Manav Rachna International Institute of Research and Studies

SCHOOL OF COMPUTER APPLICATION

MCA - B 1st SEM

PROJECT REPORT

(Course Code: 6.0CA102C01H)

SUBMITTED BY:

Shiksha 25/SCA/MCAN/085

Hemant Kumar 25/SCA/MCAN/084

Vanshika 25/SCA/MCAN/ 086

SUBMITTED TO:

Dr. Ritu Sachdeva

# A parking system that includes Entry/exit logging, parking slot assignment, fee calculation in small malls.

Code: -

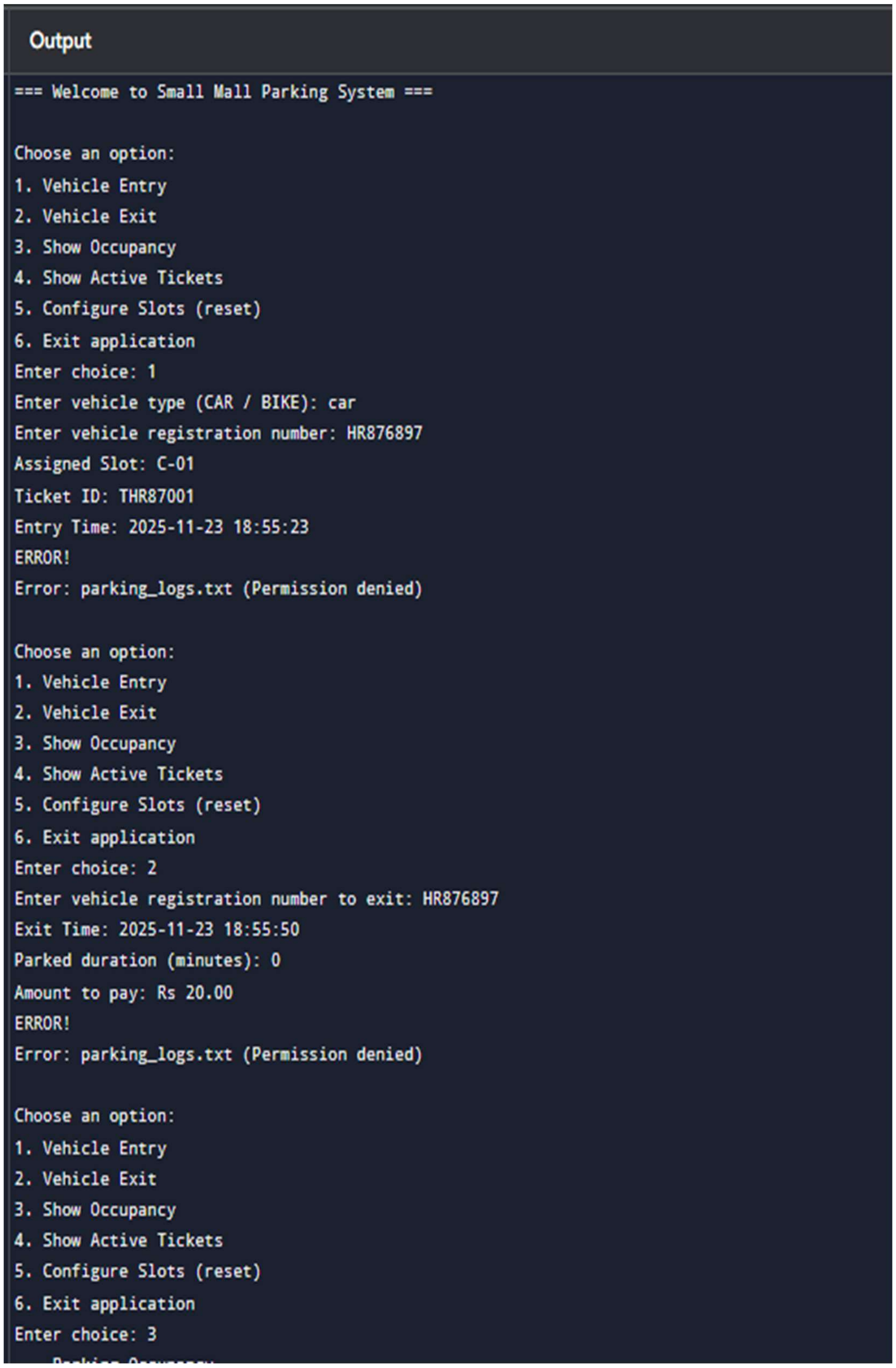
```
/*
 * ParkingSystem.java
 * Single-file Java console application that simulates a small-mall parking system.
 * Features:
 *  - Entry / Exit logging
 *  - Parking slot assignment (simple nearest-first strategy)
 *  - Fee calculation (by hour, with configurable rates and minimum fee)
 *  - Simple persistent logging to a text file (parking_logs.txt)
 *
 * How to run:
 * 1) Save this file as ParkingSystem.java
 * 2) Compile: javac ParkingSystem.java
 * 3) Run: java ParkingSystem
 *
 * Notes:
 * - This is a simple in-memory system (tickets are stored while app runs). Logs are appended
to parking_logs.txt.
 * - Designed for learning and small demos. Extend with DB, REST API, or GUI as needed.
 */

import java.io.*;
import java.time.*;
import java.time.format.DateTimeFormatter;
import java.util.*;

public class ParkingSystem {

   static final double CAR_RATE_PER_HOUR = 40.0;
   static final double BIKE_RATE_PER_HOUR = 15.0;
   static final double MINIMUM_FEE = 20.0;

   static final String LOG_FILE = "parking_logs.txt";

   public static void main(String[] args) {
      ParkingLot lot = new ParkingLot(20, 10);
      Scanner sc = new Scanner(System.in);
```

```
System.out.println("=== Welcome to Small Mall Parking System ===");

boolean running = true;
while (running) {
  System.out.println("\nChoose an option:");
  System.out.println("1. Vehicle Entry");
  System.out.println("2. Vehicle Exit");
  System.out.println("3. Show Occupancy");
  System.out.println("4. Show Active Tickets");
  System.out.println("5. Configure Slots (reset)");
  System.out.println("6. Exit application");
  System.out.print("Enter choice: ");

  String choice = sc.nextLine().trim();
  try {
    switch (choice) {
      case "1":
        doEntry(lot, sc);
        break;
      case "2":
        doExit(lot, sc);
        break;
      case "3":
        lot.printStatus();
        break;
      case "4":
        lot.printActiveTickets();
        break;
      case "5":
        System.out.print("Enter car slots: ");
        int c = Integer.parseInt(sc.nextLine());
        System.out.print("Enter bike slots: ");
        int b = Integer.parseInt(sc.nextLine());
        lot = new ParkingLot(c, b);
        System.out.println("Parking lot reset.");
        break;
      case "6":
        running = false;
        System.out.println("Exiting. Bye!");
        break;
      default:
        System.out.println("Invalid choice. Try again.");
    }
  } catch (Exception ex) {
```

