Understanding Design Patterns is a tricky subject. Should I learn all of the design patterns? When should I use a specific design pattern?

Are design patterns important?

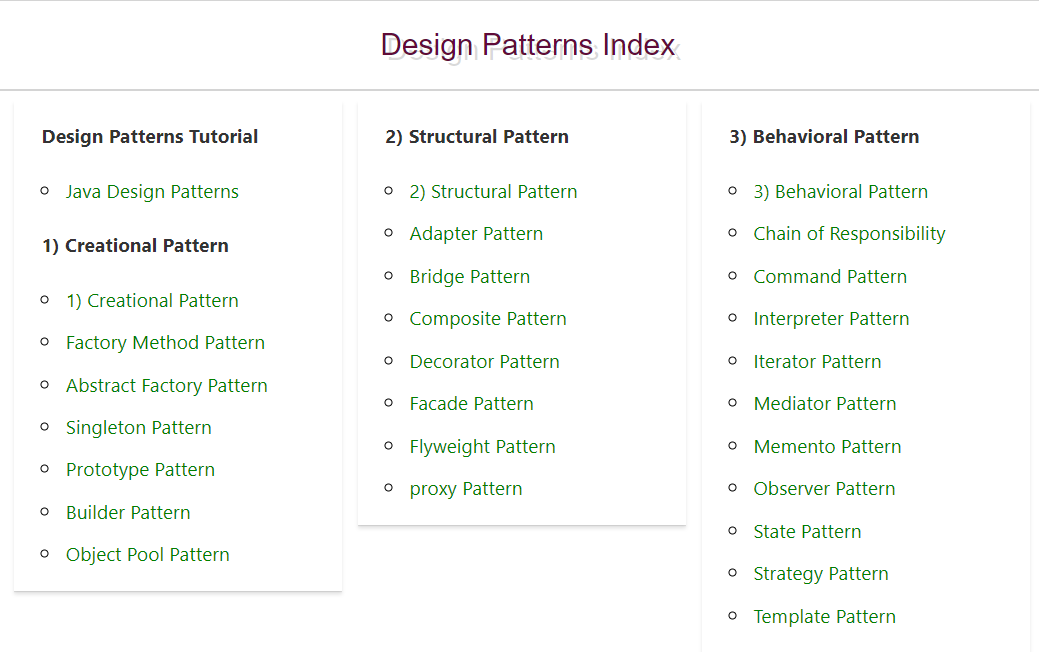
What about the Gang of Four patterns?

Should I learn them all? These are just some of the questions that I get surrounding design patterns. In this video, I am going to answer your design patterns questions.

Design patterns you can make your code more flexible, reusable and maintainable. It is the most important part because java internally follows design patterns.

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.



<https://www.javatpoint.com/factory-method-design-pattern>

**Christopher Alexander** was the first person who invented all the above Design Patterns in 1977.

But later the **Gang of Four - Design patterns, elements of reusable object-oriented software** book was written by a group of four persons named as Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides in 1995.

That's why all the above 23 Design Patterns are known as **Gang of Four (GoF) Design Patterns.**

GoF Design Patterns are broken into three categories: Creational Patterns for the creation of objects; Structural Patterns to provide relationship between objects; and finally, Behavioral Patterns to help define how objects interact.

## Gang of Four Design Patterns

### **Creational Design Patterns**

* [Abstract Factory](https://springframework.guru/gang-of-four-design-patterns/abstract-factory-design-pattern/). Allows the creation of objects without specifying their concrete type.
* [Builder](https://springframework.guru/gang-of-four-design-patterns/builder-pattern/). Uses to create complex objects.
* [Factory Method](https://springframework.guru/gang-of-four-design-patterns/factory-method-design-pattern/). Creates objects without specifying the exact class to create.
* [Prototype](https://springframework.guru/gang-of-four-design-patterns/prototype-pattern/). Creates a new object from an existing object.
* [Singleton](https://springframework.guru/gang-of-four-design-patterns/singleton-design-pattern/). Ensures only one instance of an object is created.

### **Structural Design Patterns**

* [Adapter](https://springframework.guru/gang-of-four-design-patterns/adapter-pattern/). Allows for two incompatible classes to work together by wrapping an interface around one of the existing classes.
* [Bridge](https://springframework.guru/gang-of-four-design-patterns/bridge-pattern/). Decouples an abstraction so two classes can vary independently.
* [Composite](https://springframework.guru/gang-of-four-design-patterns/composite-pattern/). Takes a group of objects into a single object.
* [Decorator](https://springframework.guru/gang-of-four-design-patterns/decorator-pattern/). Allows for an object’s behavior to be extended dynamically at run time.
* [Facade](https://springframework.guru/gang-of-four-design-patterns/facade-pattern/). Provides a simple interface to a more complex underlying object.
* [Flyweight](https://springframework.guru/gang-of-four-design-patterns/flyweight-pattern/). Reduces the cost of complex object models.
* [Proxy](https://springframework.guru/gang-of-four-design-patterns/proxy-pattern/). Provides a placeholder interface to an underlying object to control access, reduce cost, or reduce complexity.

### **Behavior Design Patterns**

* [Chain of Responsibility](https://springframework.guru/gang-of-four-design-patterns/chain-of-responsibility-pattern/). Delegates commands to a chain of processing objects.
* [Command](https://springframework.guru/gang-of-four-design-patterns/command-pattern/). Creates objects which encapsulate actions and parameters.
* [Interpreter](https://springframework.guru/gang-of-four-design-patterns/interpreter-pattern/). Implements a specialized language.
* [Iterator](https://springframework.guru/gang-of-four-design-patterns/iterator-pattern/). Accesses the elements of an object sequentially without exposing its underlying representation.
* [Mediator](https://springframework.guru/gang-of-four-design-patterns/mediator-pattern/). Allows loose coupling between classes by being the only class that has detailed knowledge of their methods.
* [Memento](https://springframework.guru/gang-of-four-design-patterns/memento-pattern/). Provides the ability to restore an object to its previous state.
* [Observer](https://springframework.guru/gang-of-four-design-patterns/observer-pattern/). Is a publish/subscribe pattern which allows a number of observer objects to see an event.
* [State](https://springframework.guru/gang-of-four-design-patterns/state-pattern/). Allows an object to alter its behavior when its internal state changes.
* [Strategy](https://springframework.guru/gang-of-four-design-patterns/strategy-pattern/). Allows one of a family of algorithms to be selected on-the-fly at run-time.
* [Template Method](https://springframework.guru/gang-of-four-design-patterns/template-method-pattern/). Defines the skeleton of an algorithm as an abstract class, allowing its sub-classes to provide concrete behavior.
* [Visitor](https://springframework.guru/gang-of-four-design-patterns/visitor-pattern/). Separates an algorithm from an object structure by moving the hierarchy of methods into one object.

**Problem**

**Given:**  
Suppose you want to create a class for which only a single instance (or object) should be created and that single object can be used by all other classes.

**Solution:**  
**Singleton design pattern** is the best solution of above specific problem. So, every design pattern has **some specification or set of rules** for solving the problems.

1. Singleton (DB driver)
2. Facade (building front view)
3. Bridge/Connector (APIs)
4. Strategy ( notification to sales clients)
5. 5. Pub-Sub/Observer (event based)