IY4113 Milestone 4

Assessment Details	Please Complete All Details
Group	B
Module Title	Object Oreinatated Programming
Assessment Type	Coursework
Module Tutor Name	Jonathon Shore
Student ID Number	P508377
Date of Submission	15/02/26
Word Count	1603
GItHub Link	

- [x] *I confirm that this assignment is my own work. Where I have referred to academic sources, I have provided in-text citations and included the sources in the final reference list.*
- [x] *Where I have used AI, I have cited and referenced appropriately.

Program Code

```
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.ArrayList;
import java.util.Scanner;

public class CityRideLite {
```

```
private final ArrayList<Journey> journeyList;
private int nextJourneyId;


private BigDecimal adultTotal = BigDecimal.ZERO;
private BigDecimal studentTotal = BigDecimal.ZERO;
private BigDecimal childTotal = BigDecimal.ZERO;
private BigDecimal seniorTotal = BigDecimal.ZERO;

public CityRideLite() {
    journeyList = new ArrayList<>();
    nextJourneyId = 1;
}

public static void main(String[] args) {
    CityRideLite app = new CityRideLite();
    app.runProgram();
}

private void runProgram() {

    Scanner inputScanner = new Scanner(System.in);
    boolean isRunning = true;

    System.out.println("---------------------------------");
    System.out.println("     Welcome to CityRide Lite    ");
    System.out.println("---------------------------------");

    while (isRunning) {

        displayMainMenu();
        String userChoice = inputScanner.nextLine();

        if (userChoice.equals("1")) {
            addJourney(inputScanner);

        } else if (userChoice.equals("2")) {
            listAllJourneys();

        } else if (userChoice.equals("3")) {
            isRunning = false;
            System.out.println("Goodbye!");

        } else {
            System.out.println("Error: Please select a valid option.");
        }
    }

    inputScanner.close();
}
```

```
private void displayMainMenu() {

    System.out.println("\n--- Main Menu ---");
    System.out.println("1. Add Journey");
    System.out.println("2. List Journeys");
    System.out.println("3. Exit");
    System.out.print("Enter choice: ");
}

private void addJourney(Scanner inputScanner) {

    System.out.println("\n--- Add Journey ---");

    System.out.print("Enter date (DD/MM/YYYY): ");
    String journeyDate = inputScanner.nextLine();

    int fromZone = getValidZone(inputScanner, "Enter starting zone (1-5): ");
    int toZone = getValidZone(inputScanner, "Enter destination zone (1-5): ");

    CityRideDataset.PassengerType passengerType =
            getValidPassengerType(inputScanner);

    CityRideDataset.TimeBand timeBand =
            getValidTimeBand(inputScanner);


    BigDecimal baseFare = CityRideDataset.getBaseFare(fromZone, toZone, timeBand);

    BigDecimal discountedFare =
            FareCalculator.applyDiscount(baseFare, passengerType);

    BigDecimal chargedFare =
            applyDailyCap(passengerType, discountedFare);

    Journey newJourney = new Journey(
            nextJourneyId,
            journeyDate,
            fromZone,
            toZone,
            passengerType,
            timeBand,
            baseFare,
            discountedFare,
            chargedFare
    );

    journeyList.add(newJourney);
    nextJourneyId++;

    System.out.println("Journey added successfully.");
```

```
    System.out.println("Zones crossed: " + newJourney.getZonesCrossed());
    System.out.println("Charged fare: £" + chargedFare);
}

private void listAllJourneys() {

    System.out.println("\n--- Journey Records ---");

    if (journeyList.isEmpty()) {
        System.out.println("No journeys have been recorded yet.");
        return;
    }

    for (Journey journey : journeyList) {
        System.out.println(journey);
    }
}

private int getValidZone(Scanner inputScanner, String message) {

    int zoneNumber = 0;
    boolean isValidZone = false;

    while (!isValidZone) {

        System.out.print(message);

        try {
            zoneNumber = Integer.parseInt(inputScanner.nextLine());

            if (zoneNumber >= 1 && zoneNumber <= 5) {
                isValidZone = true;
            } else {
                System.out.println("Error: Zones must be between 1 and 5.");
            }

        } catch (NumberFormatException exception) {
            System.out.println("Error: Please enter a valid numeric zone.");
        }
    }

    return zoneNumber;
}

private CityRideDataset.PassengerType getValidPassengerType(Scanner inputScanner) {

    while (true) {

        System.out.print("Enter passenger type (ADULT/STUDENT/CHILD/SENIOR_CITIZEN): ");
        String input = inputScanner.nextLine().toUpperCase();
```

```
        try {
            return CityRideDataset.PassengerType.valueOf(input);

        } catch (IllegalArgumentException exception) {
            System.out.println("Error: Unknown passenger type.");
        }
    }
}

private CityRideDataset.TimeBand getValidTimeBand(Scanner inputScanner) {

    while (true) {

        System.out.print("Enter time band (PEAK/OFF_PEAK): ");
        String input = inputScanner.nextLine().toUpperCase();

        try {
            return CityRideDataset.TimeBand.valueOf(input);

        } catch (IllegalArgumentException exception) {
            System.out.println("Error: Unknown time band.");
        }
    }
}

private BigDecimal applyDailyCap(CityRideDataset.PassengerType type, BigDecimal fare) {

    BigDecimal cap = CityRideDataset.DAILY_CAP.get(type);
    BigDecimal currentTotal = getPassengerTotal(type);


    if (currentTotal.compareTo(cap) >= 0) {
        return BigDecimal.ZERO.setScale(2);
    }

    BigDecimal newTotal = currentTotal.add(fare);


    if (newTotal.compareTo(cap) > 0) {
        BigDecimal allowed = cap.subtract(currentTotal);
        setPassengerTotal(type, cap);
        return allowed.setScale(2, RoundingMode.HALF_UP);
    }

    setPassengerTotal(type, newTotal);
    return fare.setScale(2, RoundingMode.HALF_UP);
}

private BigDecimal getPassengerTotal(CityRideDataset.PassengerType type) {

    if (type == CityRideDataset.PassengerType.ADULT) return adultTotal;
```

```
    if (type == CityRideDataset.PassengerType.STUDENT) return studentTotal;
    if (type == CityRideDataset.PassengerType.CHILD) return childTotal;
    return seniorTotal;
  }

  private void setPassengerTotal(CityRideDataset.PassengerType type, BigDecimal value) {

    if (type == CityRideDataset.PassengerType.ADULT) adultTotal = value;
    else if (type == CityRideDataset.PassengerType.STUDENT) studentTotal = value;
    else if (type == CityRideDataset.PassengerType.CHILD) childTotal = value;
    else seniorTotal = value;
  }
```

