ParkZone Clean System Architecture

Simplified Java Client-Server Simulation

Core Design Principles

- **Single-threaded centralized coordinator** (ParkingServer)
- **Three distinct client applications** (Customer, Operator, Admin)
- **File-based persistence** (no database required)
- **TCP/IP socket communication** only
- **Simulated hardware operations** (no physical integration)

System Hierarchy

Server Layer (Single Application)

```
ParkingServer (main coordinator)
├── ParkingGarage (data management)
│   ├── ParkingFloor (floor management)
│   └── ParkingSpace (individual spaces)
├── ReservationManager (booking logic)
├── PaymentProcessor (fee calculations)
└── ClientConnectionManager (network handling)
```

Client Layer (Three Separate Applications)

```
CustomerClient.java    - Find spaces, make reservations, pay fees
OperatorClient.java    - Monitor garage, process payments, handle issues
AdminClient.java       - Configure system, manage floors, set pricing
```

Data Layer (File-Based Storage)

```
garage_state.dat       - Current space availability
reservations.dat       - Active and historical bookings
transactions.dat       - Payment records
system_config.dat      - Pricing rules and garage layout
```

Functional Scope (Realistic for Timeline)

Phase 1: Core Operations

1. **Space Management**
   - Track 50 parking spaces across 2 floors
   - Support 3 space types: Regular, Handicapped, Electric
   - Real-time availability tracking
2. **Basic Reservations**
   - Reserve space for up to 4 hours
   - 5-minute grace period implementation
   - Simple cancellation system
3. **Payment Simulation**
   - Hourly rate calculation ($5/hour flat rate)
   - Cash/credit simulation (no actual processing)
   - Receipt generation
4. **Client Communication**
   - Real-time space updates to all clients
   - Basic reservation confirmations
   - Status broadcasts

Phase 2: Enhanced Features (if time permits)

- User profiles and history
- Dynamic pricing by time/demand
- Overstay penalties
- Basic reporting

Class Architecture (Top 5 Core Classes)

1. ParkingServer

```
public class ParkingServer extends Thread {
    private ParkingGarage garage;
    private ArrayList<ClientHandler> clients;
    private ReservationManager reservations;

    // Centralized coordinator - all operations synchronized
    public synchronized void processReservation(ReservationRequest req)
    public synchronized void broadcastSpaceUpdate(int spaceId, String status)
    public synchronized void saveSystemState()
}
```

2. ParkingGarage

```
public class ParkingGarage {
    private ArrayList<ParkingFloor> floors;
    private HashMap<Integer, ParkingSpace> spaces;

    public synchronized boolean reserveSpace(int spaceId, String customerId)
    public synchronized ArrayList<ParkingSpace> getAvailableSpaces()
    public synchronized void releaseSpace(int spaceId)
}
```

3. ParkingSpace

```
public class ParkingSpace {
    public enum SpaceType { REGULAR, HANDICAPPED, ELECTRIC }
    public enum SpaceStatus { AVAILABLE, OCCUPIED, RESERVED }

    private int spaceId;
    private SpaceType type;
    private SpaceStatus status;
    private int floorNumber;
}
```

4. ReservationManager

```
public class ReservationManager {
    private ArrayList<Reservation> activeReservations;

    public synchronized Reservation createReservation(int spaceId, String customerId)
    public synchronized boolean checkGracePeriod(int reservationId)
    public synchronized void expireOverdueReservations()
}
```

5. PaymentProcessor

```
public class PaymentProcessor {
    public synchronized double calculateFee(Reservation reservation)
    public synchronized Receipt processPayment(PaymentRequest request)
    public synchronized boolean validatePayment(String paymentMethod)
}
```

Network Communication Protocol

Message Types (Serialized Objects)

```
// Client → Server
SpaceListRequest
ReservationRequest
PaymentRequest
CancelReservationRequest

// Server → Client
SpaceListResponse
ReservationConfirmation
PaymentReceipt
SpaceUpdateBroadcast
```

Connection Flow

1. Client connects via Socket to server port 8080
2. Server creates ClientHandler thread for each connection
3. All business logic processed through synchronized server methods
4. Server broadcasts updates to all connected clients immediately

---

File Storage Strategy

Simple Serialization Approach

```
// Save garage state
public void saveGarageState() {
    ObjectOutputStream out = new ObjectOutputStream(
        new FileOutputStream("garage_state.dat"));
    out.writeObject(garage);
    out.close();
}

// Load on server startup
public void loadGarageState() {
    ObjectInputStream in = new ObjectInputStream(
        new FileInputStream("garage_state.dat"));
    garage = (ParkingGarage) in.readObject();
    in.close();
}
```

Implementation Proposal (Project Phase 2)

Part 1: Core Infrastructure

- ParkingServer, basic socket handling
- ParkingGarage, ParkingSpace classes
- Basic client connection, simple space listing

Part 2: Business Logic & Integration

- Reservation system, payment processing
- Client GUIs (Swing-based)
- File persistence, testing, documentation

Success Criteria

Functional Demonstration

- Customer can view available spaces
- Customer can reserve and pay for space
- Operator can monitor garage status
- Admin can modify pricing/spaces
- All clients see real-time updates

Technical Requirements Met

- Java desktop application (no web components)
- TCP/IP client-server communication
- File-based data storage
- Single-threaded coordination
- No external libraries/databases

Risk Mitigation

Simplified Scope Decisions

- **50 spaces maximum** (not unlimited scaling)
- **2 floors only** (not complex multi-level)
- **3 space types** (not 10+ variations)
- **Flat hourly pricing** (not complex time-based rules)
- **Basic GUI** (Swing, not fancy interfaces)

Technical De-risking

- Start with console-based testing before GUIs
- Use simple object serialization instead of databases
- Implement core functionality before adding features
- Focus on working system over perfect user experience

This architecture provides a clear vision that the team can actually build within the constraints while demonstrating the core parking management concepts from Nathan's requirements and Sophia's detailed specifications.