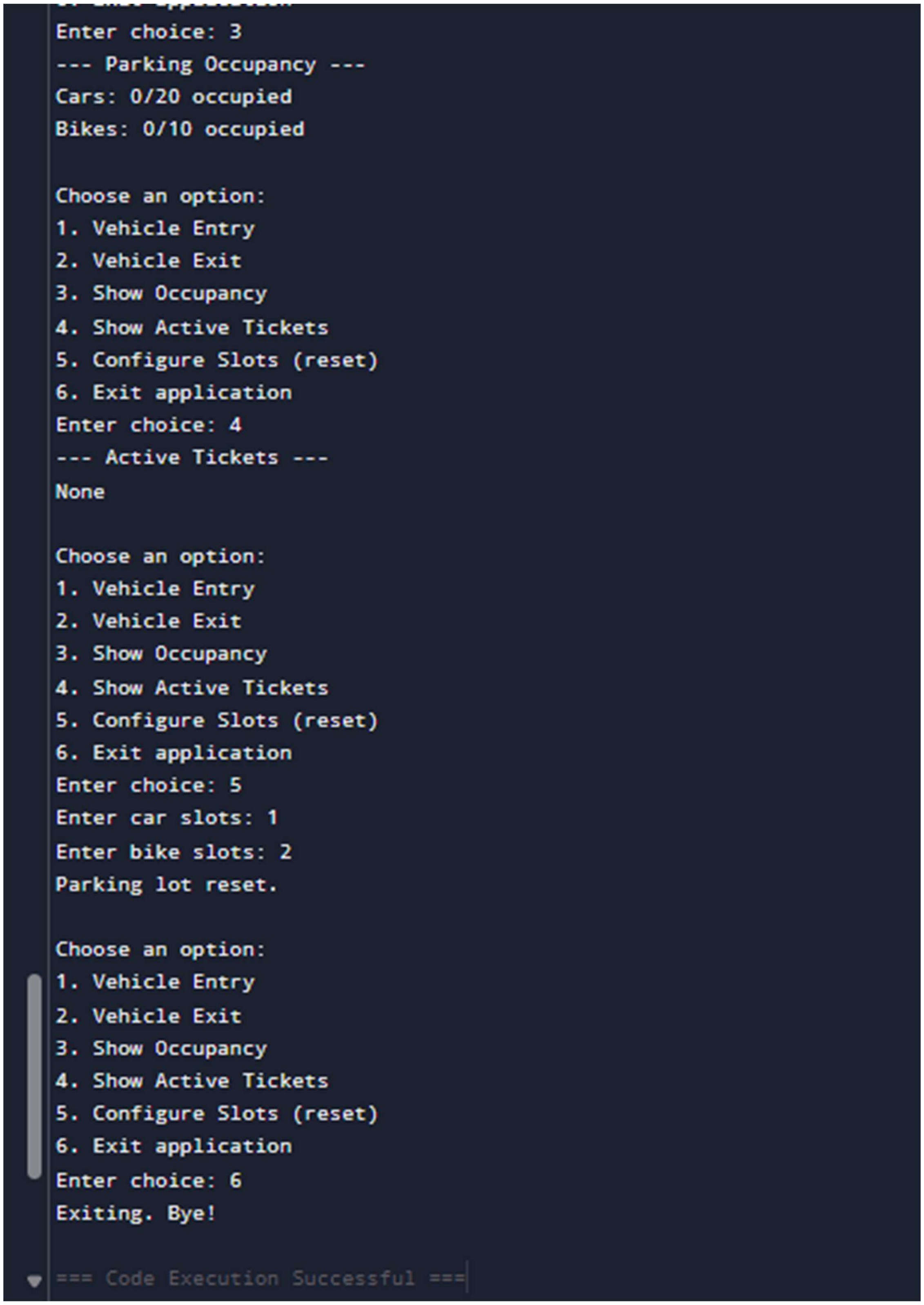
```
            System.out.println("Error: " + ex.getMessage());
        }
    }

    sc.close();
}

static void doEntry(ParkingLot lot, Scanner sc) throws IOException {
    System.out.print("Enter vehicle type (CAR / BIKE): ");
    String typeStr = sc.nextLine().trim().toUpperCase();
    VehicleType type;
    if (typeStr.equals("CAR")) type = VehicleType.CAR;
    else if (typeStr.equals("BIKE")) type = VehicleType.BIKE;
    else {
        System.out.println("Unknown type. Use CAR or BIKE.");
        return;
    }

    System.out.print("Enter vehicle registration number: ");
    String reg = sc.nextLine().trim().toUpperCase();
    if (reg.isEmpty()) {
        System.out.println("Registration cannot be empty.");
        return;
    }

    Ticket t = lot.entryVehicle(new Vehicle(reg, type));
    if (t == null) {
        System.out.println("Parking full for this vehicle type.");
        return;
    }

    System.out.println("Assigned Slot: " + t.slot.slotId);
    System.out.println("Ticket ID: " + t.ticketId);
                              System.out.println("Entry        Time:        "        +
t.entryTime.format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")));

    log(String.format("ENTRY | Ticket:%s | Reg:%s | Type:%s | Slot:%s | Time:%s\n",
            t.ticketId, t.vehicle.registrationNumber, t.vehicle.type, t.slot.slotId,
            t.entryTime.format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss"))));
}

static void doExit(ParkingLot lot, Scanner sc) throws IOException {
    System.out.print("Enter vehicle registration number to exit: ");
    String reg = sc.nextLine().trim().toUpperCase();
```

```
        if (reg.isEmpty()) {
            System.out.println("Registration cannot be empty.");
            return;
        }

        Ticket t = lot.findActiveTicketByReg(reg);
        if (t == null) {
            System.out.println("No active ticket found for this registration.");
            return;
        }

        LocalDateTime exitTime = LocalDateTime.now();
        long minutesParked = Duration.between(t.entryTime, exitTime).toMinutes();
        double amount = FeeCalculator.calculateFee(t.vehicle.type, minutesParked);

        lot.exitVehicle(t.ticketId);

         System.out.println("Exit Time: " + exitTime.format(DateTimeFormatter.ofPattern("yyyy-
MM-dd HH:mm:ss")));
        System.out.println("Parked duration (minutes): " + minutesParked);
        System.out.printf("Amount to pay: Rs %.2f\n", amount);