}

class Journey {

```
private final int journeyId;
private final String date;
private final int fromZone;
private final int toZone;

private final CityRideDataset.PassengerType passengerType;
private final CityRideDataset.TimeBand timeBand;

private final int zonesCrossed;

private final BigDecimal baseFare;
private final BigDecimal discountedFare;
private final BigDecimal chargedFare;

public Journey(int journeyId,
               String date,
               int fromZone,
               int toZone,
               CityRideDataset.PassengerType passengerType,
               CityRideDataset.TimeBand timeBand,
               BigDecimal baseFare,
               BigDecimal discountedFare,
               BigDecimal chargedFare) {

    this.journeyId = journeyId;
    this.date = date;
    this.fromZone = fromZone;
    this.toZone = toZone;
    this.passengerType = passengerType;
    this.timeBand = timeBand;

    this.zonesCrossed = Math.abs(toZone - fromZone) + 1;

    this.baseFare = baseFare;
    this.discountedFare = discountedFare;
    this.chargedFare = chargedFare;
}

public int getZonesCrossed() {
    return zonesCrossed;
}

@Override
public String toString() {

    return "ID: " + journeyId +
            " | Date: " + date +
            " | From: " + fromZone +
            " -> To: " + toZone +
            " | Passenger: " + passengerType +
```

```
            " | Band: " + timeBand +
            " | Charged: £" + chargedFare;
  }
```

