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 chooses total slot count at startup
- Parking lot is divided 3-way into SMALL, LARGE, OVERSIZE (Extra slots go to OVERSIZE due to highest fallback)
- User can choose to: Park, Remove, Show Status, or Exit
- Park flow: vehicle number and vehicle size input → tries to find a valid slot
- Remove flow: uses vehicle number to free up slot

Demo & Working

- Vehicle number input by user (Any String)
- Vehicle size input via integer menu (1=Small, 2=Large, 3=Oversize)
- Null or empty vehicle numbers are rejected
- Logs are recorded in file (via logback), not console
- Graceful messages shown on success/failure

Slot Allocation Strategy

- Exact size preferred
- Fallback to larger size if exact slot unavailable
- Never fallback to smaller size
- HashMap<String, SlotType> used for quick removal
- EnumMap<SlotType, Integer> used for slot count tracking

Key Features

- Categorized slot management (SMALL, LARGE, OVERSIZE)
- Fallback parking logic (to larger slot type)
- Custom Exceptions and graceful error handling
- User-friendly CLI (choice-based loop)
- File-based logging with SLF4J + Logback
- Testable modular code

Extra Features Implemented

- Logging with SLF4J/Logback
- Code Quality Tools (SpotBugs, PMD, Checkstyle)
- JUnit 5 Unit Tests
- Custom Exception Handling
- CLI with User Choice Flow
- Slot Allocation Strategy

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