Theory: The concept of patterns

10 minutes 4 / 6 problems solved

Start practicing

§1. Code design

If you are reading this, you must be really interested in programming. It doesn't matter whether you are an experienced developer, just starting your career, or still working on the basics; what really matters is that you are curious, so welcome.

To begin with, let's talk about **code design**. In general, the design of your code is about expressing your ideas clearly to your teammates, colleagues, and clients. We can compare code to text: if you put the lines in the right order and make the structure clear, it will be much easier to explain and understand the text later. From an engineering point of view, your code is **well-designed** if you can agree with the following statements:

1) When you make a small change, it ***does not*** produce a ripple effect elsewhere in the code.
2) Your code is ***easy*** to reuse.
3) It is ***easy*** to maintain your code after release.

§2. Design patterns

In application development, the design of code has to **match the problem** and be **general** enough in order to fit all the requirements that may arise in the future. Everyone tries to find more elegant, suitable and flexible solutions that can also be **reused**. Here is where **design patterns** come into play: these are repeatable solutions to common problems that developers face. Design patterns even have names! For example, if you want to confine yourself to just one instance of a class, ***Singleton*** pattern is going to be the best choice; if you see family relations between objects and you want to encapsulate creational process, you should use ***AbstractFactory***, etc.

As a rule, examples of well-structured object-oriented architecture make use of patterns a lot. When a suitable pattern is used, it tells us that the developer has really paid attention to typical interactions between elements in the system. As a result, the architecture of an application becomes easier to understand.

Being so useful, design patterns have made it onto many programmers' bookshelves: one of the most famous examples is the book *Design Patterns: Elements of Reusable Object-Oriented Software*. You probably heard about its authors, "Gang of Four", which is frequently abbreviated as "GoF". Today it is considered one of the classic books on software design and programming. You may check it out to deepen your understanding or proceed directly to practical learning here.

Note that in this topic we will only consider ***object-oriented*** design patterns.

Current topic:

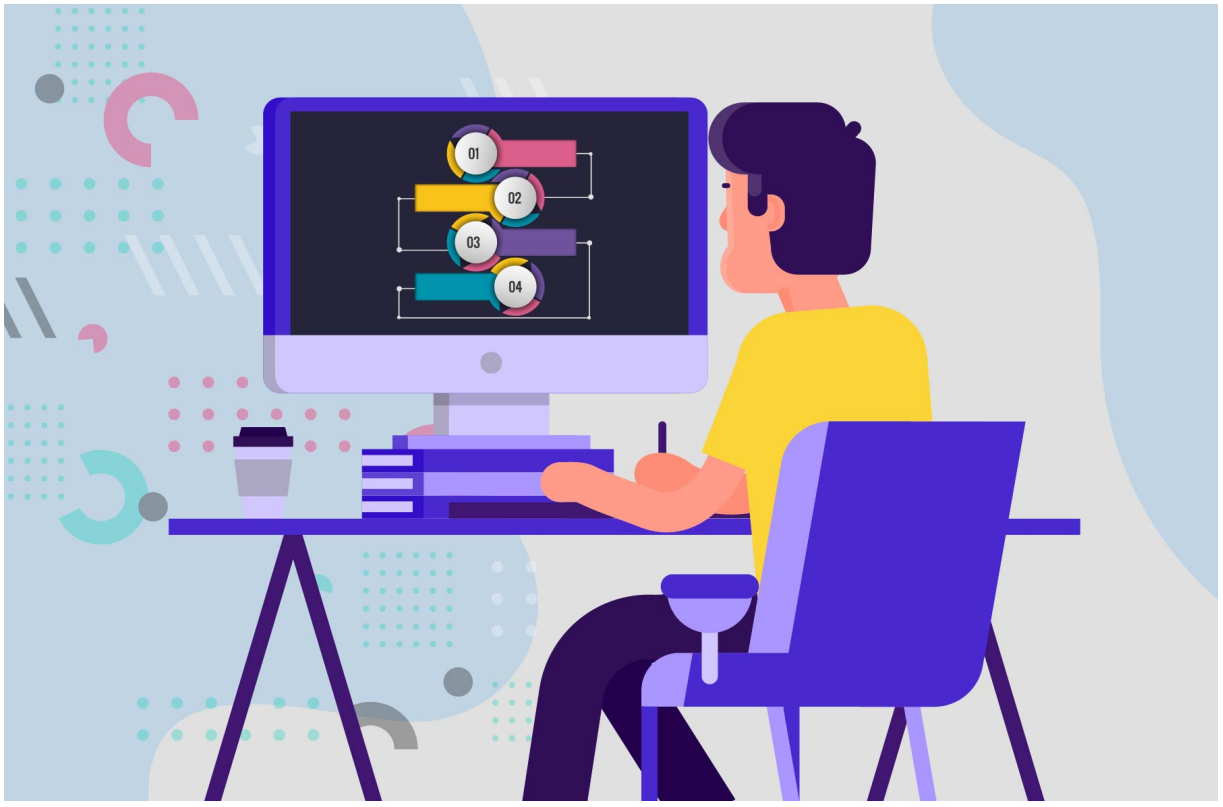
The concept of patterns

Topic depends on:

Stage 3
Introduction to OOP 3

Topic is required for:

Singleton
Strategy
Encapsulating object creation
Command
Builder
Decorator
Template method
Observer
Facade
Function composition
Django MVC



§3. Software design patterns and related concepts

A great thing about patterns is that they help you not to waste your time reinventing the wheel so you can spend it on developing cool features instead. The structure of design patterns is strict: describe the problem, the solution, when to apply the solution, and its consequences. Theoretically, you can combine a few patterns and create your own *monster pattern* that contains, for example, *Builder, Abstract Factory* and *Decorator* simultaneously. However, as you will see from the following topics, it's better to avoid such monsters because patterns have already been well-grouped for you. In other words, don't get too excited, it's really better to use them one at a time.

Using patterns does not require any specific programming language skills or striking imagination. Patterns are also language-independent: even though they can be implemented differently in different languages, the general idea of each pattern is common for all of them. That means that it's possible for you to speak the language of architecture with your colleagues even if they work with different technologies.

§4. Why should I know design patterns?

Here is a list of quite convincing reasons to get familiar with design patterns:

- Patterns provide **tested and commonly used solution templates** for design problems; you don't have to invent anything!
- Patterns improve **flexibility and maintainability** of object-oriented systems, which makes it easier to react to changing requirements.
- Patterns can **speed up the development process**.
- Patterns are a **universal vocabulary** that allows developers to describe a program design using a set of well-known identifiable concepts.
- Patterns are **often used in standard libraries and frameworks**.
- You can find patterns in the source codes of many applications and **quickly understand how they work**, instead of reading thousands of lines of code.

§5. Caveats

In order to achieve flexibility, design patterns usually introduce additional levels of indirectness, which in some cases may complicate the resulting designs and hurt application performance. In other words, even though patterns are supposed to make things easier for you, they may also become an unnecessary complication if applied unwisely. Beginner developers may try to apply patterns everywhere, even in situations where a simpler code would do just fine. Look how design patterns can complicate even the simplest "Hello, World" program.

Table of contents:

To avoid misusing the patterns, you should apply them wisely and be able to correctly adapt them to your problem and language.

§6. Final remarks

When you master the principles of working with patterns so that after successfully applying them you scream "Eureka!" without any doubt in your thoughts, your perception of object-oriented programming will probably change once and for all. In the following topics, you will learn about **Creational**, **Structural** and **Behavioral** design patterns. Be concentrated and attentive: these matters are quite advanced. Happy coding!

Report a typo

719 users liked this theory. **12** didn't like it. **What about you?**

Start practicing

Comments (11) Hints (0) Useful links (1) Show discussion