NAME: Darshan NM

STUDENT ID: CA/SE1/6780

DOMAIN: Java Programming

DURATION: 10th September 2025 to 10th October 2025

TASK 1

Student Grade Tracker:

● Build a Java program to input and manage student grades.

● Calculate average, highest, and lowest scores.

● Use arrays or ArrayLists to store and manage data.

● Display a summary report of all students.

● Make the interface console-based or GUI-based as desired.

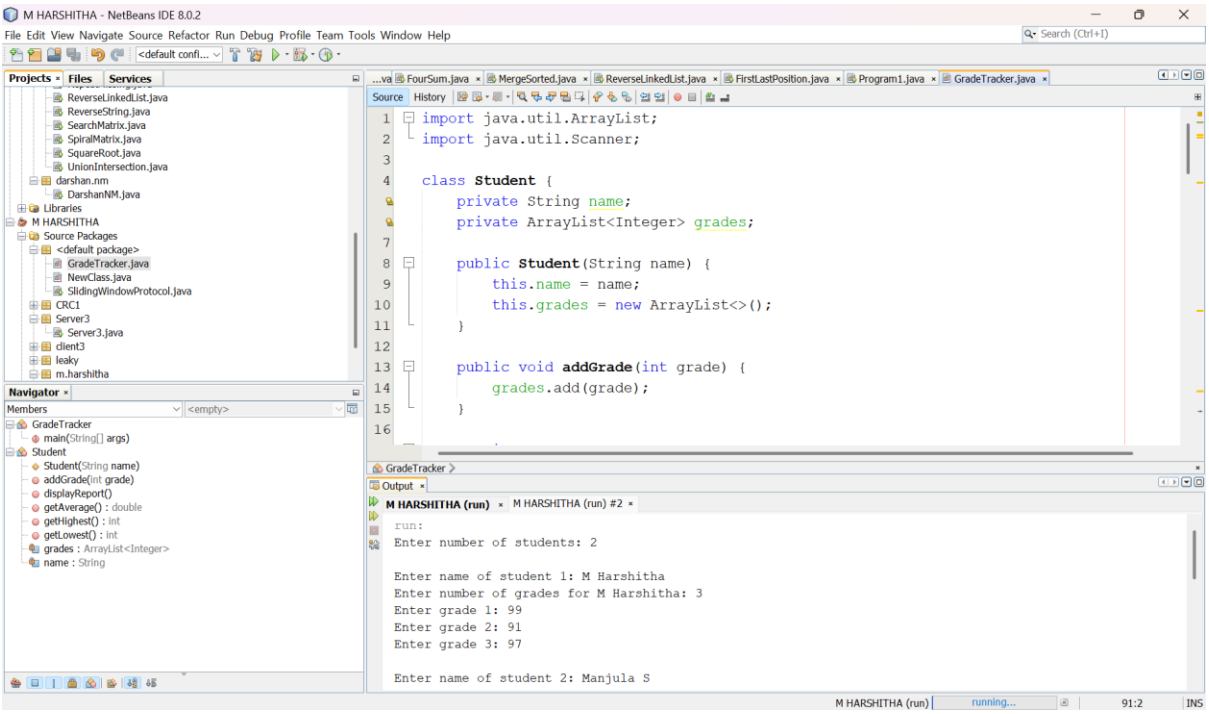
Program 1:

```
import java.util.ArrayList;
import java.util.Scanner;

class Student {
    private String name;
    private ArrayList<Integer> grades;

    public Student(String name) {
        this.name = name;
        this.grades = new ArrayList<>();
    }

    public void addGrade(int grade) {
        grades.add(grade);
    }
```

```
public double getAverage() {
    if (grades.isEmpty()) return 0;
    int sum = 0;
    for (int grade : grades) {
        sum += grade;
    }
    return (double) sum / grades.size();
}

public int getHighest() {
    if (grades.isEmpty()) return 0;
    int max = grades.get(0);
    for (int grade : grades) {
        if (grade > max) {
            max = grade;
        }
    }
    return max;
}

public int getLowest() {
    if (grades.isEmpty()) return 0;
    int min = grades.get(0);
    for (int grade : grades) {
        if (grade < min) {
            min = grade;
        }
    }
    return min;
```

```
    }

    public void displayReport() {
        System.out.println("Student Name: " + name);
        System.out.println("Grades: " + grades);
        System.out.printf("Average Score: %.2f\n", getAverage());
        System.out.println("Highest Score: " + getHighest());
        System.out.println("Lowest Score: " + getLowest());
        System.out.println("----------------------------");
    }
}

public class GradeTracker {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        ArrayList<Student> students = new ArrayList<>();

        System.out.print("Enter number of students: ");
        int numStudents = scanner.nextInt();
        scanner.nextLine(); // consume newline

        for (int i = 0; i < numStudents; i++) {
            System.out.print("\nEnter name of student " + (i + 1) + ": ");
            String name = scanner.nextLine();
            Student student = new Student(name);

            System.out.print("Enter number of grades for " + name + ": ");
            int numGrades = scanner.nextInt();
```

```
            for (int j = 0; j < numGrades; j++) {
                System.out.print("Enter grade " + (j + 1) + ": ");
                int grade = scanner.nextInt();
                student.addGrade(grade);
            }
            scanner.nextLine(); // consume newline
            students.add(student);
        }

        System.out.println("\n===== Student Grade Summary =====");
        for (Student student : students) {
            student.displayReport();
        }

        scanner.close();
    }
}
```