}

class FareCalculator {

```
  public static BigDecimal applyDiscount(BigDecimal baseFare,
                                         CityRideDataset.PassengerType type) {

      BigDecimal discountRate = CityRideDataset.DISCOUNT_RATE.get(type);

      BigDecimal discountAmount = baseFare.multiply(discountRate);

      return baseFare.subtract(discountAmount)
              .setScale(2, RoundingMode.HALF_UP);
  }
```

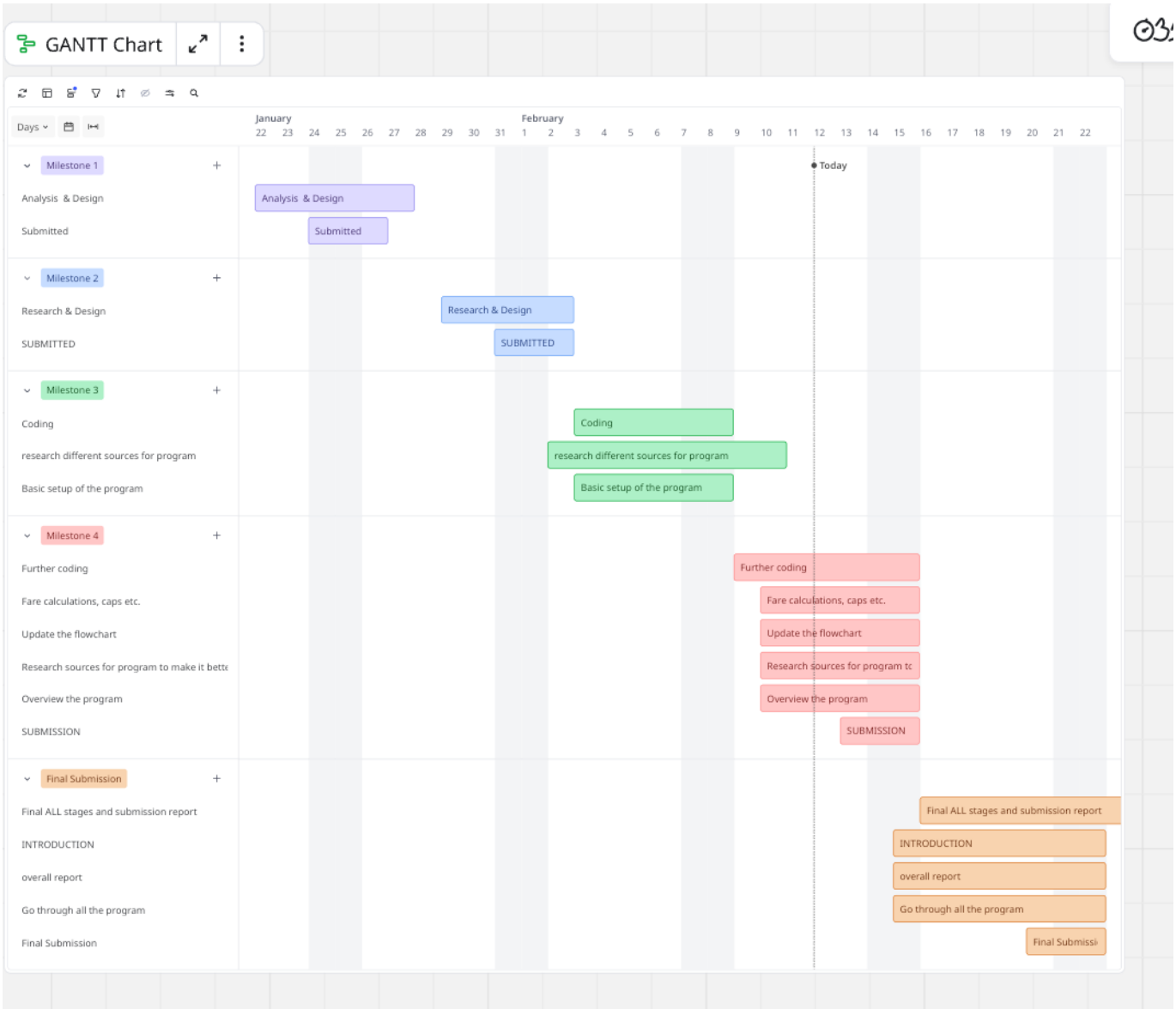
}

final class CityRideDataset {

```
  private CityRideDataset() {
  }

  public enum TimeBand {
      PEAK,
      OFF_PEAK
  }

  public enum PassengerType {
      ADULT,
      STUDENT,
      CHILD,
      SENIOR_CITIZEN
  }

  public static final java.util.Map<PassengerType, BigDecimal> DISCOUNT_RATE =
          java.util.Map.of(
                  PassengerType.ADULT, new BigDecimal("0.00"),
                  PassengerType.STUDENT, new BigDecimal("0.25"),
                  PassengerType.CHILD, new BigDecimal("0.50"),
                  PassengerType.SENIOR_CITIZEN, new BigDecimal("0.30")
          );

  public static final java.util.Map<PassengerType, BigDecimal> DAILY_CAP =
          java.util.Map.of(
                  PassengerType.ADULT, new BigDecimal("8.00"),
                  PassengerType.STUDENT, new BigDecimal("6.00"),
                  PassengerType.CHILD, new BigDecimal("4.00"),
                  PassengerType.SENIOR_CITIZEN, new BigDecimal("7.00")
          );

  public static BigDecimal getBaseFare(int fromZone, int toZone, TimeBand band) {


      if (band == TimeBand.PEAK) {
          return new BigDecimal("3.20");
      }

      return new BigDecimal("2.70");
  }
```

}

Updated Gantt Chart



Diary Entries

12/02/26-This week I focused on improving the overall structure of my CityRide Lite program. In Milestone 3, my program was only able to add journeys, calculate zones crossed, and list journeys back to the user. It worked as a basic foundation, but it was missing most of the fare rules from the assessment specification.

My main goal was to start moving the program closer to the real requirements, especially adding fare calculation and discounts. I realised early on that I needed to stop treating passenger type and time band as free text inputs, because that would allow invalid values and would cause problems later when calculating fares. To improve this, I updated the program to use enums from the dataset, which made the input more controlled and consistent.