         log(String.format("EXIT  | Ticket:%s | Reg:%s | Type:%s | Slot:%s | Entry:%s | Exit:%s |
Minutes:%d | Amount:%.2f\n",
                t.ticketId, t.vehicle.registrationNumber, t.vehicle.type, t.slot.slotId,
                t.entryTime.format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")),
                exitTime.format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")),
                minutesParked, amount));
    }

    static void log(String text) throws IOException {
          try (FileWriter fw = new FileWriter(LOG_FILE, true); BufferedWriter bw = new
BufferedWriter(fw)) {
              bw.write(LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyy-MM-dd
HH:mm:ss")) + " | " + text);
        }
    }
}


enum VehicleType { CAR, BIKE }

class Vehicle {
    String registrationNumber;
    VehicleType type;
    public Vehicle(String reg, VehicleType t) {
        this.registrationNumber = reg;
        this.type = t;
    }
}

class Slot {
    String slotId;
    boolean occupied;
    VehicleType type;

    public Slot(String id, VehicleType t) {
        this.slotId = id;
        this.type = t;
        this.occupied = false;
    }
}

class Ticket {
    String ticketId;
    Vehicle vehicle;
    Slot slot;
    LocalDateTime entryTime;

    public Ticket(String id, Vehicle v, Slot s, LocalDateTime entry) {
        this.ticketId = id;
        this.vehicle = v;
        this.slot = s;
        this.entryTime = entry;
    }
}

class ParkingLot {
    List<Slot> carSlots = new ArrayList<>();
    List<Slot> bikeSlots = new ArrayList<>();

    Map<String, Ticket> activeTickets = new HashMap<>();
    Map<String, String> regToTicket = new HashMap<>();

    int carCount, bikeCount;

    public ParkingLot(int carCount, int bikeCount) {
        this.carCount = carCount;
        this.bikeCount = bikeCount;
          for (int i = 1; i <= carCount; i++) carSlots.add(new Slot(String.format("C-%02d", i),
VehicleType.CAR));
         for (int i = 1; i <= bikeCount; i++) bikeSlots.add(new Slot(String.format("B-%02d", i),
VehicleType.BIKE));
    }

    public Ticket entryVehicle(Vehicle v) {
        // check if reg already parked
        if (regToTicket.containsKey(v.registrationNumber)) {
                         System.out.println("Vehicle already inside with ticket: " +
regToTicket.get(v.registrationNumber));
            return null;
        }

        Slot free = findFreeSlot(v.type);
        if (free == null) return null;

        free.occupied = true;
        String ticketId = generateTicketId(v);