Output 1:

```
run:
Enter number of students: 2

Enter name of student 1: M Harshitha
Enter number of grades for M Harshitha: 3
Enter grade 1: 99
Enter grade 2: 91
Enter grade 3: 97

Enter name of student 2: Manjula S
Enter number of grades for Manjula S: 3
Enter grade 1: 56
Enter grade 2: 67
Enter grade 3: 45

===== Student Grade Summary =====
Student Name: M Harshitha
Grades: [99, 91, 97]
Average Score: 95.67
Highest Score: 99
Lowest Score: 91
-----------------------------
Student Name: Manjula S
Grades: [56, 67, 45]
Average Score: 56.00
Highest Score: 67
Lowest Score: 45
-----------------------------
BUILD SUCCESSFUL (total time: 1 minute 7 seconds)
```

TASK 2

Stock Trading Platform:

- Simulate a basic stock trading environment.
- Add features for market data display and buy/sell operations.
- Allow users to track portfolio performance over time.
- Use Object-Oriented Programming (OOP) to manage stocks, users and transactions.
- Optionally, include file I/O or database to persist portfolio data.

Program 2:

```
import java.util.*;

public class StockTradingPlatform {

    // --- Stock Class ---
    static class Stock {
        private String symbol;
        private String name;
        private double price;

        public Stock(String symbol, String name, double price) {
            this.symbol = symbol;
            this.name = name;
            this.price = price;
        }

        public void updatePrice() {
            double change = (Math.random() * 10) - 5; // ±5%
            price += price * (change / 100);
```

```
        price = Math.round(price * 100.0) / 100.0;
    }

    public String getSymbol() { return symbol; }
    public String getName() { return name; }
    public double getPrice() { return price; }

    @Override
    public String toString() {
        return symbol + " - " + name + ": $" + price;
    }
}

// --- Market Class ---
static class Market {
    private Map<String, Stock> stocks = new HashMap<>();

    public Market() {
        stocks.put("AAPL", new Stock("AAPL", "Apple Inc.", 150.0));
        stocks.put("GOOG", new Stock("GOOG", "Alphabet Inc.", 2800.0));
        stocks.put("TSLA", new Stock("TSLA", "Tesla Inc.", 700.0));
    }

    public void updateMarket() {
        for (Stock stock : stocks.values()) {
            stock.updatePrice();
        }
    }
```