This also helped me keep the code cleaner and closer to object-oriented practice, because the passenger types and time bands now match the provided dataset rules instead of relying on

user spelling.

13/02/26-One of the biggest issues I encountered this week was that after adding more classes (such as fare calculation and dataset access), my program started producing a large number of compile errors. At one point, I had over 20 errors showing in the IDE, which was honestly quite stressful.

After investigating, I realised the main reason was that Java is strict about class structure. I was trying to combine multiple supporting classes without properly organising them, and I was also missing the CityRideDataset class connection, which caused many “cannot find symbol” errors.

To fix this, I reorganised the code carefully and made sure all required classes were included correctly. I also improved my validation methods so the program could safely handle incorrect inputs without crashing. For example, I continued using Integer.parseInt() inside a try/catch block to prevent non-numeric zone entries from terminating the program.

Once the structure was corrected, the program successfully ran again with journey storage, proper input validation, and fewer issues during execution.

14/02/26-After fixing the major compile problems, I moved on to implementing the most important milestone improvement: fare calculation.

This was the first time my program started behaving like an actual travel fare tracker rather than just storing journeys. I added a FareCalculator class to apply passenger discounts based on the dataset:

- Student: 25%
- Child: 50%
- Senior Citizen: 30%
- Adult: no discount

I also introduced daily cap tracking, which was one of the more challenging parts. I had to maintain separate running totals for each passenger type and ensure that once the cap was reached, further journeys were charged £0.00.

Implementing this required careful logic, especially for journeys that partially exceed the cap. For example, if the passenger only had £0.50 remaining before reaching their cap, the journey charge needed to be reduced so the total stopped exactly at the cap.

