Week 8: Detailed Design and Design Patterns

YZM2021 - Software Engineering Principles

Instructor: Ekrem Çetinkaya
Date: 2.12.2025

Where Are We?

What We've Covered:

- **Week 1-2:** Software Engineering & Principles (SOLID, DRY, KISS)
- **Week 3:** Process Models (Waterfall, RUP, Incremental)
- **Week 4:** Agile Methodologies (Scrum, XP, Kanban)
- **Week 5:** Requirements Engineering (Functional/Non-Functional, Elicitation)
- **Week 6:** System Modeling & UML (Use Case, Class, Sequence Diagrams)
- **Week 7:** Software Architecture (Patterns, Views, Components, Deployment)
- **Week 8:** 📝 **Midterm 1** ✓

Today:

- **Week 8 (9): Object-Oriented Design & Design Patterns**

Coming Up:

- **Week 9 (10):** Version Control and Code Organization
- **Week 11:** Software Testing

Object-Oriented Design (OOD)

What is an Object-Oriented System?

> An object-oriented system is made up of **interacting objects** that maintain their own **local state** and provide operations on that state.

Key Concepts:

- **Objects:** Self-contained entities with data and behavior
- **State:** Private data (cannot be accessed directly from outside)
- **Classes:** Blueprints defining objects and their interactions
- **Dynamic Creation:** Objects are created dynamically at runtime

Why OOD?

1. **Easier to Change:**
   - Objects are independent
   - Changing implementation doesn't affect other objects (if interface stays same)
2. **Maintainability:**
   - Clear mapping between real-world entities and code
   - "Wilderness Weather Station" → `WeatherStation` object

The Design Process

From Concept to Detailed Design

Sommerville identifies **5 key activities** in the OOD process:

1. **Understand the Context**
   - Define external interactions
   - What systems/users does it talk to?
2. **Design the Architecture**
   - High-level structure (Week 7)
   - How is the system distributed?
3. **Identify Principal Objects**
   - Map real-world entities to system objects
   - Start with the obvious ones
4. **Develop Design Models**
   - Create UML diagrams
   - Use Case, Class, Sequence diagrams (Week 6)
5. **Specify Interfaces**
   - Define method signatures
   - Contract between components

⚠️ Important Reality Check

Design is **creative** and **non-linear**. You don't do step 1, then 2, then 3 perfectly.
You **iterate**, **backtrack**, and **refine** as you learn more.

Example: Wilderness Weather Station

Context & Architecture

The System:
A system of weather stations deployed in remote wilderness areas.

Requirements:

- Record weather data (temp, wind, rain)
- Periodically transfer data via satellite
- Run on battery/solar power
- Self-diagnostic checks

1. System Context:

- **Actors:** Weather Station, Satellite Comms, Control Center, Maintenance Staff

2. Architecture:

- **Layered Architecture** (Sensors → Data Processing → Comms)
- **Client-Server** (Station pushes to Server)

Example: Wilderness Weather Station

Objects & Models

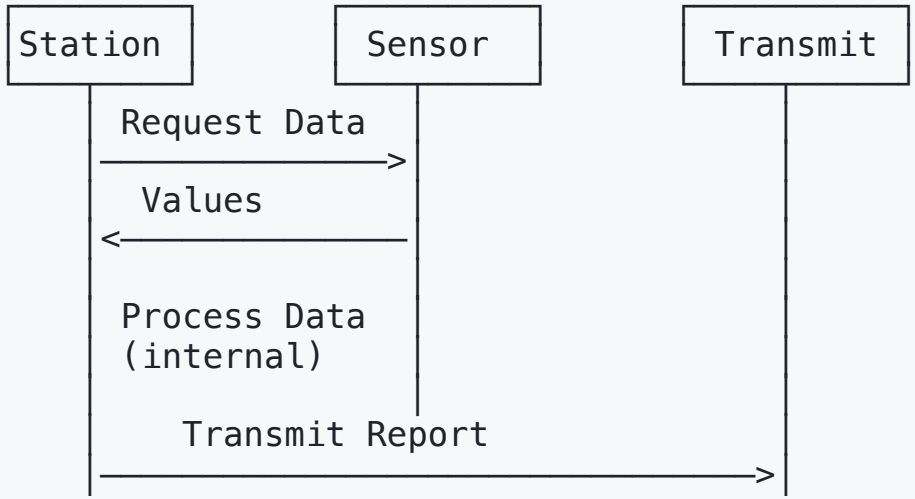
3. Principal Objects:

- `WeatherStation` (Controller)
- `WeatherData` (Entity)
- `GroundThermometer` (Sensor)
- `Anemometer` (Sensor)
- `SatelliteTransmitter` (Interface)

5. Interface Specification:

```
interface WeatherStation {
  reportWeather(): void;
  reportStatus(): void;
  powerDown(): void;
}
```

4. Design Models (Sequence):



Transition to Design Patterns

We have our Objects... Now what?

In the **Wilderness Weather Station**, we might face common problems:

1. **Creation:** We have different types of sensors. How do we create them flexibly?
    - *Hint: Factory Pattern*
2. **Notification:** When new weather data arrives, the display and the logger both need to know.
    - *Hint: Observer Pattern*
3. **Connection:** The satellite link might change (old satellite vs. new satellite).
    - *Hint: Adapter Pattern*
4. **Configuration:** The station has only *one* configuration manager.
    - *Hint: Singleton Pattern*

Design Patterns provide proven solutions to these recurring OOD problems.

What Are Design Patterns?

Definition

> **A design pattern is a reusable solution to a commonly occurring problem in software design.**

Design patterns are **templates** - not finished code, but proven approaches you can adapt to your specific situation.

Key Characteristics:

- **Proven:** Battle-tested in real-world projects
- **Reusable:** Apply to many different contexts
- **Language-independent:** Concepts work in any OOP language
- **Named:** Common vocabulary (e.g., "use Observer pattern")
- **Documented:** Structure, participants, consequences

What Patterns Are NOT:

❌ **Not** libraries or frameworks you import
❌ **Not** finished code you copy-paste
❌ **Not** algorithms (sorting, searching)
❌ **Not** always the best solution

A Brief History: The Gang of Four

The Book That Started It All

"Design Patterns: Elements of Reusable Object-Oriented Software"

Authors (Gang of Four):

- Erich Gamma
- Richard Helm
- Ralph Johnson
- John Vlissides

Published: 1994

Impact: Created a common language for software designers

The 23 Classic Patterns

Organized into three categories:

- **Creational** (5 patterns)
- **Structural** (7 patterns)
- **Behavioral** (11 patterns)

Why Patterns Matter

1. Communication

- "Let's use Factory here" (everyone understands)
- Better than explaining the entire approach

2. Proven Solutions

- Patterns solve real problems
- Avoid reinventing the wheel

3. Best Practices

- Patterns embody good OO design principles
- SOLID principles applied in practice

4. Flexibility

- Patterns make code easier to change
- Reduce coupling, increase cohesion

5. Career Skill

- Industry standard knowledge

The Three Categories of Patterns

Creational Patterns

Problem: How do we create objects?

Focus: Object instantiation mechanisms

Patterns We'll Cover:

1. **Singleton**
2. **Factory Method**
3. **Abstract Factory**
4. **Builder**
5. **Prototype**

Goal: Make object creation flexible and decoupled

Structural Patterns

Problem: How do we compose objects?

Focus: Relationships between entities

Patterns We'll Cover:

1. **Adapter**
2. **Decorator**
3. **Facade**
4. **Proxy**
5. **Composite**

Goal: Ensure flexibility in how objects work together

Behavioral Patterns

Problem: How do objects communicate?

Focus: Responsibilities and interactions

Patterns We'll Cover:

1. **Observer**
2. **Strategy**
3. **Template Method**
4. **Command**
5. **State**
6. **Iterator**
7. **Chain of Responsibility**

Goal: Make communication between objects flexible and maintainable