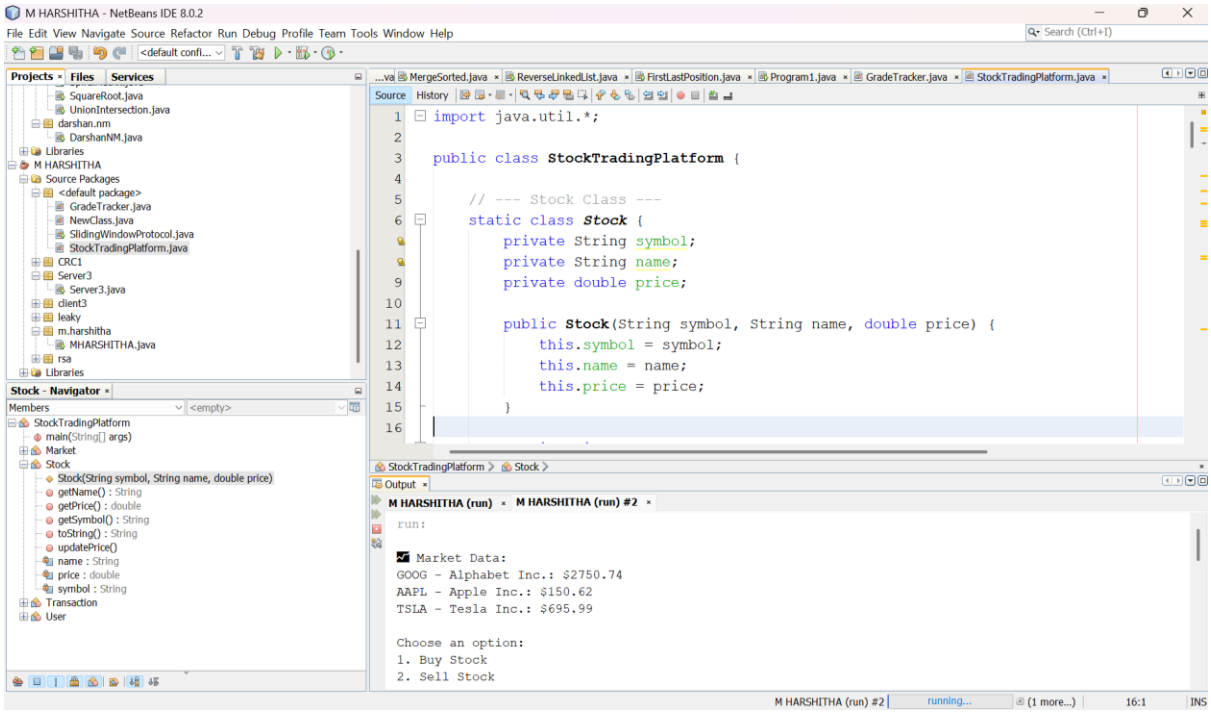
```
    public void displayMarket() {
        System.out.println("\n📈 Market Data:");
        for (Stock stock : stocks.values()) {
            System.out.println(stock);
        }
    }

    public Stock getStock(String symbol) {
        return stocks.get(symbol);
    }
}

// --- Transaction Class ---
static class Transaction {
    private String symbol;
    private int quantity;
    private double price;
    private String type;

    public Transaction(String symbol, int quantity, double price, String type) {
        this.symbol = symbol;
        this.quantity = quantity;
        this.price = price;
        this.type = type;
    }

    @Override
    public String toString() {
        return type.toUpperCase() + ": " + quantity + " shares of " + symbol + " @ $" + price;
```

```
        }
    }

    // --- User Class ---
    static class User {
        private String name;
        private Map<String, Integer> portfolio = new HashMap<>();
        private List<Transaction> transactions = new ArrayList<>();

        public User(String name) {
            this.name = name;
        }

        public void buyStock(Market market, String symbol, int quantity) {
            Stock stock = market.getStock(symbol);
            if (stock != null) {
                portfolio.put(symbol, portfolio.getOrDefault(symbol, 0) + quantity);
                transactions.add(new Transaction(symbol, quantity, stock.getPrice(), "buy"));
                System.out.println("✅ Bought " + quantity + " shares of " + symbol + " @ $" +
stock.getPrice());
            } else {
                System.out.println("❌ Stock not found.");
            }
        }

        public void sellStock(Market market, String symbol, int quantity) {
            if (portfolio.getOrDefault(symbol, 0) >= quantity) {
                Stock stock = market.getStock(symbol);
                portfolio.put(symbol, portfolio.get(symbol) - quantity);
```

```
            transactions.add(new Transaction(symbol, quantity, stock.getPrice(), "sell"));
            System.out.println("✅ Sold " + quantity + " shares of " + symbol + " @ $" +
stock.getPrice());
        } else {
            System.out.println("❌ Not enough shares to sell.");
        }
    }

    public void viewPortfolio(Market market) {
        System.out.println("\n📊 Portfolio:");
        double total = 0;
        for (String symbol : portfolio.keySet()) {
            int qty = portfolio.get(symbol);
            double price = market.getStock(symbol).getPrice();
            double value = qty * price;
            total += value;
            System.out.println(symbol + ": " + qty + " shares @ $" + price + " → $" + value);
        }
        System.out.println("Total Portfolio Value: $" + total);
    }

    public void viewTransactions() {
        System.out.println("\n📜 Transaction History:");
        for (Transaction t : transactions) {
            System.out.println(t);
        }
    }
}
```

```
// --- Main Method ---
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Market market = new Market();
    User user = new User("Harshitha");

    while (true) {
        market.updateMarket();
        market.displayMarket();

        System.out.println("\nChoose an option:");
        System.out.println("1. Buy Stock");
        System.out.println("2. Sell Stock");
        System.out.println("3. View Portfolio");
        System.out.println("4. View Transactions");
        System.out.println("5. Exit");
        System.out.print("Enter choice: ");
        int choice = sc.nextInt();

        switch (choice) {
            case 1:
                System.out.print("Enter stock symbol: ");
                String buySymbol = sc.next().toUpperCase();
                System.out.print("Enter quantity: ");
                int buyQty = sc.nextInt();
                user.buyStock(market, buySymbol, buyQty);
                break;
            case 2:
                System.out.print("Enter stock symbol: ");
```

```
                String sellSymbol = sc.next().toUpperCase();
                System.out.print("Enter quantity: ");
                int sellQty = sc.nextInt();
                user.sellStock(market, sellSymbol, sellQty);
                break;
            case 3:
                user.viewPortfolio(market);
                break;
            case 4:
                user.viewTransactions();
                break;
            case 5:
                System.out.println("👋 Exiting...");
                return;
            default:
                System.out.println("❌ Invalid choice.");
        }
    }
}
```



