***“Smart Parking System with Analytics”***



***DSA – Lab Project***

**GROUP MEMBERS:**

|  |  |
| --- | --- |
| Name | Registration Number |
| Areef ul Rehman | L1F23BSSE0389 |
| Talha Atif | L1F23BSSE0065 |
| Muzaffar Ali | L1F23BSSE0395 |
| Abdullah Maqbool | L1F23BSSE0391 |

***Section: P4***

***Submitted to:***

***Ma’am Javaria Tanveer***

**1. Introduction**

The Smart Parking System is a Java-based desktop application that manages vehicle entry and exit in a parking lot with support for VIP prioritization. It uses core data structures—Circular Queue for regular slots, Max Heap for VIP slots, and a LinkedList-based log—to ensure efficient operations. An Analytics Engine computes metrics such as average stay time and peak hour. A Swing GUI provides interaction for check-in, check-out, viewing parked vehicles, and displaying analytics.

**2. System Overview**

* **Total slots:** 1,000 (100 reserved for VIP)
* **Data structures:**
  + CircularQueue: regular slots
  + MaxHeap: VIP slots
  + LinkedList: entry/exit logs
  + HashMap: mapping vehicles to slots
* **Persistence:** Logs saved to parking\_logs.txt
* **GUI:** Java Swing with panels for entry, status, list of parked vehicles, and analytics

**3. Class Descriptions**

**3.1 Slot**

**Responsibility:** Represents a single parking slot.

* **Fields:**
  + int slotId
  + boolean isOccupied
  + boolean isVip
  + Vehicle currentVehicle
* **Constructors:**
  + Slot(int slotId, boolean isVip)
* **Methods:**
  + int getSlotId()
  + boolean isOccupied()
  + boolean isVip()
  + Vehicle getCurrentVehicle()
  + void occupy(Vehicle vehicle)
  + void free()
* **Time complexities:**
  + All getters & setters: O(1)
  + occupy / free: O(1)

**3.2 Vehicle**

**Responsibility:** Encapsulates vehicle data.

* **Fields:**
  + String licensePlate
  + boolean isVip
  + LocalDateTime entryTime
* **Constructors:**
  + Vehicle(String licensePlate, boolean isVip)
* **Methods:**
  + String getLicensePlate()
  + boolean isVip()
  + LocalDateTime getEntryTime()
* **Time complexities:** All O(1)

**3.3 CircularQueue**

**Responsibility:** Manages regular slots in FIFO order.

* **Fields:**
  + Slot[] slots
  + int front, rear, size, capacity
* **Constructors:**
  + CircularQueue(int capacity)
* **Methods:**
  + boolean enqueue(Slot slot) – add slot to rear
  + Slot dequeue() – remove from front
  + boolean isEmpty()
  + boolean isFull()
* **Time complexities:**
  + enqueue / dequeue / isEmpty / isFull: O(1)

**3.4 MaxHeap**

**Responsibility:** Manages VIP slots by smallest slotId first (min-heap behavior).

* **Fields:**
  + Slot[] heap
  + int size, capacity
* **Constructors:**
  + MaxHeap(int capacity)
* **Methods:**
  + boolean insert(Slot slot)
  + Slot extractMax() (i.e., extract slot with smallest ID)
  + private void heapify(int index)
  + private void swap(int i, int j)
* **Time complexities:**

|  |  |
| --- | --- |
| Operation | Time Complexity |
| insert | O(log n) |
| extractMax | O(log n) |
| heapify | O(log n) |

3.5 **EntryExitLog**

**Responsibility:** Maintains a persistent log of entries/exits.

* **Inner Class:**

LogEntry { String licensePlate; LocalDateTime entryTime, exitTime; boolean isVip; int slotId; }

* **Fields:**
  + LinkedList<LogEntry> logs
  + static final String LOG\_FILE
* **Constructors:**
  + EntryExitLog() – loads existing logs from file
* **Methods:**
  + private void loadLogsFromFile()
  + private void saveLogsToFile()
  + void addEntry(String licensePlate, boolean isVip, int slotId)
  + void addExit(String licensePlate)
  + LinkedList<LogEntry> getLogs()
* **Time complexities:**

|  |  |
| --- | --- |
| Operation | Time Complexity |
| Loading/saving file | O(m), where m = number of log entries |
| addEntry / addExit | O(m) in worst case (search) |

**3.6 AnalyticsEngine**

**Responsibility:** Computes parking metrics from logs.

* **Fields:**
  + EntryExitLog log
* **Constructors:**
  + AnalyticsEngine(EntryExitLog log)
* **Methods:**
  + double getAverageStayTime()
  + String getPeakHour()