This part helped me understand how important it is to test edge cases, not just normal journeys. It also made the program feel much closer to the assessment specification.

15/02/26-Alongside coding, I also made improvements to my design work based on feedback. My earlier flowchart was too simple and didn't include enough detail about subroutines or internal processes.

To address this, I updated the flowchart to show clearer functional decomposition, including separate steps for:

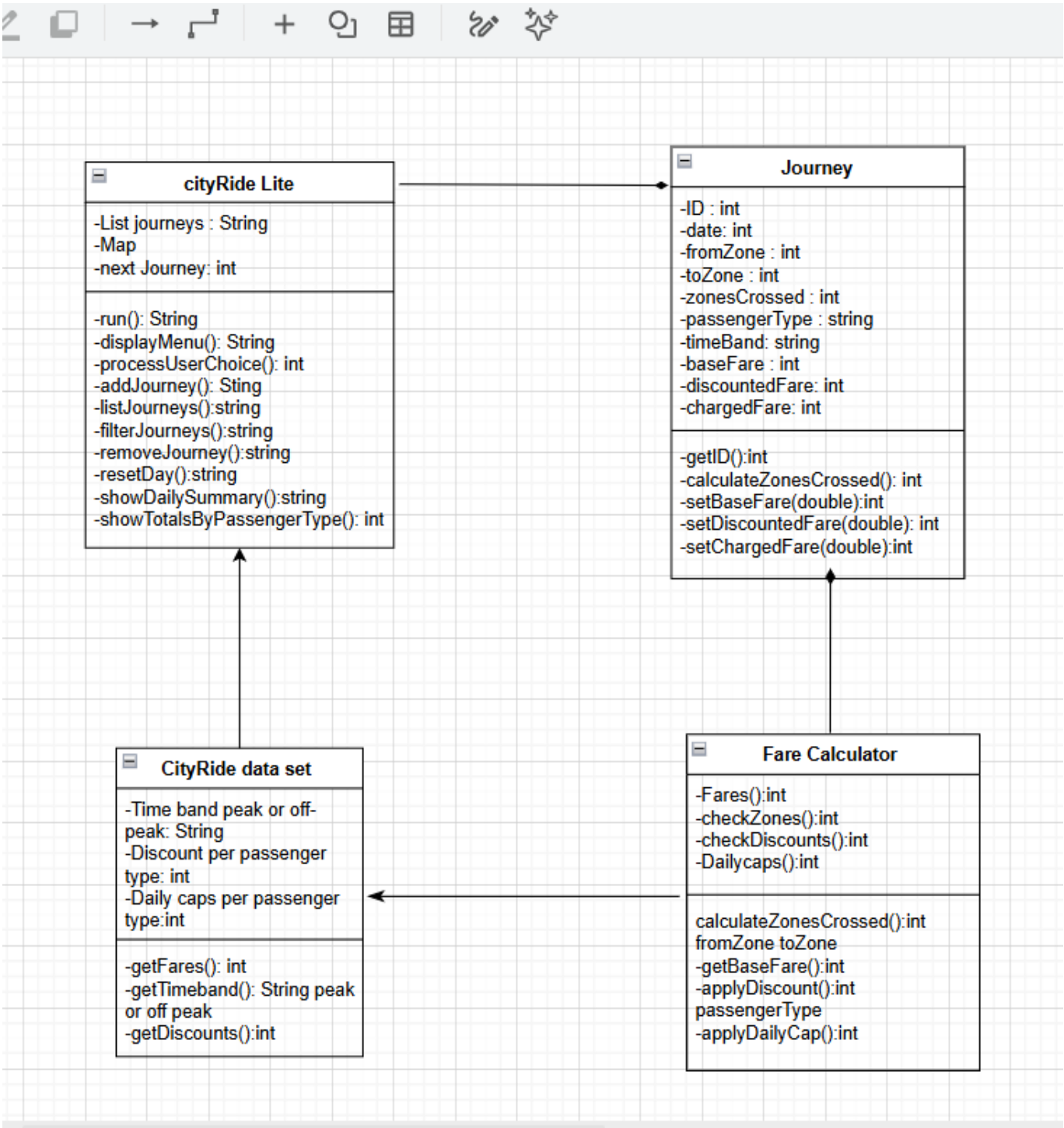
- validating input
- calculating zones crossed
- applying discounts
- enforcing daily caps
- storing journeys in memory

This made the algorithm much easier to follow and more aligned with how the program actually works.