Output 2:

```
run:

📈 Market Data:
GOOG - Alphabet Inc.: $2750.74
AAPL - Apple Inc.: $150.62
TSLA - Tesla Inc.: $695.99

Choose an option:
1. Buy Stock
2. Sell Stock
3. View Portfolio
4. View Transactions
5. Exit
Enter choice: 2
Enter stock symbol: APPL
Enter quantity: 2
❌ Not enough shares to sell.

📈 Market Data:
GOOG - Alphabet Inc.: $2621.89
AAPL - Apple Inc.: $150.25
TSLA - Tesla Inc.: $723.52

Choose an option:
1. Buy Stock
2. Sell Stock
3. View Portfolio
4. View Transactions
5. Exit
```

TASK 3

Hotel Reservation System:

● Design a system to search, book, and manage hotel rooms.

● Add room categorization (e.g., Standard, Deluxe, Suite)

● Allow users to make and cancel reservations.

● Implement payment simulation and booking details view.

● Use OOP + database/File I/O for storing bookings and availability.

Program 3:

```
import java.io.*;
import java.util.*;

class Room {
    int roomNumber;
    String category;
    boolean isAvailable;

    Room(int roomNumber, String category) {
        this.roomNumber = roomNumber;
        this.category = category;
        this.isAvailable = true;
    }
}

class Booking {
    String customerName;
    int roomNumber;
    String category;
    String bookingDate;
```

```
    boolean isPaid;

    Booking(String customerName, int roomNumber, String category, String bookingDate) {
        this.customerName = customerName;
        this.roomNumber = roomNumber;
        this.category = category;
        this.bookingDate = bookingDate;
        this.isPaid = false;
    }
}

public class HotelReservationSystem {
    static List<Room> rooms = new ArrayList<>();
    static List<Booking> bookings = new ArrayList<>();
    static Scanner sc = new Scanner(System.in);

    public static void main(String[] args) {
        loadRooms();
        loadBookings();

        while (true) {
            System.out.println("\n--- Hotel Reservation System ---");
            System.out.println("1. Search Available Rooms");
            System.out.println("2. Book a Room");
            System.out.println("3. Cancel Booking");
            System.out.println("4. View Booking Details");
            System.out.println("5. Simulate Payment");
            System.out.println("6. Exit");
            System.out.print("Choose an option: ");
```

```
        int choice = sc.nextInt();
        sc.nextLine(); // consume newline

        switch (choice) {
            case 1:
                searchRooms();
                break;
            case 2:
                bookRoom();
                break;
            case 3:
                cancelBooking();
                break;
            case 4:
                viewBooking();
                break;
            case 5:
                simulatePayment();
                break;
            case 6:
                saveBookings();
                System.out.println("Thank you for using the system!");
                return;
            default:
                System.out.println("Invalid choice.");
        }
    }
}
```