**Formulas:**

1. **Average Stay Time**

* **Peak Hour**
* **Time complexities:**

|  |  |  |
| --- | --- | --- |
| Method Name | Time Complexity | Explanation |
| getAverageStayTime | **O(m)** | Iterates over m stay records |
| getPeakHour | **O (m + 24)** | Scans all m records and aggregates into 24 hours (one per hour of the day). |

**3.7 ParkingManager**

**Responsibility:** Coordinates slots, logs, and analytics.

* **Fields:**
  + CircularQueue regularSlots
  + MaxHeap vipSlots
  + HashMap<String, Slot> vehicleToSlot
  + EntryExitLog log
  + AnalyticsEngine analytics
* **Constructors:**
  + ParkingManager(int totalSlots, int vipSlotsCount) – initializes data structures
* **Methods:**

|  |  |
| --- | --- |
| Method | Time Complexity |
| checkInVehicle(String licensePlate, boolean isVip) | O(n) in worst case for duplicate-check |
| checkOutVehicle(String licensePlate) | O(n) search in vehicleToSlot keys |
| getAnalytics() | O(1) |
| getVehicleToSlot() | O(1) |
| getLog() | O(1) |

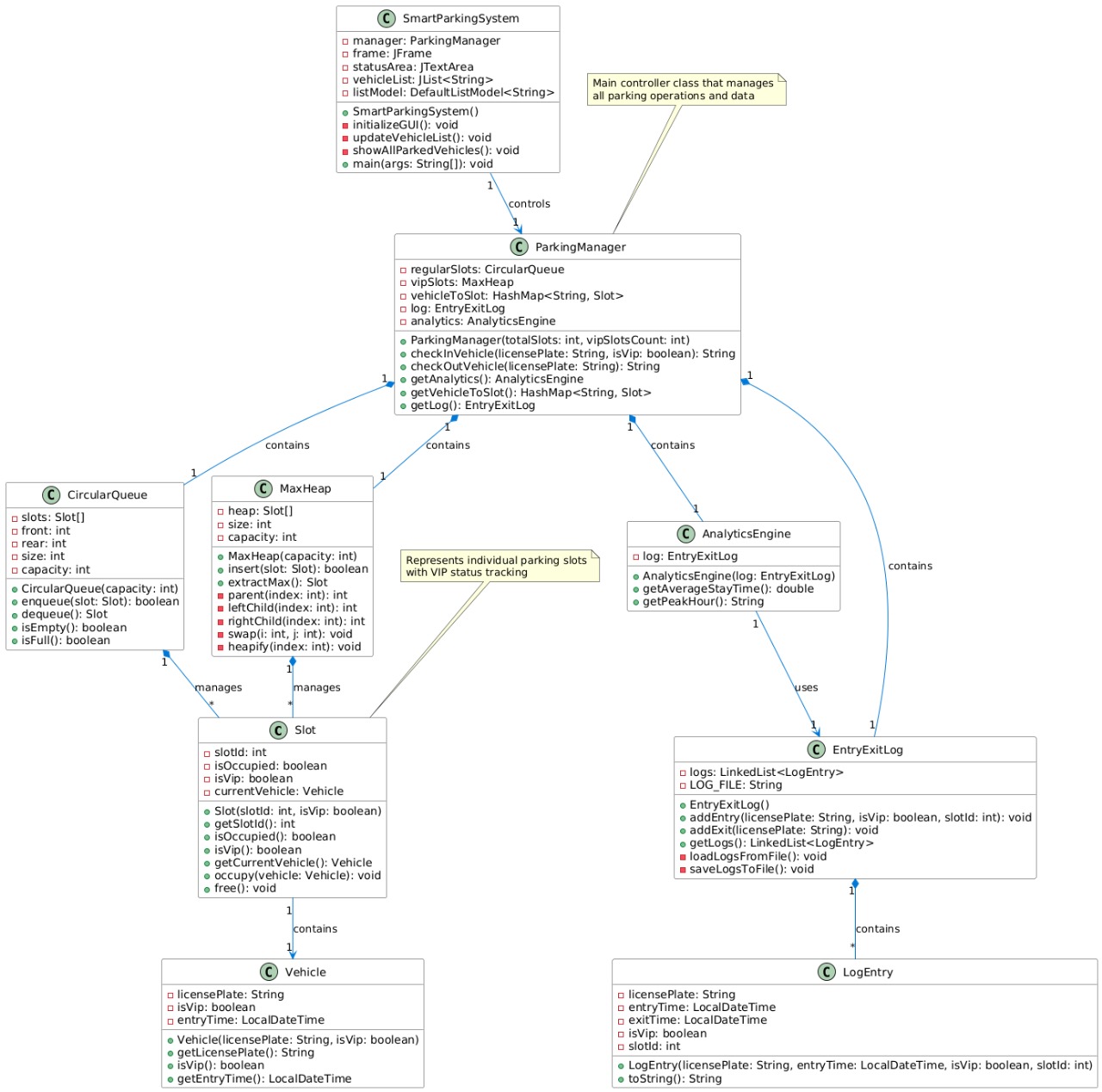
**3.8 SmartParkingSystem (Main GUI)**

**Responsibility:** Builds and runs the Swing interface.

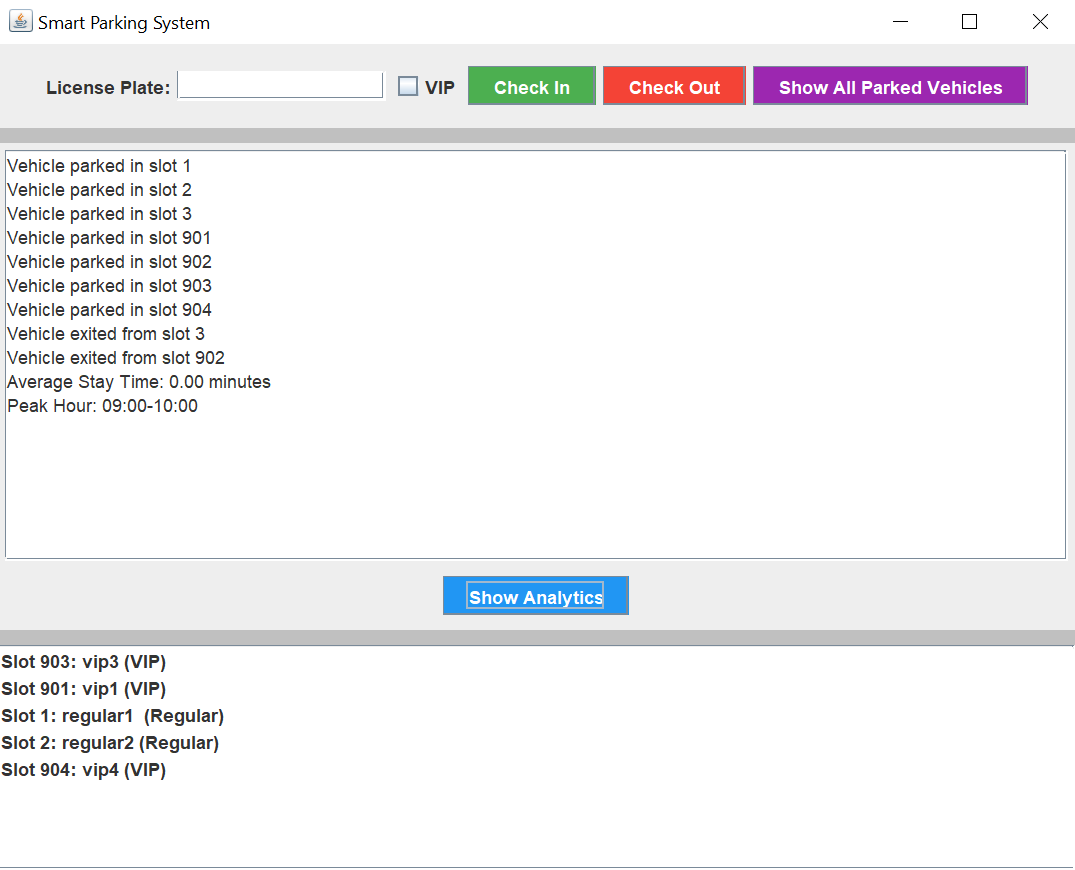
* **Fields:**
  + ParkingManager manager
  + JFrame frame
  + JTextArea statusArea
  + JList<String> vehicleList
  + DefaultListModel<String> listModel
* **Methods:**

|  |  |
| --- | --- |
| Method | Description / Time Complexity |
| SmartParkingSystem() | Constructor initializes ParkingManager(1000, 100) and GUI |
| initializeGUI() | Builds panels, buttons, list, event handlers |
| updateVehicleList() | Refreshes list of parked vehicles (**O(p)** where p = parked count) |
| showAllParkedVehicles() | Displays log in a JTable (**O(m)** to populate) |
| main(String[] args) | Launches application |

**4. Class Diagram**



**5. GUI Screenshots:**

***MAIN FRAME:***  


***PARKED VEHICLES FRAME:***

