design-patterns # group-project

Introduction to Design Patterns

TCSS 305 Programming Practicum

Design patterns are reusable solutions to commonly occurring problems in software design. They are not finished code you can copy and paste, but blueprints that capture the accumulated wisdom of experienced developers. This guide introduces the concept of design patterns, traces their history from architecture to software engineering, and previews the four patterns demonstrated in the TCSS 305 group project codebase.

---

1 What is a Design Pattern?

A **design pattern** is a template for solving a problem that recurs across many different software projects. Think of design patterns like architectural patterns in building construction. If you are designing a house and want natural light, you do not reinvent the solution -- you use the well-established pattern of "windows on two sides of every room." The architect Christopher Alexander documented such patterns in his 1977 book *A Pattern Language*, showing that certain design problems recur across many buildings, and certain solutions work reliably.

Software design patterns work the same way. When you need to notify multiple objects about state changes, you do not invent a new mechanism -- you use the Observer pattern. When you need to ensure only one instance of a class exists, you use the Singleton pattern. These solutions have been proven over decades of software development.

> **Patterns are Tools, Not Rules**
>
> Design patterns are templates that you adapt to your specific situation. Do not force a pattern where it does not fit. The goal is to recognize recurring problems and know which pattern provides an elegant solution.

Design patterns provide:

- **A shared vocabulary** -- Saying "use the Observer pattern" communicates a complete design approach in three words

- **Proven solutions** -- These patterns have been tested and refined across countless projects
- **Best practices** -- They embody the accumulated wisdom of experienced developers
- **Flexibility** -- Patterns are templates, not rigid rules; you adapt them to your needs

---

2 A Brief History

The concept of design patterns in software engineering emerged in the 1990s, but the roots go back further.

2.1 The Inspiration: Christopher Alexander (1977)

Christopher Alexander, an architect and design theorist, published *A Pattern Language: Towns, Buildings, Construction*, which cataloged 253 architectural patterns for designing buildings and communities. His insight was profound: certain design problems recur, and certain solutions work better than others. By documenting these patterns, architects could build on collective experience rather than starting from scratch.

2.2 The Gang of Four (1994)

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (known as the "Gang of Four" or GoF) adapted Alexander's concept to software in their landmark book *Design Patterns: Elements of Reusable Object-Oriented Software*. Published in 1994, this book cataloged 23 fundamental design patterns organized into three categories:

Category	Focus	Examples
Creational Patterns	Object creation mechanisms	Singleton, Factory, Builder
Structural Patterns	Object composition	Adapter, Composite, Decorator
Behavioral Patterns	Object interaction and responsibility	Observer, Strategy, Command

The Gang of Four explicitly credited Alexander in their book's introduction, noting they were applying his architectural pattern concept to object-oriented software design.

2.3 Evolution and Modern Usage

Since 1994, the software community has:

- Identified many new patterns (Enterprise Integration Patterns, Concurrency Patterns, etc.)
- Adapted classic patterns to new programming paradigms (functional programming, reactive systems)
- Created pattern languages for specific domains (web development, distributed systems, mobile apps)
- Built frameworks and libraries that implement common patterns (Java's Observer framework, Spring's dependency injection, etc.)

> **Why Classic Patterns Still Matter**
>
> While newer paradigms like reactive programming have introduced alternatives, classic design patterns remain foundational because they teach principles -- separation of concerns, loose coupling, composition over inheritance -- that transcend any single pattern or programming style.

---

3 Patterns Demonstrated in This Codebase

The TCSS 305 group project implements several classic design patterns from the Gang of Four catalog. The guides below explain each pattern conceptually, then show how it is implemented in the game application.

3.1 Pattern Guides

1. **Observer Pattern** -- How objects can subscribe to state changes without tight coupling. The backbone of event-driven programming and the foundation of the MVC architecture. Demonstrates Java's `PropertyChangeListener` framework enhanced with type-safe sealed events.
2. **Model-View-Controller (MVC) Pattern** -- Separating business logic (Model), presentation (View), and user input handling (Controller) for maintainable, testable applications. Shows how Observer enables the Model to notify multiple Views, and explores the realistic complexity of controllers that handle multiple input types.

3. **Singleton Pattern** -- Ensuring only one instance of a class exists throughout the application. Demonstrates thread-safe lazy initialization with double-checked locking and the critical role of the `volatile` keyword in Java's memory model.
4. **Strategy Pattern** -- Encapsulating algorithms so they can be selected and swapped at runtime. Shows both the classic object-oriented approach and a modern functional implementation using Java's method references and `Map`-based strategy lookup.

> **Recommended Reading Order**
>
> Start with **Observer** if you are new to design patterns -- it is the most widely used pattern in GUI programming and forms the foundation for understanding MVC. Work through the guides in order, as later patterns reference earlier ones.

3.2 What Each Pattern Guide Covers

Each pattern guide follows a consistent structure:

Section	Purpose
Brief History	Origins and evolution of the pattern
The Problem It Solves	Non-technical, real-world analogies explaining why the pattern exists
How the Pattern Solves It	Conceptual explanation of the solution
Technical Implementation	Detailed code examples with line references from the codebase
Benefits and Tradeoffs	When to use the pattern and what it costs
Common Pitfalls	Mistakes students typically make
Related Patterns	How patterns connect and compose
Further Reading	Books, articles, and papers for deeper understanding

Each guide includes:

- **UML class diagrams** showing pattern structure

- **Code examples** from the actual codebase with specific line numbers
- **Relative links** to source files so you can explore the implementation

> **Gen AI & Learning: Design Patterns and AI Assistants**
>
> Design patterns give you a shared vocabulary that works with both human teammates and AI coding assistants. When you tell an AI tool "implement the Observer pattern here," it immediately understands the structure you want -- the subject, the observers, the notification mechanism. Without that vocabulary, you would need to describe the entire architecture from scratch every time. Learning patterns makes you more effective at communicating design intent, whether to a colleague or to an AI.

Summary

Concept	Key Point
Design Pattern	A reusable template for solving a recurring software design problem
Christopher Alexander	Architect who originated the pattern concept in 1977 with *A Pattern Language*
Gang of Four	Four authors who adapted architectural patterns to software in their 1994 book, cataloging 23 patterns
Three Categories	Creational (object creation), Structural (object composition), Behavioral (object interaction)
Observer	Subscribe to state changes without tight coupling
MVC	Separate Model, View, and Controller for maintainability
Singleton	Ensure only one instance of a class exists
Strategy	Encapsulate interchangeable algorithms for runtime selection

---

Further Reading

External Resources

- Source Making: Design Patterns - Accessible explanations of all 23 GoF patterns with examples
- Refactoring Guru: Design Patterns - Visual guides to design patterns in multiple languages
- Oracle Java Tutorials: Design Patterns - Pattern usage in the Java standard library

---

References

Primary Texts:

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley. -- The original Gang of Four catalog of 23 design patterns.
- Freeman, E., & Robson, E. (2020). *Head First Design Patterns* (2nd ed.). O'Reilly Media. -- Accessible, example-driven coverage of design patterns.
- Bloch, J. (2018). *Effective Java* (3rd ed.). Addison-Wesley. -- Best practices for Java API design, including pattern-related items.

Additional Resources:

- Alexander, C. (1977). *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press. -- The architectural origin of the pattern concept.
- Kerievsky, J. (2004). *Refactoring to Patterns*. Addison-Wesley. -- When and how to introduce patterns into existing code.

Language Documentation:

- Oracle JDK 25 Documentation -- Official Java SE reference

This guide is part of TCSS 305 Programming Practicum, School of Engineering and Technology, University of Washington Tacoma.