PARKNG LOT

```
public class ParkingLotDemo {
    public static void main(String[] args) throws InterruptedException {
        // Create address
        Address lotAddress = new Address("India", "Karnataka", "Bangalore", "MG
Road", "560001");

        // Create parking lot
        ParkingLot parkingLot = new ParkingLot("Central Parking", lotAddress);

        // Create floors with spaces for different vehicle types
        for (int i = 1; i <= 3; i++) {
            ParkingFloor floor = new ParkingFloor(i);
            // 5 car spaces, 3 bike spaces, 2 bus spaces per floor
            for (int j = 1; j <= 5; j++)
                floor.addParkingSpace(new ParkingSpace(j, 50.0,
ParkingSpaceType.CAR_PARKING));
            for (int j = 6; j <= 8; j++)
                floor.addParkingSpace(new ParkingSpace(j, 20.0,
ParkingSpaceType.BIKE_PARKING));
            for (int j = 9; j <= 10; j++)
                floor.addParkingSpace(new ParkingSpace(j, 100.0,
ParkingSpaceType.TRUCK_PARKING));
            parkingLot.parkingFloors.add(floor);
        }

        // Create parking attendants
        ParkingAttendant attendant1 = new ParkingAttendant("Alice",
"alice@example.com", "pass1", "EMP001", lotAddress);
        ParkingAttendant attendant2 = new ParkingAttendant("Bob",
"bob@example.com", "pass2", "EMP002", lotAddress);

        // Create gates and assign attendants
        Entrance entrance1 = new Entrance(1, attendant1);
        Entrance entrance2 = new Entrance(2, attendant2);
        Exit exit1 = new Exit(3, attendant1);
        Exit exit2 = new Exit(4, attendant2);

        parkingLot.entrances.add(entrance1);
        parkingLot.entrances.add(entrance2);
        parkingLot.exits.add(exit1);
        parkingLot.exits.add(exit2);

        // Simulate vehicles arriving
        Vehicle car = new Vehicle("KA-01-1234", VehicleType.CAR);
        Vehicle bike = new Vehicle("KA-02-5678", VehicleType.BIKE);
        Vehicle bus = new Vehicle("KA-03-9999", VehicleType.TRUCK);

        // Entry process: Each vehicle gets a ticket at an entrance
        ParkingTicket carTicket = entrance1.getParkingTicket(car, parkingLot);
        ParkingTicket bikeTicket = entrance2.getParkingTicket(bike, parkingLot);
        ParkingTicket busTicket = entrance1.getParkingTicket(bus, parkingLot);

        // Display available spots after entry
        System.out.println("Available spots after entry:");
        for (ParkingFloor floor : parkingLot.parkingFloors) {
            System.out.println("Floor " + floor.levelId + ": " +
floor.parkingDisplayBoard.freeSpotsAvailableMap);
        }
        // Simulate time passing (e.g., 2 hours for car, 1 hour for bike, 3
hours for bus)
        Thread.sleep(1000); // Simulate time (in real system, use proper time
tracking)
        carTicket.updateVehicleExitTime(new
Date(carTicket.vehicleEntryDateTime.getTime() + 2 * 60 * 60 * 1000));
        bikeTicket.updateVehicleExitTime(new
Date(bikeTicket.vehicleEntryDateTime.getTime() + 1 * 60 * 60 * 1000));
        busTicket.updateVehicleExitTime(new
Date(busTicket.vehicleEntryDateTime.getTime() + 3 * 60 * 60 * 1000));

        // Exit process: Each vehicle pays at an exit
        exit1.payForParking(carTicket, PaymentType.CASH, parkingLot);
        exit2.payForParking(bikeTicket, PaymentType.UPI, parkingLot);
        exit1.payForParking(busTicket, PaymentType.CREDIT_CARD, parkingLot);

        // Display available spots after exit
        System.out.println("\nAvailable spots after exit:");
        for (ParkingFloor floor : parkingLot.parkingFloors) {
            System.out.println("Floor " + floor.levelId + ": " +
floor.parkingDisplayBoard.freeSpotsAvailableMap);
        }

        // Print payment details
        System.out.println("\nPayment Details:");
        System.out.println("Car Ticket: " + carTicket.totalCost + ", Status: " +
carTicket.parkingTicketStatus);
        System.out.println("Bike Ticket: " + bikeTicket.totalCost + ", Status: "
+ bikeTicket.parkingTicketStatus);
        System.out.println("Bus Ticket: " + busTicket.totalCost + ", Status: " +
busTicket.parkingTicketStatus);
    }
}

//Address class
class Address {
 String country;
 String state;
 String city;
 String street;
 String pinCode;

 public Address(String country, String state, String city, String street, String
pinCode) {
     this.country = country;
     this.state = state;
     this.city = city;
     this.street = street;
     this.pinCode = pinCode;
 }
}

//Enums
enum PaymentType {
 CASH, CREDIT_CARD, DEBIT_CARD, UPI;
}
enum ParkingSpaceType {
 BIKE_PARKING, CAR_PARKING, TRUCK_PARKING;
