DESIGN PATTERNS

Syromolotov M.D., Klimbasov A.A.

*Belarusian State University of Informatics and Radioelectronics
Minsk, Republic of Belarus*

Maksimchuk R. T. – teacher

In this paper you will understand the importance of the role the design patterns play in programming. First you will be told about what actually design patterns are and will be enlightened on their specifications. Then you can find descriptions of some design patterns: information about their operating principles and scope of application.

In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Uses of Design Patterns

Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.

Often, people only understand how to apply certain software design techniques to certain problems. These techniques are difficult to apply to a broader range of problems. Design patterns provide general solutions, documented in a format that doesn't require specifics tied to a particular problem.

In addition, patterns allow developers to communicate using well-known, well understood names for software interactions. Common design patterns can be improved over time, making them more robust than ad-hoc designs.

There are three basic kinds of design patterns:

Creational:

These design patterns are all about class instantiation. In software engineering, creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design problems or added complexity to the design. Creational design patterns solve this problem by somehow controlling this object creation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.

Examples: Abstract Factory, Builder, Factory Method, Object Pool, Prototype, Singleton.

Structural:

These design patterns are all about Class and Object composition. Structural class-creation patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality.

Examples: Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy.

Behavioral:

These design patterns are all about Class' objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

Examples: Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template method, Visitor.

Generally, the three groups above define how your program elements relate to each other, how they get created, and how they communicate with each other.

Why to use them?

Design patterns are well-thought out solutions to programming problems. Let's draw a parallel with a real-world example.

Suppose you are standing at the base of a mountain and have been given the task of reaching its top by the end of the day. Assuming that you are unaware of the available routes to the mountain top, what will you do? Of course, you will start to climb up as per your own understanding. Your lack of knowledge about the right route makes it extremely difficult for you to decide the best possible route.

If somebody gives you a detailed map along with all possible routes, including the pros and cons of each, then you will be in a much better position to begin your journey and choose the right route.

Conclusion

Design patterns help you solve commonly-occurring problems in software design. But you can’t just find a pattern and copy it into your program, the way you can with off-the-shelf functions or libraries. A pattern is not a specific piece of code, but a general concept for solving a particular problem. They are like pre-made blueprints that you can customize to solve a recurring design problem in your code.

References:

1. Java Design Patterns: A Hands-On Experience with Real-World Examples/ V. Sarcar // Apress,2019
2. Hands-On RESTful API Design Patterns and Best Practices: Design, develop, and deploy highly adaptable, scalable, and secure RESTful web APIs/ Harihara Subramanian[et al.]// Packt Publishing, 2019
3. Head First Design Patterns/Eric Freeman[et al.]// O’Reilly, 2014
4. Hands-On Design Patterns with C# and .NET Core/ Gaurav Aroraa[et al.] // Packt Publishing, 2019
5. Java Program Design: Principles, Polymorphism, and Patterns/Edward Sciore// Apress, 2019
6. Head First Design Patterns/Eric Freeman[et al.]// O’Reilly, 2018
7. Распределенные системы. Паттерны проектирования/ Б. Бёрнс// O’Reilly ,2019