ParkZone Phase 1 Presentation Outline

7-Minute Class Presentation - 9 Slides

Slide 1: Title Slide (30 seconds)

Speaker: Mario

Content:

- **Title:** ParkZone - Parking Garage Management System
- **Subtitle:** Software Requirements Specification - Phase 1
- **Team:** Group 7
  - Mario Salinas
  - Jose Ignacio Zaragoza
  - Sophia Ronquillo
- **Date:** [Presentation Date]

Speaker Notes:

"Good morning/afternoon. We're Group 7, and today we'll present ParkZone, our parking garage management system. This is a Java client-server application that simulates real-world parking operations."

Slide 2: Problem & Solution (60 seconds)

Speaker: Mario

Content:

The Problem:

- Finding parking is frustrating and time-consuming
- Garage operators lack real-time visibility into space availability
- Manual payment processing is inefficient

Our Solution: ParkZone provides real-time parking space tracking, automated reservations, and streamlined payment processing through a centralized management system.

What Makes It Work:

- Centralized server prevents double-booking
- Real-time updates to all connected clients
- Simplified scope fits 2-week timeline

Speaker Notes:

"Parking is a common frustration. Drivers waste time searching for spaces, and garage operators struggle with manual tracking. ParkZone solves this by providing real-time availability and automated management through a centralized coordinator that prevents conflicts."

Slide 3: System Architecture (60 seconds)

Speaker: Mario

Content:

Centralized Client-Server Model

Server Side:

- ParkingServer (centralized coordinator)
- Single-threaded business logic prevents race conditions
- TCP/IP communication on port 8080
- File-based persistence (no database required)

Client Side:

- CustomerClient: Find spaces, reserve, pay
- OperatorClient: Monitor, assist customers
- AdminClient: Configure system, generate reports

Key Design Decision: All business logic flows through synchronized server methods - one source of truth, no conflicts

Visual:

[Show network diagram: Server in center (yellow/orange), three client types (blue boxes) connected via arrows, file storage (purple) at bottom]

Speaker Notes:

"Our architecture uses a centralized coordinator. The server manages all operations through synchronized methods, so when multiple customers try to reserve the same space, the server processes requests sequentially and broadcasts updates immediately to all clients."

Slide 4: System Scope & Constraints (50 seconds)

Speaker: Sophia

Content:

What We're Building:

- 50 parking spaces across 2 floors
- 3 space types: Regular, Handicapped, Electric
- Reservation system with 5-minute grace period
- Payment simulation at $5/hour flat rate
- Real-time availability broadcasts

Technical Constraints:

- Java 11+ only (no external libraries)
- Desktop application (no web/mobile)
- File-based storage (no database)
- Simulated hardware (no physical integration)
- Local network operation

Speaker Notes:

"We've carefully scoped this project to fit Professor Smith's requirements and our timeline. We're focusing on core parking operations - 50 spaces, simple pricing, and essential features - all implemented in pure Java without databases or web frameworks."

Slide 5: Top 5 Classes (70 seconds)

Speaker: Sophia

Content:

Core Classes:

1. **ParkingServer**
   - Central coordinator managing all operations
   - Methods: processReservation(), broadcastUpdate(), saveState()
2. **ParkingGarage**
   - Manages 50 spaces across 2 floors
   - Methods: reserveSpace(), getAvailableSpaces(), releaseSpace()
3. **ParkingSpace**
   - Individual space with status, type, location
   - Attributes: spaceId, type (REGULAR/HANDICAPPED/ELECTRIC), status (AVAILABLE/RESERVED/OCCUPIED)
4. **ReservationManager**
   - Handles booking logic and validation
   - Methods: createReservation(), checkGracePeriod(), expireReservations()
5. **PaymentProcessor**
   - Calculates fees and processes transactions
   - Methods: calculateFee(), processPayment(), generateReceipt()

Visual:

[Show UML-style class boxes with key attributes and methods]

Speaker Notes:

"These five classes form our system backbone. ParkingServer coordinates everything, ParkingGarage manages the physical layout, ParkingSpace represents individual spots, and our business logic classes handle reservations and payments. All interactions flow through the server's synchronized methods."

