1

Introduction to Design Patterns

Objectives

After completing this lesson, you should be able to do the following:

- Introduction to Design Patterns
- History of Design Patterns
- Importance of Design Patterns
- Criticism of Design Patterns
- Classification of Design Patterns

Course Roadmap

Java Design Patterns

- **Lesson 1: Introduction to Design Patterns** — You are here!
- Lesson02: Creational Design Patterns
- Lesson03: Structural Design Patterns
- Lesson04: Behavioral Design Patterns
- Lesson 5: Most Useful Design Patterns

Introduction to Design Patterns

What's a design pattern?

- ***Design patterns*** *are typical solutions to commonly occurring problems in software design. They are like pre-made blueprints that you can customize to solve a recurring design problem in your code.*

- In Simple terms A **design pattern** is proved solution for solving the specific problem/task. [Set of Guide Lines]

- You can’t just find a pattern and copy it into your program, the way you can with off-the-shelf functions or libraries. The pattern is not a specific piece of code, but a general concept for solving a particular problem. You can follow the pattern details and implement a solution that suits the realities of your own program.

- We need to keep in mind that design patterns are programming language independent for solving the common object-oriented design problems. In Other Words, a design pattern represents an idea, not a particular implementation. Using design patterns you can make your code more flexible, reusable, and maintainable.

- **Patterns** are often confused with algorithms, because both concepts describe typical solutions to some known problems.
- While an **algorithm** always defines a clear set of actions that can achieve some goal, a pattern is a more high-level description of a solution. The code of the same pattern applied to two different programs may be different.
  - An analogy to an algorithm is a **cooking recipe**: both have clear steps to achieve a goal.
  - On the other hand, a pattern is more like a **blueprint**: you can see what the result and its features are, but the exact order of implementation is up to you

What does the pattern consist of?

- Most patterns are described very formally so people can reproduce them in many contexts.

1. **Intent** of the pattern briefly describes both the problem and the solution.
2. **Motivation** further explains the problem and the solution the pattern makes possible.
3. **Structure** of classes shows each part of the pattern and how they are related.

Advantage of design pattern:

1. They are reusable in multiple projects.
2. They provide the solutions that help to define the system architecture.
3. They capture the software engineering experiences.
4. They provide transparency to the design of an application.
5. They are well-proved and testified solutions since they have been built upon the knowledge and experience of expert software developers.
6. Design patterns don’t guarantee an absolute solution to a problem. They provide clarity to the system architecture and the possibility of building a better system.

When should we use the design patterns?

- We must use the design patterns **during the analysis and requirement phase of SDLC**(Software Development Life Cycle).
- Design patterns ease the analysis and requirement phase of SDLC by providing information based on prior hands-on experiences.

History of Design Patterns

- Who invented patterns? That's a good, but not a very accurate, question. Design patterns aren't obscure, sophisticated concepts—quite the opposite.
  - Patterns are typical solutions to common problems in object-oriented design. When a solution gets repeated over and over in various projects, someone eventually puts a name to it and describes the solution in detail. That's basically how a pattern gets discovered.

Cont …

- The concept of patterns was first described by Christopher Alexander in **A Pattern Language: Towns, Buildings, Construction**. The book describes a "language" for designing the urban environment. The units of this language are patterns. They may describe how high windows should be, how many levels a building should have, how large green areas in a neighborhood are supposed to be, and so on.

- The idea was picked up by four authors: Erich Gamma, John Vlissides, Ralph Johnson, and Richard Helm. In 1994, they published **Design Patterns: Elements of Reusable Object-Oriented Software**, in which they applied the concept of design patterns to programming.
- The book featured 23 patterns solving various problems of object-oriented design and became a best-seller very quickly. Due to its lengthy name, people started to call it “the book by the gang of four” which was soon shortened to simply “the GoF book”.

Importance Of Design Patterns

- The truth is that you might manage to work as a programmer for many years without knowing about a single pattern.

- A lot of people do just that. Even in that case, though, you might be implementing some patterns without even knowing it. So why would you spend time learning them?

- Design patterns are a toolkit of **tried and tested solutions** to common problems in software design. Even if you never encounter these problems, knowing patterns is still useful because it teaches you how to solve all sorts of problems using principles of object-oriented design.

- Design patterns define a common language that you and your teammates can use to communicate more efficiently. You can say, “Oh, just use a Singleton for that,” and everyone will understand the idea behind your suggestion. No need to explain what a singleton is if you know the pattern and its name.

Criticism Of Design Patterns

- It seems like only lazy people haven’t criticized design patterns yet. Let’s take a look at the most typical arguments against using patterns

Kludges for a weak programming language

- Usually the need for patterns arises when people choose a programming language or a technology that lacks the necessary level of abstraction. In this case, patterns become a kludge that gives the language much-needed super-abilities.
- For example, the **Strategy** pattern can be implemented with a simple anonymous (lambda) function in most modern programming languages.

Inefficient solutions

- Patterns try to systematize approaches that are already widely used. This unification is viewed by many as a dogma, and they implement patterns “to the letter”, without adapting them to the context of their project.

Unjustified use

- This is the problem that haunts many novices who have just familiarized themselves with patterns. Having learned about patterns, they try to apply them everywhere, even in situations where simpler code would do just fine.

> If all you have is a hammer, everything looks like a nail.

Classification Of Design Patterns

Creational Design Patterns	Structural Design Patterns	Behavioural Design Patterns
• Singleton Pattern	• Adapter Pattern	• Template Method Pattern
• Factory Pattern	• Composite Pattern	• Mediator Pattern
• Abstract Factory Pattern	• Proxy Pattern	• Chain of Responsibility Pattern
• Builder Pattern	• Flyweight Pattern	• Observer Pattern
• Prototype Pattern	• Façade Pattern	• Strategy Pattern
	• Bridge Pattern	• Command Pattern
	• Decorator Pattern	• State Pattern
		• Visitor Pattern
		• Interpreter Pattern
		• Iterator Pattern
		• Memento Pattern

Summary

In this lesson, you should have learned how to:

- Introduction to Design Patterns
- History of Design Patterns
- Importance of Design Patterns
- Criticism of Design Patterns
- Classification of Design Patterns