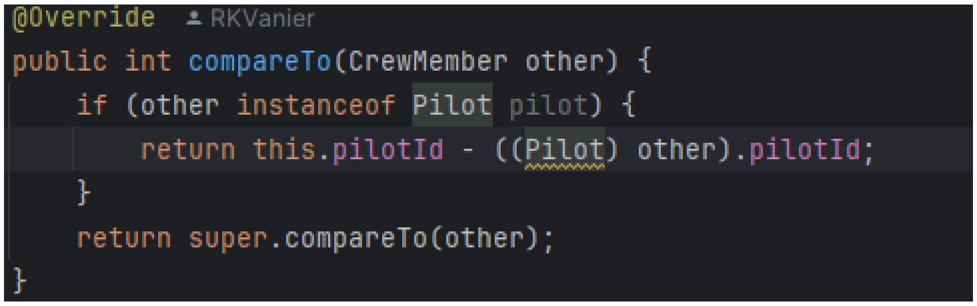
The case of polymorphism:



```
@Override  2 overrides  RKVanier
public int compareTo(CrewMember other) { return this.flightsHours - other.flightsHours; }
```



```
@Override  RKVanier
public int compareTo(CrewMember other) {

    if (other instanceof FlightAttendant flightAttendant) {
        return this.attendantId - flightAttendant.attendantId;
    }
    return super.compareTo(other);
}
```



```
@Override  RKVanier
public int compareTo(CrewMember other) {
    if (other instanceof Pilot pilot) {
        return this.pilotId - ((Pilot) other).pilotId;
    }
    return super.compareTo(other);
}
```

## 4. Challenges

- **Class Communication**: One of the most challenging aspects was getting different classes to interact properly, especially when passing objects like Admin, Passenger, or Flight between methods. Coordinating references across unrelated classes such as ensuring a Booking had access to both the Admin and the Passenger at the right time required careful design and debugging.
- **File I/O Complexity**: Implementing file writing and optionally reading for booking reports introduced edge cases like handling file overwriting, ensuring thread safety, and formatting readable output.
- **NullPointerExceptions**: Several errors occurred due to uninitialized objects such as missing admin references in the booking logic. These bugs were often hard to trace due to indirect method calls.

- **Comparable in Subclasses**: Implementing compareTo in subclasses like Pilot and FlightAttendant while preserving the behavior of CrewMember required understanding how inheritance and overriding work in Java's Comparable system.
- **Static ID Conflicts in Tests**: Managing auto-increment IDs such as pilotId and attendantId across unit tests occasionally caused inconsistency in expected values, especially when tests were re-run without resetting static counters.

## 5. Learning Outcomes

- Deepened understanding of **inheritance and class hierarchies** in Java.
- Gained practical experience with **interfaces and abstract classes**.
- Learned to apply **Comparable** and **Comparator** interfaces for sorting.
- Understood the importance of **unit testing** using JUnit.
- Developed the ability to use **file I/O** for persistent data storage.
- Improved debugging skills, especially for object references and null handling.