Design Patterns C

Design Patterns

Design Patterns: Elements of Reusable Object-Oriented Software (1994) is a software engineering book describing software design patterns. The book was

Design Patterns: Elements of Reusable Object-Oriented Software (1994) is a software engineering book describing software design patterns. The book was written by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, with a foreword by Grady Booch. The book is divided into two parts, with the first two chapters exploring the capabilities and pitfalls of object-oriented programming, and the remaining chapters describing 23 classic software design patterns. The book includes examples in C++ and Smalltalk.

It has been influential to the field of software engineering and is regarded as an important source for object-oriented design theory and practice. More than 500,000 copies have been sold in English and in 13 other languages. The authors are often referred to as the Gang of Four (GoF).

Software design pattern

Reusing design patterns can help to prevent such issues, and enhance code readability for those familiar with the patterns. Software design techniques

In software engineering, a software design pattern or design pattern is a general, reusable solution to a commonly occurring problem in many contexts in software design. A design pattern is not a rigid structure to be transplanted directly into source code. Rather, it is a description or a template for solving a particular type of problem that can be deployed in many different situations. Design patterns can be viewed as formalized best practices that the programmer may use to solve common problems when designing a software application or system.

Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying the final application classes or objects that are involved. Patterns that imply mutable state may be unsuited for functional programming languages. Some patterns can be rendered unnecessary in languages that have built-in support for solving the problem they are trying to solve, and object-oriented patterns are not necessarily suitable for non-object-oriented languages.

Design patterns may be viewed as a structured approach to computer programming intermediate between the levels of a programming paradigm and a concrete algorithm.

Design pattern

interaction design / human–computer interaction Pedagogical patterns, in teaching Pattern gardening, in gardening Business models also have design patterns. See

A design pattern is the re-usable form of a solution to a design problem. The idea was introduced by the architect Christopher Alexander and has been adapted for various other disciplines, particularly software engineering.

Pattern

considered a pattern. Mathematics can be taught as a collection of patterns. Gravity is a source of ubiquitous scientific patterns or patterns of observation

A pattern is a regularity in the world, in human-made design, or in abstract ideas. As such, the elements of a pattern repeat in a predictable manner. A geometric pattern is a kind of pattern formed of geometric shapes and typically repeated like a wallpaper design.

Any of the senses may directly observe patterns. Conversely, abstract patterns in science, mathematics, or language may be observable only by analysis. Direct observation in practice means seeing visual patterns, which are widespread in nature and in art. Visual patterns in nature are often chaotic, rarely exactly repeating, and often involve fractals. Natural patterns include spirals, meanders, waves, foams, tilings, cracks, and those created by symmetries of rotation and reflection. Patterns have an underlying mathematical structure; indeed, mathematics can be seen as the search for regularities, and the output of any function is a mathematical pattern. Similarly in the sciences, theories explain and predict regularities in the world.

In many areas of the decorative arts, from ceramics and textiles to wallpaper, "pattern" is used for an ornamental design that is manufactured, perhaps for many different shapes of object. In art and architecture, decorations or visual motifs may be combined and repeated to form patterns designed to have a chosen effect on the viewer.

Visitor pattern

Visitor design pattern is one of the twenty-three well-known Gang of Four design patterns that describe how to solve recurring design problems to design flexible

A visitor pattern is a software design pattern that separates the algorithm from the object structure. Because of this separation, new operations can be added to existing object structures without modifying the structures. It is one way to follow the open/closed principle in object-oriented programming and software engineering.

In essence, the visitor allows adding new virtual functions to a family of classes, without modifying the classes. Instead, a visitor class is created that implements all of the appropriate specializations of the virtual function. The visitor takes the instance reference as input, and implements the goal through double dispatch.

Programming languages with sum types and pattern matching obviate many of the benefits of the visitor pattern, as the visitor class is able to both easily branch on the type of the object and generate a compiler error if a new object type is defined which the visitor does not yet handle.

