**GoF Design Pattern Types**

<https://www.journaldev.com/31902/gangs-of-four-gof-design-patterns?utm_source=browser&utm_medium=rss_notification&utm_id=287396.27052469685>

GoF Design Patterns are divided into three categories:

1. **Creational**: The design patterns that deal with the creation of an object.
2. **Structural**: The design patterns in this category deals with the class structure such as Inheritance and Composition.
3. **Behavioral**: This type of design patterns provide solution for the better interaction between objects, how to provide lose coupling, and flexibility to extend easily in future.

**Creational Design Patterns**

There are 5 design patterns in the creational design patterns category.

|  |  |
| --- | --- |
| PATTERN NAME | DESCRIPTION |
| [Singleton](https://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples) | The [singleton](https://www.journaldev.com/1377/java-singleton-design-pattern-best-practices-examples) pattern restricts the initialization of a class to ensure that only one instance of the class can be created. |
| [Factory](https://www.journaldev.com/1392/factory-design-pattern-in-java) | The factory pattern takes out the responsibility of instantiating an object from the class to a Factory class. |
| [Abstract Factory](https://www.journaldev.com/1418/abstract-factory-design-pattern-in-java) | Allows us to create a Factory for factory classes. |
| [Builder](https://www.journaldev.com/1425/builder-design-pattern-in-java) | Creating an object step by step and a method to finally get the object instance. |
| [Prototype](https://www.journaldev.com/1440/prototype-design-pattern-in-java) | Creating a new object instance from another similar instance and then modify according to our requirements. |

**Structural Design Patterns**

There are 7 structural design patterns defined in the Gangs of Four design patterns book.

|  |  |
| --- | --- |
| PATTERN NAME | DESCRIPTION |
| [Adapter](https://www.journaldev.com/1487/adapter-design-pattern-java) | Provides an interface between two unrelated entities so that they can work together. |
| [Composite](https://www.journaldev.com/1535/composite-design-pattern-in-java) | Used when we have to implement a part-whole hierarchy. For example, a diagram made of other pieces such as circle, square, triangle, etc. |
| [Proxy](https://www.journaldev.com/1572/proxy-design-pattern) | Provide a surrogate or placeholder for another object to control access to it. |
| [Flyweight](https://www.journaldev.com/1562/flyweight-design-pattern-java) | Caching and reusing object instances, used with [immutable](https://www.journaldev.com/129/how-to-create-immutable-class-in-java) objects. For example, string pool. |
| [Facade](https://www.journaldev.com/1557/facade-design-pattern-in-java) | Creating a wrapper interfaces on top of existing interfaces to help client applications. |
| [Bridge](https://www.journaldev.com/1491/bridge-design-pattern-java) | The bridge [design pattern](https://www.journaldev.com/1827/java-design-patterns-example-tutorial) is used to decouple the interfaces from implementation and hiding the implementation details from the client program. |
| [Decorator](https://www.journaldev.com/1540/decorator-design-pattern-in-java-example) | The decorator design pattern is used to modify the functionality of an object at runtime. |

**Behavioral Design Patterns**

There are 11 behavioral design patterns defined in the GoF design patterns.

|  |  |
| --- | --- |
| PATTERN NAME | DESCRIPTION |
| [Template Method](https://www.journaldev.com/1763/template-method-design-pattern-in-java) | used to create a template method stub and defer some of the steps of implementation to the subclasses. |
| [Mediator](https://www.journaldev.com/1730/mediator-design-pattern-java) | used to provide a centralized communication medium between different objects in a system. |
| [Chain of Responsibility](https://www.journaldev.com/1617/chain-of-responsibility-design-pattern-in-java) | used to achieve loose coupling in software design where a request from the client is passed to a chain of objects to process them. |
| [Observer](https://www.journaldev.com/1739/observer-design-pattern-in-java) | useful when you are interested in the state of an object and want to get notified whenever there is any change. |
| [Strategy](https://www.journaldev.com/1754/strategy-design-pattern-in-java-example-tutorial) | Strategy pattern is used when we have multiple algorithm for a specific task and client decides the actual implementation to be used at runtime. |
| [Command](https://www.journaldev.com/1624/command-design-pattern) | Command Pattern is used to implement lose coupling in a request-response model. |
| [State](https://www.journaldev.com/1751/state-design-pattern-java) | State design pattern is used when an Object change it’s behavior based on it’s internal state. |
| [Visitor](https://www.journaldev.com/1769/visitor-design-pattern-java) | Visitor pattern is used when we have to perform an operation on a group of similar kind of Objects. |
| [Interpreter](https://www.journaldev.com/1635/interpreter-design-pattern-java) | defines a grammatical representation for a language and provides an interpreter to deal with this grammar. |
| [Iterator](https://www.journaldev.com/1716/iterator-design-pattern-java) | used to provide a standard way to traverse through a group of Objects. |
| [Memento](https://www.journaldev.com/1734/memento-design-pattern-java) | The memento design pattern is used when we want to save the state of an object so that we can restore later on. |