Slide 6: Key Requirements (60 seconds)

Speaker: Sophia

Content:

Critical Functional Requirements:

- **SR-F203:** Unique space IDs (Floor-Number format: "1A-05")
- **SR-F302:** Reservations for 1-4 hours duration

- **SR-F304:** Auto-release after 5-minute grace period
- **SR-F401:** Flat rate $5.00 per hour
- **SR-F103:** Updates broadcast within 3 seconds

Non-Functional Requirements:

- **SR-N301:** Server response under 2 seconds
- **SR-N304:** Support 10 concurrent clients
- **SR-N205:** Java Swing GUI only
- **SR-N307:** 24-hour continuous operation

Total: 60+ documented requirements in our SRS

Speaker Notes:

"We've defined over 60 specific, testable requirements. These are the critical ones that define system behavior. Every requirement is numbered and traceable to our use cases and class design. The SRS document contains the complete specification."

Slide 7: Use Case Flow (70 seconds)

Speaker: Jose

Content:

Example: Customer Makes Reservation

Normal Flow:

1. Customer views available spaces filtered by type
2. System displays real-time availability
3. Customer selects space and duration (1-4 hours)
4. Server validates availability through synchronized method
5. Server creates reservation with unique ID
6. Server broadcasts update to ALL clients
7. Customer receives confirmation

Special Cases:

- Garage at 90% capacity → Warning message

- Space becomes unavailable → Suggest alternatives
- Grace period expires → Automatic release and broadcast

Visual:

[Show sequence diagram with Customer, Server, Garage, and other clients]

Speaker Notes:

"Let me walk through our most important use case. When a customer makes a reservation, the system validates availability, creates the booking, and immediately notifies all other connected clients. This coordination through the centralized server prevents double-booking and ensures everyone sees current availability."

Slide 8: Technical Implementation (60 seconds)

Speaker: Jose

Content:

Technology Stack:

- **Language:** Java 11+
- **GUI:** Java Swing
- **Networking:** TCP/IP Sockets (Port 8080)
- **Data Storage:** Java Object Serialization
- **Architecture:** Synchronized Coordinator Pattern

Communication Protocol:

- Serialized Java objects over sockets
- Message types: REQUEST, RESPONSE, BROADCAST, ERROR
- Automatic reconnection on disconnect
- File backup every 5 minutes

File Structure:

- garage_state.dat (space availability)
- reservations.dat (booking records)
- transactions.dat (payment history)

- system_config.dat (pricing/settings)

Speaker Notes:

"We're using pure Java as required. Communication happens through TCP/IP sockets with serialized objects - no web protocols needed. File-based storage keeps things simple but functional. The synchronized coordinator pattern eliminates race conditions without complex threading."

Slide 9: Timeline & Next Steps (50 seconds)

Speaker: Jose

Content:

Project Status:

Phase 1 (Complete):

- ✓ SRS Document (60+ requirements)
- ✓ Architecture design
- ✓ Class diagrams
- ✓ Use case specifications
- ✓ Sequence diagrams

Phase 2 (Weeks 3-4):

- Server infrastructure and networking
- Client GUI applications
- Business logic implementation
- File persistence system
- Testing and integration

Phase 3 (Week 5):

- Live demonstration with working system
- Final documentation

Team Responsibilities:

- **Mario:** Server & networking
- **Sophia:** Client GUIs

- **Jose:** Data management & business logic

Questions?

Speaker Notes:

"We've completed Phase 1 with comprehensive documentation. Next phase focuses on implementation - we'll build the server, three client applications, and integrate everything. In our final presentation, you'll see a live demonstration of customers making reservations, payments processing, and real-time updates across all clients. We're happy to answer questions. Thank you."

Presentation Delivery Notes

Time Management:

- Mario: Slides 1-3 (2 min 30 sec)
- Sophia: Slides 4-6 (3 min)
- Jose: Slides 7-9 (3 min)
- Buffer: 30 seconds for transitions
- Total: ~7 minutes

Transitions:

- Mario → Sophia: "And now Sophia will discuss our system scope and core classes"
- Sophia → Jose: "Jose will walk us through a use case and our implementation plan"

Visual Guidelines:

- Use consistent color scheme throughout
- Server/coordinator: Orange/Yellow
- Clients: Blue shades
- Data storage: Purple
- Minimum 24pt font for body text
- Diagrams on Slides 3, 5, 7

Rehearsal Checklist:

☐ Each person practices their 3 slides
☐ Timing verified (aim for 2-3 min per person)
☐ Smooth transitions between speakers

- [ ] Technical terms clearly defined
- [ ] Diagrams/visuals prepared
- [ ] SRS document ready to share
- [ ] GitHub link ready (if applicable)