        Ticket t = new Ticket(ticketId, v, free, LocalDateTime.now());
        activeTickets.put(ticketId, t);
        regToTicket.put(v.registrationNumber, ticketId);
        return t;
    }

    public void exitVehicle(String ticketId) {
        Ticket t = activeTickets.remove(ticketId);
        if (t == null) return;
        t.slot.occupied = false;
        regToTicket.remove(t.vehicle.registrationNumber);
    }

    public Ticket findActiveTicketByReg(String reg) {
        String ticketId = regToTicket.get(reg);
        if (ticketId == null) return null;
        return activeTickets.get(ticketId);
    }

    private Slot findFreeSlot(VehicleType type) {
        List<Slot> list = (type == VehicleType.CAR) ? carSlots : bikeSlots;
        for (Slot s : list) if (!s.occupied) return s;
        return null;
    }
    private String generateTicketId(Vehicle v) {
        String shortReg = v.registrationNumber.replaceAll("[^A-Z0-9]", "");
        String id = String.format("T%s%03d", shortReg.length() > 4 ? shortReg.substring(0,4) :
shortReg, activeTickets.size()+1);
        return id;
    }

    public void printStatus() {
        long occupiedCars = carSlots.stream().filter(s -> s.occupied).count();
        long occupiedBikes = bikeSlots.stream().filter(s -> s.occupied).count();
        System.out.println("--- Parking Occupancy ---");
        System.out.printf("Cars: %d/%d occupied\n", occupiedCars, carCount);
        System.out.printf("Bikes: %d/%d occupied\n", occupiedBikes, bikeCount);
    }

    public void printActiveTickets() {
        System.out.println("--- Active Tickets ---");
        if (activeTickets.isEmpty()) System.out.println("None");
        else {
            for (Ticket t : activeTickets.values()) {
                System.out.printf("Ticket:%s | Reg:%s | Type:%s | Slot:%s | Entry:%s\n",
                        t.ticketId, t.vehicle.registrationNumber, t.vehicle.type, t.slot.slotId,
                        t.entryTime.format(DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss")));
            }
        }
    }
}

class FeeCalculator {

    public static double calculateFee(VehicleType type, long minutes) {
        if (minutes <= 0) return ParkingSystem.MINIMUM_FEE;
        long hours = minutes / 60;
        if (minutes % 60 != 0) hours += 1;

        double rate = (type == VehicleType.CAR) ? ParkingSystem.CAR_RATE_PER_HOUR :
ParkingSystem.BIKE_RATE_PER_HOUR;
        double amount = hours * rate;
        if (amount < ParkingSystem.MINIMUM_FEE) amount = ParkingSystem.MINIMUM_FEE;
        return amount;
    }
}
```