Builder pattern

23 classic design patterns described in the book Design Patterns and is sub-categorized as a creational pattern. The builder design pattern solves problems

The builder pattern is a design pattern that provides a flexible solution to various object creation problems in object-oriented programming. The builder pattern separates the construction of a complex object from its representation. It is one of the 23 classic design patterns described in the book Design Patterns and is sub-categorized as a creational pattern.

Factory method pattern

overridden by subclasses. It is one of the 23 classic design patterns described in the book Design Patterns (often referred to as the "Gang of Four" or simply

In object-oriented programming, the factory method pattern is a design pattern that uses factory methods to deal with the problem of creating objects without having to specify their exact classes. Rather than by calling a constructor, this is accomplished by invoking a factory method to create an object. Factory methods can be specified in an interface and implemented by subclasses or implemented in a base class and optionally overridden by subclasses. It is one of the 23 classic design patterns described in the book Design Patterns

(often referred to as the "Gang of Four" or simply "GoF") and is subcategorized as a creational pattern.

Anti-pattern

organizational, and cultural anti-patterns. According to the authors of Design Patterns, there are two key elements to an anti-pattern that distinguish it from

An anti-pattern in software engineering, project management, and business processes is a common response to a recurring problem that is usually ineffective and risks being highly counterproductive. The term, coined in 1995 by computer programmer Andrew Koenig, was inspired by the book Design Patterns (which highlights a number of design patterns in software development that its authors considered to be highly reliable and effective) and first published in his article in the Journal of Object-Oriented Programming.

A further paper in 1996 presented by Michael Ackroyd at the Object World West Conference also documented anti-patterns.

It was, however, the 1998 book AntiPatterns that both popularized the idea and extended its scope beyond the field of software design to include software architecture and project management.

Other authors have extended it further since to encompass environmental, organizational, and cultural anti-patterns.

Singleton pattern

singleton pattern can also be used as a basis for other design patterns, such as the abstract factory, factory method, builder and prototype patterns. Facade

In object-oriented programming, the singleton pattern is a software design pattern that restricts the instantiation of a class to a singular instance. It is one of the well-known "Gang of Four" design patterns, which describe how to solve recurring problems in object-oriented software. The pattern is useful when exactly one object is needed to coordinate actions across a system.

More specifically, the singleton pattern allows classes to:

Ensure they only have one instance

Provide easy access to that instance

Control their instantiation (for example, hiding the constructors of a class)

The term comes from the mathematical concept of a singleton.

Structural pattern

In software engineering, structural design patterns are design patterns that ease the design by identifying a simple way to realize relationships among

In software engineering, structural design patterns are design patterns that ease the design by identifying a simple way to realize relationships among entities.

Examples of Structural Patterns include:

Adapter pattern: 'adapts' one interface for a class into one that a client expects

Adapter pipeline: Use multiple adapters for debugging purposes.

Retrofit Interface Pattern: An adapter used as a new interface for multiple classes at the same time.

Aggregate pattern: a version of the Composite pattern with methods for aggregation of children

Bridge pattern: decouple an abstraction from its implementation so that the two can vary independently

Tombstone: An intermediate "lookup" object contains the real location of an object.

Composite pattern: a tree structure of objects where every object has the same interface

Decorator pattern: add additional functionality to an object at runtime where subclassing would result in an exponential rise of new classes

Extensibility pattern: a.k.a. Framework - hide complex code behind a simple interface

Facade pattern: create a simplified interface of an existing interface to ease usage for common tasks

Flyweight pattern: a large quantity of objects share a common properties object to save space

Marker pattern: an empty interface to associate metadata with a class.

Pipes and filters: a chain of processes where the output of each process is the input of the next

Opaque pointer: a pointer to an undeclared or private type, to hide implementation details

Proxy pattern: a class functioning as an interface to another thing