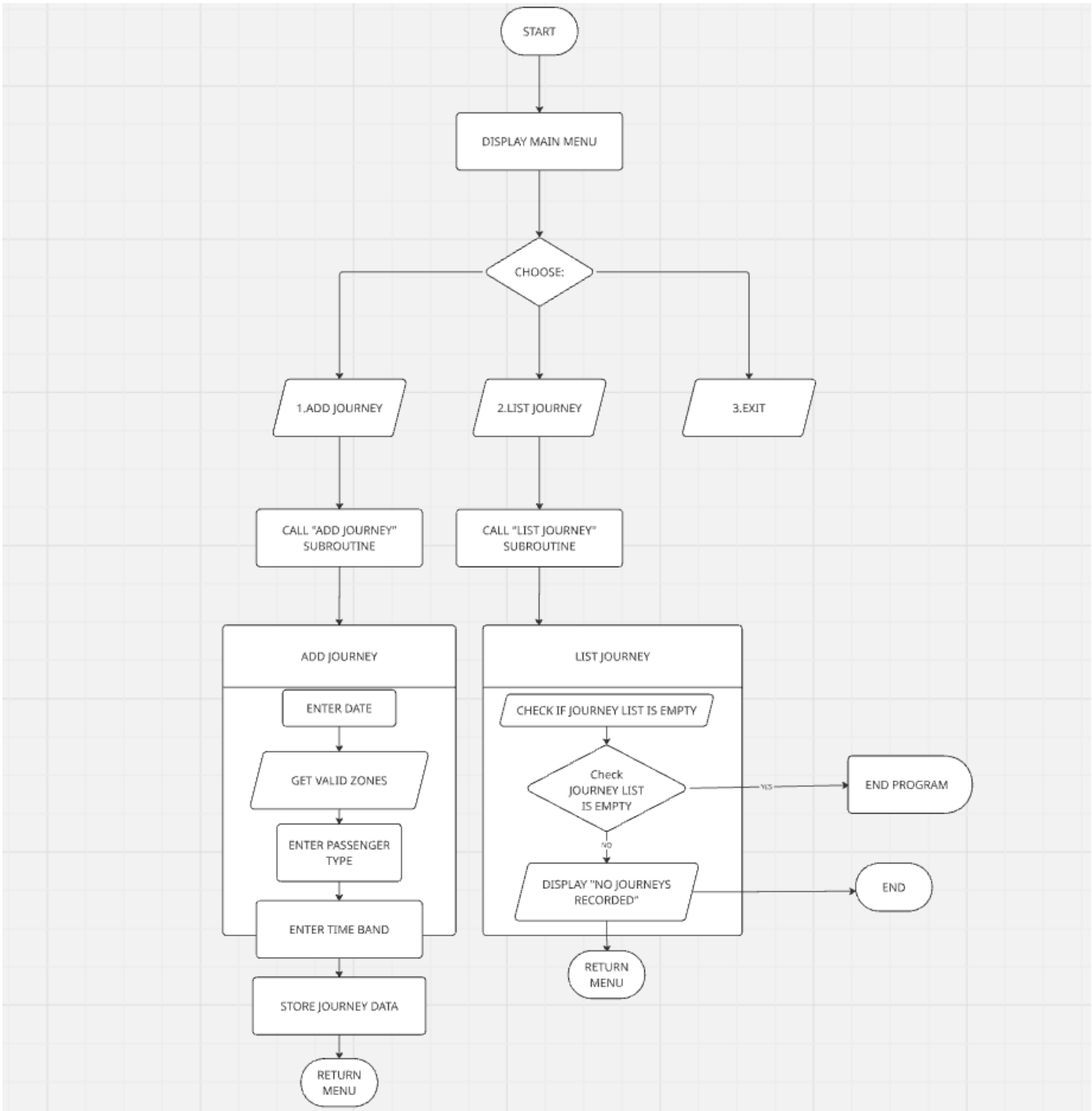
I also improved the UML class diagram after receiving feedback that an important class was missing. I added the CityRideDataset class properly, along with fare-related attributes such as base fare, discounted fare, and charged fare. This helped clarify the relationship between the Journey class, FareCalculator, and dataset rules.