}
enum VehicleType {
 BIKE, CAR, TRUCK;
}
```

```java
enum ParkingTicketStatus {
 PAID, ACTIVE;
}

enum PaymentStatus {
 UNPAID, PENDING, COMPLETED, DECLINED, CANCELLED, REFUNDED;
}

//Vehicle class
class Vehicle {
 String licenseNumber;
 VehicleType vehicleType;
 ParkingTicket parkingTicket;
 PaymentInfo paymentInfo;

 public Vehicle(String licenseNumber, VehicleType vehicleType) {
     this.licenseNumber = licenseNumber;
     this.vehicleType = vehicleType;
 }
}

//ParkingTicket class
class ParkingTicket {
 static int idCounter = 1;
 int ticketId;
 int levelId;
 int spaceId;
 Date vehicleEntryDateTime;
 Date vehicleExitDateTime;
 ParkingSpaceType parkingSpaceType;
 double totalCost;
 ParkingTicketStatus parkingTicketStatus;

 public ParkingTicket(int levelId, int spaceId, ParkingSpaceType
parkingSpaceType) {
     this.ticketId = idCounter++;
     this.levelId = levelId;
     this.spaceId = spaceId;
     this.parkingSpaceType = parkingSpaceType;
     this.vehicleEntryDateTime = new Date();
     this.parkingTicketStatus = ParkingTicketStatus.ACTIVE;
 }

 public void updateTotalCost(double costPerHour) {
     if (vehicleExitDateTime == null) return;
     long duration = vehicleExitDateTime.getTime() -
vehicleEntryDateTime.getTime();
     double hours = Math.ceil(duration / (1000.0 * 60 * 60));
     this.totalCost = hours * costPerHour;
 }

 public void updateVehicleExitTime(Date vehicleExitDateTime) {
     this.vehicleExitDateTime = vehicleExitDateTime;
 }
}

//ParkingSpace class
class ParkingSpace {
 int spaceId;
 boolean isFree;
 double costPerHour;
 Vehicle vehicle;
 ParkingSpaceType parkingSpaceType;

 public ParkingSpace(int spaceId, double costPerHour, ParkingSpaceType
parkingSpaceType) {
     this.spaceId = spaceId;
     this.isFree = true;
     this.costPerHour = costPerHour;
     this.parkingSpaceType = parkingSpaceType;
 }
}

//ParkingDisplayBoard class
class ParkingDisplayBoard {
 Map<ParkingSpaceType, Integer> freeSpotsAvailableMap = new HashMap<>();

 public void updateFreeSpotsAvailable(ParkingSpaceType parkingSpaceType, int
spaces) {
     freeSpotsAvailableMap.put(parkingSpaceType, spaces);
 }

 public int getFreeSpots(ParkingSpaceType type) {
     return freeSpotsAvailableMap.getOrDefault(type, 0);
 }
}

//ParkingFloor class
class ParkingFloor {
 int levelId;
 List<ParkingSpace> parkingSpaces = new ArrayList<>();
 ParkingDisplayBoard parkingDisplayBoard;

 public ParkingFloor(int levelId) {
     this.levelId = levelId;
     this.parkingDisplayBoard = new ParkingDisplayBoard();
 }

 public void addParkingSpace(ParkingSpace parkingSpace) {
     parkingSpaces.add(parkingSpace);
     updateDisplayBoard();
 }

 public void updateDisplayBoard() {
     Map<ParkingSpaceType, Integer> countMap = new HashMap<>();
     for (ParkingSpace space : parkingSpaces) {
         if (space.isFree) {
             countMap.put(space.parkingSpaceType,
countMap.getOrDefault(space.parkingSpaceType, 0) + 1);
         }
     }
     for (ParkingSpaceType type : ParkingSpaceType.values()) {
         parkingDisplayBoard.updateFreeSpotsAvailable(type,
countMap.getOrDefault(type, 0));
     }
 }

 public ParkingSpace getFreeSpace(ParkingSpaceType type) {
     for (ParkingSpace space : parkingSpaces) {
         if (space.isFree && space.parkingSpaceType == type) {
             return space;
         }
     }
     return null;
 }
}

//Account class
class Account {
 String name;
 String email;
 String password;
 String empId;
 Address address;

 public Account(String name, String email, String password, String empId,
Address address) {
     this.name = name;
     this.email = email;
     this.password = password;
     this.empId = empId;
     this.address = address;
 }
}

//Admin class
class Admin extends Account {
 public Admin(String name, String email, String password, String empId, Address
address) {
     super(name, email, password, empId, address);
 }

 public boolean addParkingFloor(ParkingLot parkingLot, ParkingFloor floor) {
     return parkingLot.parkingFloors.add(floor);
 }

 public boolean addParkingSpace(ParkingFloor floor, ParkingSpace parkingSpace) {
     floor.addParkingSpace(parkingSpace);
     return true;
 }

 public boolean addParkingDisplayBoard(ParkingFloor floor, ParkingDisplayBoard
parkingDisplayBoard) {
     floor.parkingDisplayBoard = parkingDisplayBoard;
     return true;
 }
}

//PaymentInfo class
class PaymentInfo {
 double amount;
 Date paymentDate;
 int transactionId;
 ParkingTicket parkingTicket;
 PaymentStatus paymentStatus;

 public PaymentInfo(double amount, int transactionId, ParkingTicket ticket,