Output: -


Output

```
=== Welcome to Small Mall Parking System ===

Choose an option:
1. Vehicle Entry
2. Vehicle Exit
3. Show Occupancy
4. Show Active Tickets
5. Configure Slots (reset)
6. Exit application
Enter choice: 1
Enter vehicle type (CAR / BIKE): car
Enter vehicle registration number: HR876897
Assigned Slot: C-01
Ticket ID: THR87001
Entry Time: 2025-11-23 18:55:23
ERROR!
Error: parking_logs.txt (Permission denied)

Choose an option:
1. Vehicle Entry
2. Vehicle Exit
3. Show Occupancy
4. Show Active Tickets
5. Configure Slots (reset)
6. Exit application
Enter choice: 2
Enter vehicle registration number to exit: HR876897
Exit Time: 2025-11-23 18:55:50
Parked duration (minutes): 0
Amount to pay: Rs 20.00
ERROR!
Error: parking_logs.txt (Permission denied)

Choose an option:
1. Vehicle Entry
2. Vehicle Exit
3. Show Occupancy
4. Show Active Tickets
5. Configure Slots (reset)
6. Exit application
Enter choice: 3
```



```
6. Exit application
Enter choice: 3
--- Parking Occupancy ---
Cars: 0/20 occupied
Bikes: 0/10 occupied

Choose an option:
1. Vehicle Entry
2. Vehicle Exit
3. Show Occupancy
4. Show Active Tickets
5. Configure Slots (reset)
6. Exit application
Enter choice: 4
--- Active Tickets ---
None

Choose an option:
1. Vehicle Entry
2. Vehicle Exit
3. Show Occupancy
4. Show Active Tickets
5. Configure Slots (reset)
6. Exit application
Enter choice: 5
Enter car slots: 1
Enter bike slots: 2
Parking lot reset.

Choose an option:
1. Vehicle Entry
2. Vehicle Exit
3. Show Occupancy
4. Show Active Tickets
5. Configure Slots (reset)
6. Exit application
Enter choice: 6
Exiting. Bye!

=== Code Execution Successful ===
```