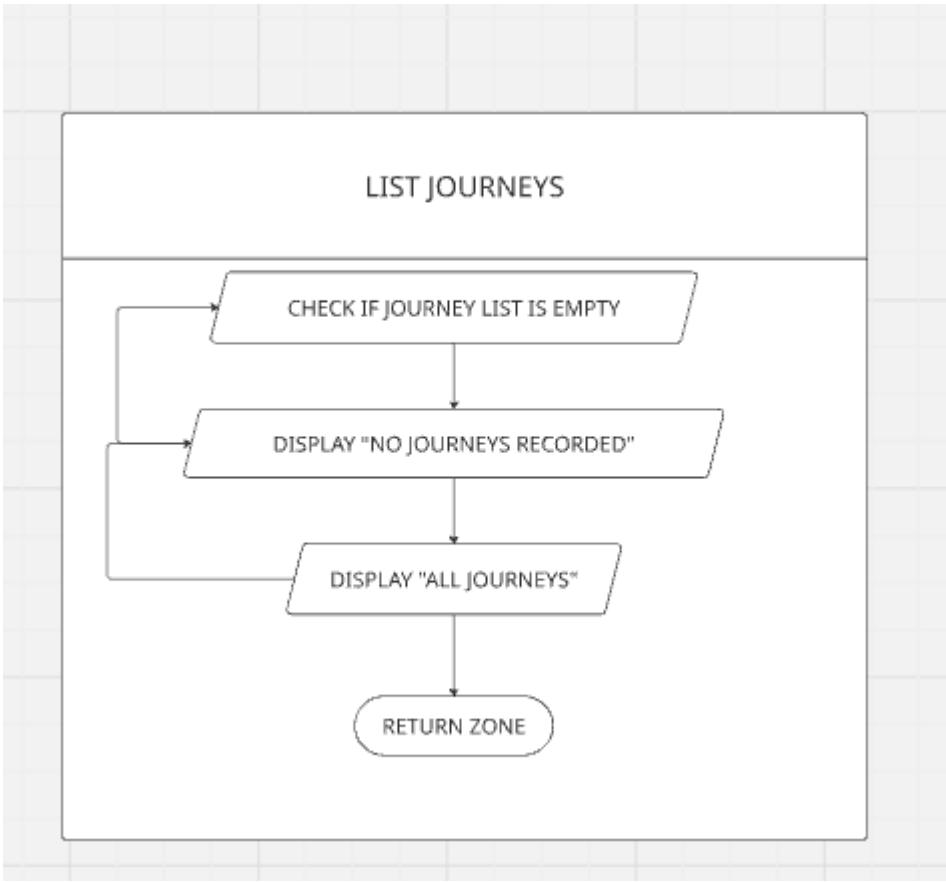
Overall, this week felt like a major step forward because both the code and design documentation now match each other more accurately. The program is still not fully complete yet (filtering, removal, and summaries are still in progress), but the fare system is now working correctly, and the structure is much more professional.