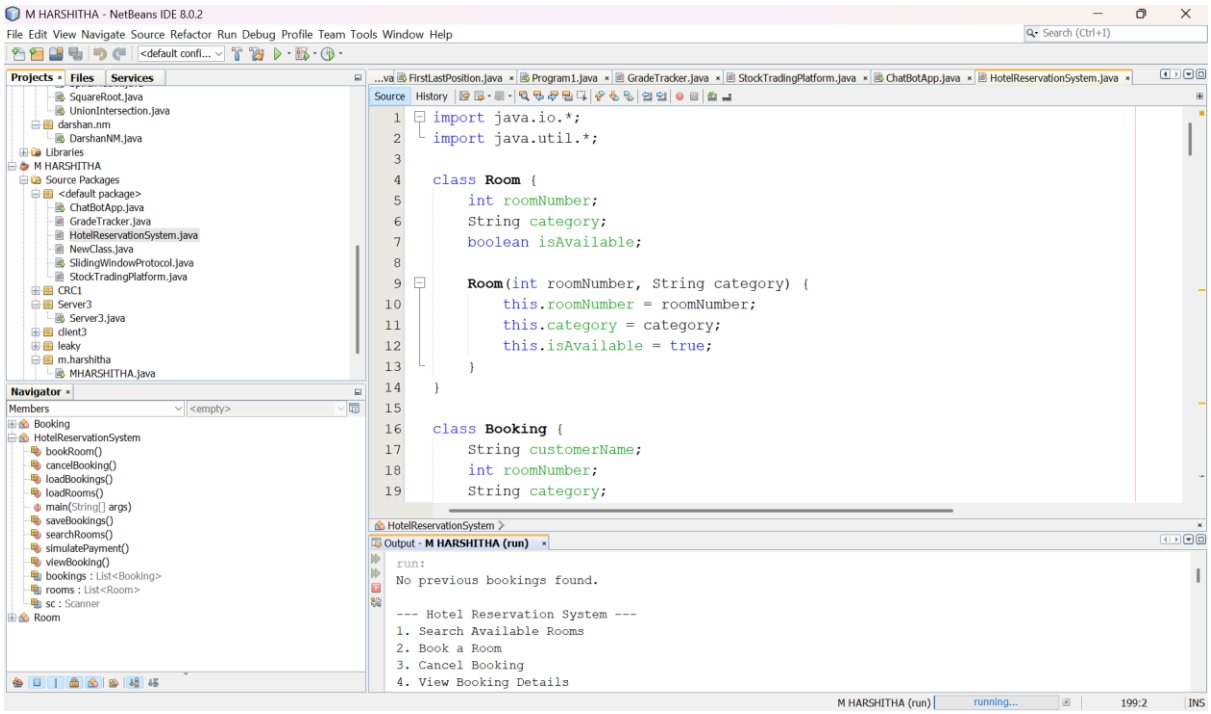
```
static void loadRooms() {
    rooms.add(new Room(101, "Standard"));
    rooms.add(new Room(102, "Standard"));
    rooms.add(new Room(201, "Deluxe"));
    rooms.add(new Room(202, "Deluxe"));
    rooms.add(new Room(301, "Suite"));
    rooms.add(new Room(302, "Suite"));
}

static void loadBookings() {
    try (BufferedReader br = new BufferedReader(new FileReader("bookings.txt"))) {
        String line;
        while ((line = br.readLine()) != null) {
            String[] p = line.split(",");
            Booking b = new Booking(p[0], Integer.parseInt(p[1]), p[2], p[3]);
            b.isPaid = Boolean.parseBoolean(p[4]);
            bookings.add(b);
            for (Room r : rooms) {
                if (r.roomNumber == b.roomNumber) r.isAvailable = false;
            }
        }
    } catch (IOException e) {
        System.out.println("No previous bookings found.");
    }
}

static void saveBookings() {
    try (BufferedWriter bw = new BufferedWriter(new FileWriter("bookings.txt"))) {
        for (Booking b : bookings) {
```

```
            bw.write(b.customerName + "," + b.roomNumber + "," + b.category + "," +
b.bookingDate + "," + b.isPaid);
            bw.newLine();
        }
    } catch (IOException e) {
        System.out.println("Error saving bookings.");
    }
}

static void searchRooms() {
    System.out.print("Enter room category (Standard/Deluxe/Suite): ");
    String category = sc.nextLine();
    boolean found = false;
    for (Room r : rooms) {
        if (r.category.equalsIgnoreCase(category) && r.isAvailable) {
            System.out.println("Available: Room " + r.roomNumber);
            found = true;
        }
    }
    if (!found) System.out.println("No available rooms in this category.");
}

static void bookRoom() {
    System.out.print("Enter your name: ");
    String name = sc.nextLine();
    System.out.print("Enter room category (Standard/Deluxe/Suite): ");
    String category = sc.nextLine();
    System.out.print("Enter booking date (YYYY-MM-DD): ");
    String date = sc.nextLine();
```

```
    for (Room r : rooms) {
        if (r.category.equalsIgnoreCase(category) && r.isAvailable) {
            r.isAvailable = false;
            Booking b = new Booking(name, r.roomNumber, category, date);
            bookings.add(b);
            saveBookings();
            System.out.println("Room " + r.roomNumber + " booked successfully.");
