🎯 SOLID Principles Benchmark - Parking Lot System

Project: LLD-3 (Parking Lot System)
Date: January 23, 2026
Overall Score: 74/100 | 7.4/10 | Grade: B+

📊 Score Summary

Principle	Full Name	Score (/20)	Grade
S	Single Responsibility	15/20	B+
O	Open/Closed	12/20	B-
L	Liskov Substitution	16/20	A-
I	Interface Segregation	13/20	B
D	Dependency Inversion	18/20	A

📁 Project Structure Analyzed

```
LLD-3/
├── Controller/
│   └── Api.java
├── Model/Entity/
│   ├── Billing.java
│   └── Ticket.java
├── Repository/
│   ├── ParkingLotPlace.java (interface)
│   ├── ParkingLotPlace_Impl.java
│   ├── ParkingLotVehicle.java (interface)
│   ├── ParkingLotVehicle_Impl.java
│   ├── Tickets.java (interface)
│   └── Tickets_Impl.java
├── Service/
│   ├── AdminService.java (interface)
│   ├── ParkingService.java (interface)
│   └── Implementations/
│       ├── AdminServiceImpl.java
│       └── ParkingServiceImpl.java
├── View/
│   └── Frontend.java
├── util/
│   └── idgenerator.java
└── Main.java
```

🔍 Detailed Analysis

S - Single Responsibility Principle (15/20)

> *"A class should have only one reason to change."*

✅ **Strengths**

Component	Analysis
`Api.java`	Controller handles only user interaction flow
`Frontend.java`	View handles only display & input operations
`AdminServiceImpl.java`	Focused on admin operations only
`ParkingServiceImpl.java`	Focused on parking user operations
`Ticket.java` & `Billing.java`	Clean entity models with minimal logic

❌ **Areas for Improvement**

Issue	Location	Impact
Large method	`Api.Mainview()` - 142 lines with nested switch cases	Should be split into smaller handler methods
Multiple responsibilities	`ParkingServiceImpl.cancelTicket()`	Calculates charge + updates ticket + updates parking lot
Mixed concern	`Ticket.java` constructor	Generates own ID via `idgenerator` - mixing entity with ID generation

---

O - Open/Closed Principle (12/20)

> *"Software entities should be open for extension, but closed for modification."*

✅ **Strengths**

- Repository interfaces (`ParkingLotPlace` , `ParkingLotVehicle` , `Tickets`) allow new implementations
- Service interfaces (`AdminService` , `ParkingService`) enable extension without modification
- Clean abstraction allows swapping implementations

❌ **Areas for Improvement**

Issue	Impact
No strategy pattern for billing	Cost logic is hardcoded in `ParkingServiceImpl`
No enum/constants for floors	Adding new floor types requires code changes
Switch statements in `Api.java`	New features require modifying existing code
Hardcoded vehicle types	Adding new vehicle categories needs code changes

---

L - Liskov Substitution Principle (16/20)

> *"Objects of a superclass should be replaceable with objects of subclasses without breaking the application."*

✅ **Strengths**

Implementation	Interface	Status
`ParkingLotPlace_Impl`	`ParkingLotPlace`	✅ Correct
`ParkingLotVehicle_Impl`	`ParkingLotVehicle`	✅ Correct
`Tickets_Impl`	`Tickets`	✅ Correct
`AdminServiceImpl`	`AdminService`	✅ Correct
`ParkingServiceImpl`	`ParkingService`	✅ Correct

❌ **Areas for Improvement**

Issue	Location
Magic number return	`ParkingLotVehicle.getCost()` returns `-1` if not found
Null handling	Some methods could throw exceptions with null inputs

---

I - Interface Segregation Principle (13/20)

> *"Clients should not be forced to depend on interfaces they do not use."*

✅ **Strengths**

- `ParkingService` and `AdminService` are reasonably focused
- Separate interfaces for tickets, vehicles, and parking places

❌ **Areas for Improvement**

Interface	Issue
`ParkingLotPlace`	7 methods mixing user (`//user`) and admin (`//admin`) concerns
`AdminService`	Contains `get_avaliability()` which duplicates `ParkingService` method

💡 **Suggested Refactoring**

```
// Better Interface Segregation
interface ParkingPlaceReader {
    Map<String,Integer> get_avaliability();
    boolean check_capacity(String fname);
}

interface ParkingPlaceWriter {
    void add_capacity(String fname, int capacity);
    void reduce_capacity(String fname);
}

interface ParkingPlaceAdmin extends ParkingPlaceReader, ParkingPlaceWriter {
```

```
    Map<String,Integer> getfixed_avaliability();
}
```

D - Dependency Inversion Principle (18/20)

> *"High-level modules should not depend on low-level modules. Both should depend on abstractions."*

✅ Strengths (Excellent!)

Component	Depends On	Type
`Api.java`	`AdminService`, `ParkingService`	✅ Interfaces
`AdminServiceImpl`	`ParkingLotPlace`, `ParkingLotVehicle`	✅ Interfaces
`ParkingServiceImpl`	`Tickets`, `ParkingLotVehicle`, `ParkingLotPlace`	✅ Interfaces
`Main.java`	All dependencies constructor-injected	✅ Excellent DI

❌ Areas for Improvement

Issue	Location
Direct dependency	`Ticket.java` → concrete `idgenerator` class
Concrete types	`Frontend.java` depends on `Ticket` and `Billing` entities

📈 Final Scores

Scoring Scale

Score Range	Grade	Description
90-100	A+	Excellent adherence
80-89	A/A-	Strong adherence
70-79	B+/B	Good adherence
60-69	B-/C+	Moderate adherence
Below 60	C/D	Needs improvement

Your Results

Metric	Score
Total Points	74/100
Out of 10	7.4/10
Grade	**B+**
Status	✅ Good

🏆 Summary

What You Did Well ✅

1. **Excellent Dependency Injection** - All dependencies are injected via constructors in `Main.java`
2. **Clean Layered Architecture** - Controller → Service → Repository → Model
3. **Proper Abstraction** - Using interfaces for all major components
4. **MVC Pattern** - Clear separation between View, Controller, and Model/Service layers
5. **Liskov Compliance** - All implementations correctly honor their interfaces

Areas to Improve ⚠️

1. **Interface Segregation** - Split `ParkingLotPlace` into user and admin interfaces
2. **Single Responsibility** - Break down large methods like `Api.Mainview()`
3. **Extensibility** - Add strategy patterns for billing/charge calculation
4. **ID Generation** - Extract from `Ticket` constructor to a factory

---

🚀 Quick Wins for Better Scores

Priority 1: Split ParkingLotPlace Interface

```
interface UserParkingPlace {
    boolean check_capacity(String fname);
    Map<String,Integer> get_avaliability();
}

interface AdminParkingPlace extends UserParkingPlace {
    void add_capacity(String fname, int capacity);
    Map<String,Integer> getfixed_avaliability();
}
```

Priority 2: Extract Charge Calculator

```
interface ChargeCalculator {
    double calculate(LocalDateTime start, LocalDateTime end, int costPerHour);
}

class HourlyChargeCalculator implements ChargeCalculator {
    public double calculate(LocalDateTime start, LocalDateTime end, int costPerHour) {
        long hours = (Duration.between(start, end).toMinutes() + 59) / 60;
        return hours * costPerHour;
    }
}
```

Priority 3: Inject ID Generator into Ticket

```
interface IdGenerator {
    String generate(String prefix);
```

```
}

// Ticket constructor
public Ticket(IdGenerator idGen, Billing vtype, ...) {
    this.pid = idGen.generate(pfloor);
    // ...
}
```

Generated on: January 23, 2026