Parking Lot Management Application

Pradyumna Girish Deshpande

Design & Problem Approach

- Design a Parking Lot Management System.
- CLI-based interaction
- Supports park, remove, display status operations
- Handles multiple vehicle sizes (small, large, oversize)
- Each vehicle maps to one of the slot sizes
- Vehicle parked in same size if available; else next bigger size
- Only one vehicle entry allowed at a time (handled using Map)

Technology Stack

- Java 17
- Maven (Build Tool)
- JUnit 5 (Unit Testing)
- SLF4J + Logback (Logging)
- Code Quality Tools: SpotBugs, Checkstyle, PMD
- Runnable JAR Packaging with Maven Shade Plugin

Key Files and Folder Structure

- Main.java – Entry point, CLI interface
- ParkingLotManager – Business logic handler
- Vehicle, SlotType – Domain models
- CustomExceptions – Domain-specific errors
- LoggerConfig – Logback configuration
- ParkingLotManagerTest - Unit tests
- logback.xml: Logging configuration
- pom.xml: Maven dependencies and plugins

How the Application Works (Overview)

- User sets total slots at start
- Split into SMALL, LARGE, OVERSIZE (extras → OVERSIZE)
- Park: Vehicle # + Size (1–3) → exact slot or larger fallback
- Remove: Free slot via vehicle #
- Show Status, Exit
- Reject empty vehicle numbers
- Clear success/failure messages
- Logs go to file (Logback)
- HashMap<String, SlotType> – fast removal
- EnumMap<SlotType, Integer> – slot tracking

Key Features

- Categorized Slot Management (SMALL, LARGE, OVERSIZE)
- Fallback Parking Logic (parks in next larger slot if needed)
- User-Friendly CLI (choice-based input loop)
- Custom Exception Handling (NoAvailableSlotException, VehicleNotFoundException)
- File-Based Logging with SLF4J + Logback (no System.out.println)
- Code Quality Tools: SpotBugs, PMD, Checkstyle (via Maven)
- JUnit 5 Unit Tests (covers edge and fallback cases)
- Modular & Testable Design (SRP-friendly structure)

Unit Testing Summary

- 10 total unit tests
- Tested initialization, parking, removal, overflow, duplication
- Verified slot count and internal map integrity
- Handled custom exceptions for failures
- Visual testing for display status

How to Run the JAR File

- Ensure Java 17+ is installed
- Find the JAR file in:

  ParkingLotApp/release/
- Open Terminal or Command Prompt
- Navigate to directory containing JAR file
- Run using command:

  ```
  java -jar ParkingLotApp-1.0-SNAPSHOT.jar
  ```
- (No setup or IDE required)

Run, Test & Verify the Application

- Build and run (In terminal) :

```
mvn clean install
mvn exec:java -Dexec.mainClass="com.parkinglot.Main"
```

- Alternatively, if `mainClass` is configured in `pom.xml` for `exec-maven-plugin`:

```
mvn clean install
mvn exec:java
```

- Ensure `exec-maven-plugin` is configured in `pom.xml`.
- Run all unit tests:

```
mvn test
```

- Bug-check: SpotBugs plugin integrated

Challenges & Trade-offs

- Serialization scrapped to maintain simplicity; suitable for future scope of the project
- Slot ID mapping removed due to not being in current project scope; better for future scope of the project
- Focus kept on simplicity, maintainability, and clarity

Future Scope

- Add state persistence using Java Serialization or JSON
- Dynamic Slot Configuration via properties file
- Vehicle-slot mapping with unique slot IDs for deeper functionality and analysis
- Web-based UI using Spring Boot or Angular
- Use Database for persistence in scalable deployments

Resources

- Source Code, README, JAR File, Presentation PDF available on GitHub: https://github.com/PradyumnaD2999/ParkingLotApp