            return;
        }
    }
    System.out.println("No available rooms in this category.");
}

static void cancelBooking() {
    System.out.print("Enter your name to cancel booking: ");
    String name = sc.nextLine();
    Iterator<Booking> it = bookings.iterator();
    boolean found = false;
    while (it.hasNext()) {
        Booking b = it.next();
        if (b.customerName.equalsIgnoreCase(name)) {
            it.remove();
            for (Room r : rooms) {
                if (r.roomNumber == b.roomNumber) r.isAvailable = true;
            }
            saveBookings();
            System.out.println("Booking cancelled.");
            found = true;
```

```
            break;
        }
    }
    if (!found) System.out.println("No booking found for " + name);
}

static void viewBooking() {
    System.out.print("Enter your name to view booking: ");
    String name = sc.nextLine();
    for (Booking b : bookings) {
        if (b.customerName.equalsIgnoreCase(name)) {
            System.out.println("Room: " + b.roomNumber);
            System.out.println("Category: " + b.category);
            System.out.println("Date: " + b.bookingDate);
            System.out.println("Paid: " + b.isPaid);
            return;
        }
    }
    System.out.println("No booking found.");
}

static void simulatePayment() {
    System.out.print("Enter your name to pay: ");
    String name = sc.nextLine();
    for (Booking b : bookings) {
        if (b.customerName.equalsIgnoreCase(name)) {
            b.isPaid = true;
            saveBookings();
            System.out.println("Payment successful.");
```

```
                return;
            }
        }
        System.out.println("No booking found.");
    }
}
```



Output 3:

```
run:
No previous bookings found.

--- Hotel Reservation System ---
1. Search Available Rooms
2. Book a Room
3. Cancel Booking
4. View Booking Details
5. Simulate Payment
6. Exit
Choose an option: 1
Enter room category (Standard/Deluxe/Suite): Standard
Available: Room 101
Available: Room 102

--- Hotel Reservation System ---
1. Search Available Rooms
2. Book a Room
3. Cancel Booking
4. View Booking Details
5. Simulate Payment
6. Exit
Choose an option: 2
Enter your name: M Harshitha
Enter room category (Standard/Deluxe/Suite): Standard
Enter booking date (YYYY-MM-DD): (2025-10-01)
Room 101 booked successfully.
```