PaymentStatus status) {
     this.amount = amount;
     this.paymentDate = new Date();
     this.transactionId = transactionId;
     this.parkingTicket = ticket;
     this.paymentStatus = status;
 }
}
```

```
//Payment class
class Payment {
 static int transactionCounter = 1;

 public PaymentInfo makePayment(ParkingTicket parkingTicket, PaymentType paymentType) {
     // For simplicity, always succeed
     PaymentInfo info = new PaymentInfo(parkingTicket.totalCost, transactionCounter++,
parkingTicket, PaymentStatus.COMPLETED);
     parkingTicket.parkingTicketStatus = ParkingTicketStatus.PAID;
     return info;
 }
}

//ParkingAttendant class
class ParkingAttendant extends Account {
 Payment paymentService;

 public ParkingAttendant(String name, String email, String password, String empId,
Address address) {
     super(name, email, password, empId, address);
     this.paymentService = new Payment();
 }

 public boolean processVehicleEntry(Vehicle vehicle) {
     // Implementation should be handled at Entrance
     return true;
 }

 public PaymentInfo processPayment(ParkingTicket parkingTicket, PaymentType paymentType)
{
     return paymentService.makePayment(parkingTicket, paymentType);
 }
}

//Gate class
class Gate {
 int gateId;
 ParkingAttendant parkingAttendant;

 public Gate(int gateId, ParkingAttendant attendant) {
     this.gateId = gateId;
     this.parkingAttendant = attendant;
 }
}

//Entrance class
class Entrance extends Gate {
 public Entrance(int gateId, ParkingAttendant attendant) {
     super(gateId, attendant);
 }

 public ParkingTicket getParkingTicket(Vehicle vehicle, ParkingLot parkingLot) {
     // Find a free space for the vehicle type
     for (ParkingFloor floor : parkingLot.parkingFloors) {
         ParkingSpaceType type = ParkingSpaceType.valueOf(vehicle.vehicleType +
"_PARKING");
         ParkingSpace space = floor.getFreeSpace(type);
         if (space != null) {
             space.isFree = false;
             space.vehicle = vehicle;
             floor.updateDisplayBoard();
             ParkingTicket ticket = new ParkingTicket(floor.levelId, space.spaceId,
space.parkingSpaceType);
             vehicle.parkingTicket = ticket;
             return ticket;
         }
     }
     return null; // No space available
 }
```

```java
}
//Exit class
class Exit extends Gate {
 public Exit(int gateId, ParkingAttendant attendant) {
     super(gateId, attendant);
 }
 public ParkingTicket payForParking(ParkingTicket parkingTicket, PaymentType
paymentType, ParkingLot parkingLot) {
     parkingTicket.updateVehicleExitTime(new Date());
     // Find floor and space
     for (ParkingFloor floor : parkingLot.parkingFloors) {
         if (floor.levelId == parkingTicket.levelId) {
             for (ParkingSpace space : floor.parkingSpaces) {
                 if (space.spaceId == parkingTicket.spaceId) {
                     parkingTicket.updateTotalCost(space.costPerHour);
                     PaymentInfo paymentInfo =
parkingAttendant.processPayment(parkingTicket, paymentType);
                     if (paymentInfo.paymentStatus == PaymentStatus.COMPLETED) {
                         space.isFree = true;
                         space.vehicle = null;
                         floor.updateDisplayBoard();
                     }
                     return parkingTicket;
                 }
             }
         }
     }
     return parkingTicket;
 }
}
//ParkingLot class
class ParkingLot {
 List<ParkingFloor> parkingFloors = new ArrayList<>();
 List<Entrance> entrances = new ArrayList<>();
 List<Exit> exits = new ArrayList<>();
 Address address;
 String parkingLotName;

 public ParkingLot(String parkingLotName, Address address) {
     this.parkingLotName = parkingLotName;
     this.address = address;
 }
 public boolean isParkingSpaceAvailableForVehicle(Vehicle vehicle) {
     ParkingSpaceType type = ParkingSpaceType.valueOf(vehicle.vehicleType + "_PARKING");
     for (ParkingFloor floor : parkingFloors) {
         if (floor.parkingDisplayBoard.getFreeSpots(type) > 0) {
             return true;
         }
     }
     return false;
 }
 public boolean updateParkingAttndant(ParkingAttendant parkingAttendant, int gateId) {
     for (Entrance entrance : entrances) {
         if (entrance.gateId == gateId) {
             entrance.parkingAttendant = parkingAttendant;
             return true;
         }
     }
     for (Exit exit : exits) {
         if (exit.gateId == gateId) {
             exit.parkingAttendant = parkingAttendant;
             return true;
         }
     }
     return false;
 }
}
```