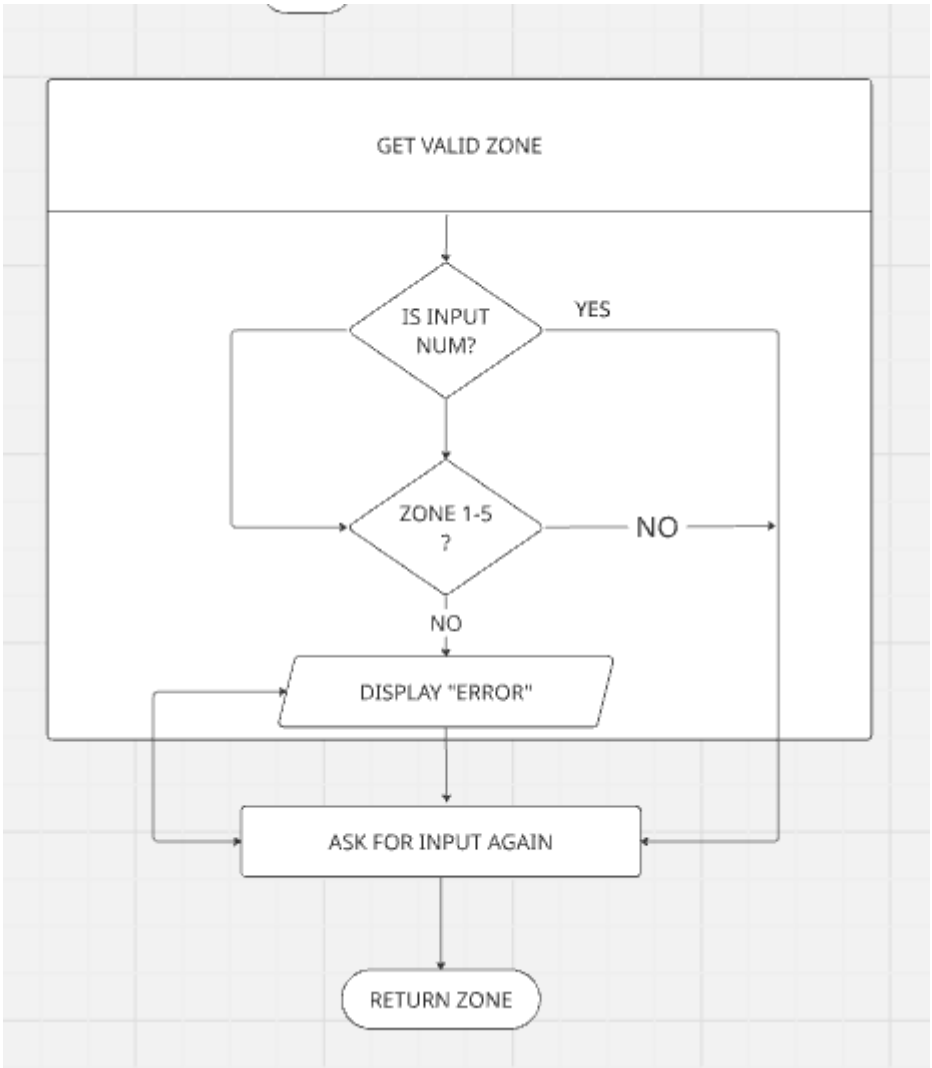
Class Diagram(Updated)



FLOWCHART UPDATED


START
DISPLAY MAIN MENU
CHOOSE:
1.ADD JOURNEY
2.LIST JOURNEY
3.EXIT
CALL "ADD JOURNEY" SUBROUTINE
CALL "LIST JOURNEY" SUBROUTINE
ADD JOURNEY
ENTER DATE
GET VALID ZONES
ENTER PASSENGER TYPE
ENTER TIME BAND
STORE JOURNEY DATA
RETURN MENU
LIST JOURNEY
CHECK IF JOURNEY LIST IS EMPTY
Check JOURNEY LIST IS EMPTY
YES
END PROGRAM
NO
DISPLAY "NO JOURNEYS RECORDED"
END
RETURN MENU


LIST JOURNEYS
CHECK IF JOURNEY LIST IS EMPTY
DISPLAY "NO JOURNEYS RECORDED"
DISPLAY "ALL JOURNEYS"
RETURN ZONE


GET VALID ZONE
IS INPUT NUM?
YES
ZONE 1-5 ?
NO
NO
DISPLAY "ERROR"
ASK FOR INPUT AGAIN
RETURN ZONE