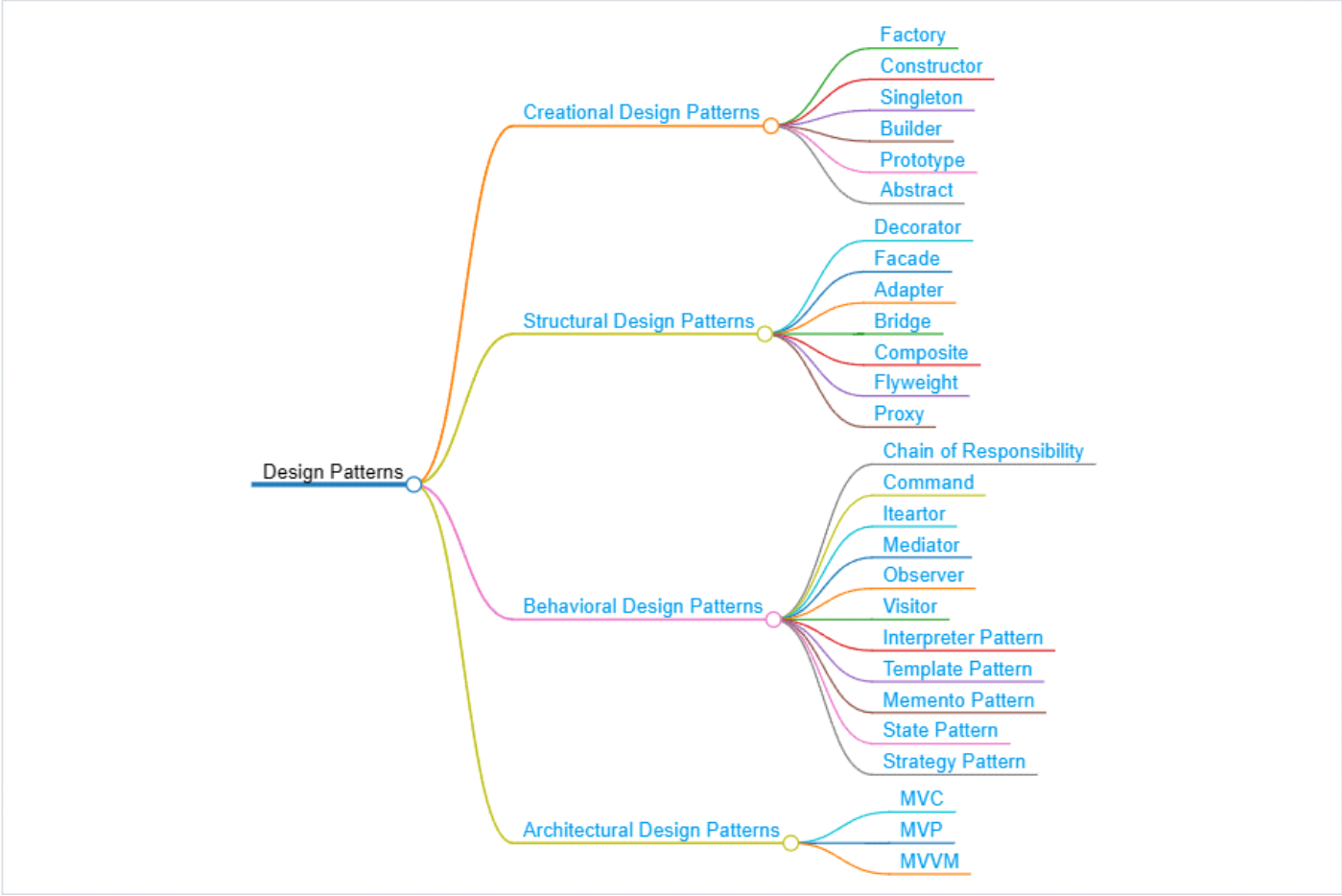
Design Patterns

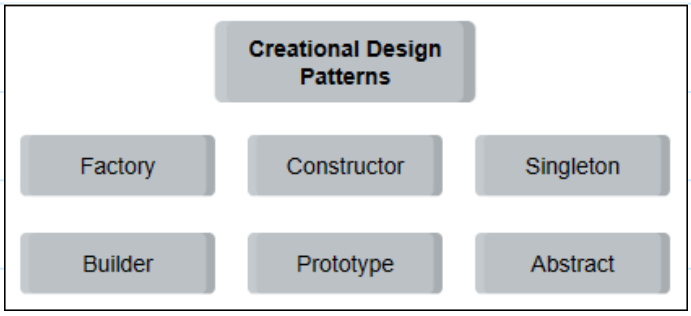
Friday, January 19, 2024 6:10 PM



- Creational design patterns: provide a mechanism for creating objects in specific situations without revealing the creation method.
- Structural design patterns: concerned with class/object composition and relationships between objects
- Behavior design patterns: communication between dissimilar objects in the system
- Architectural design patterns: solving architectural problems within a given context in software architecture.

Creational Design Patterns

Friday, January 19, 2024 6:10 PM

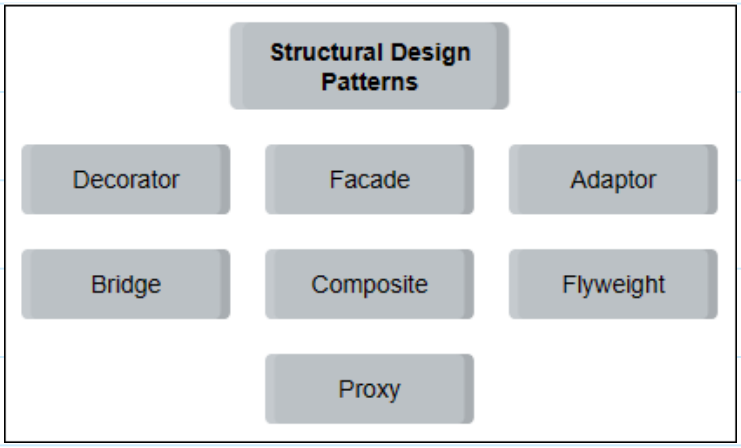


- Factory pattern: provides a template that can be used to create objects; does not involve using a constructor.
- Constructor pattern: a class-based pattern that relies on constructor to create objects
- Singleton pattern: restricts the initiation of a class to a single object. The next time to create the same object will return the first-time created one
- Builder pattern: create objects using similar objects
- Prototype pattern: create objects with default values using an existing objects
- Abstract pattern: create multiple objects from the same family

Factory pattern	• When the type of objects required cannot be anticipated beforehand. • When multiple objects that share similar characteristics need to be created. • When you want to generalize the object instantiation process, since the object set up is complex in nature.
Constructor pattern	• You can use it when you want to create multiple instances of the same template, since the instances can share methods but can still be different. • It can be useful in the Libraries and Plugins design.
Singleton pattern	• The Singleton pattern is mostly used in cases where we want a single object to coordinate actions across a system. • **Services** can be singleton since they store the state, configuration, and provide access to resources. Therefore, it makes sense to have a single instance of a service in an application. • **Databases** such as MongoDB utilize the Singleton pattern when it comes to database connections. • **Configurations** are used if there is an object with a specific configuration, and we don't need a new instance every time that configuration object is needed.
Builder pattern	• You can use this design pattern when building apps that require you to create complex objects. It can help you hide the construction process of building these objects. • A good example would be a DOM, where we might need to create plenty of nodes and attributes. The construction process can get quite messy if we are building a complex DOM object. In cases like these, the Builder pattern can be used.
Prototype pattern	• To eliminate the overhead of initializing an object. • When we want the system to be independent about how the products in it are created. • When creating objects from a database whose values are copied to the cloned object.
Abstract pattern	• Applications requiring the reuse or sharing of objects. • Applications with complex logic because they have multiple families of related objects that need to be used together. • When we require object caching. • When the object creation process is to be shielded from the client.

Structural Design Patterns

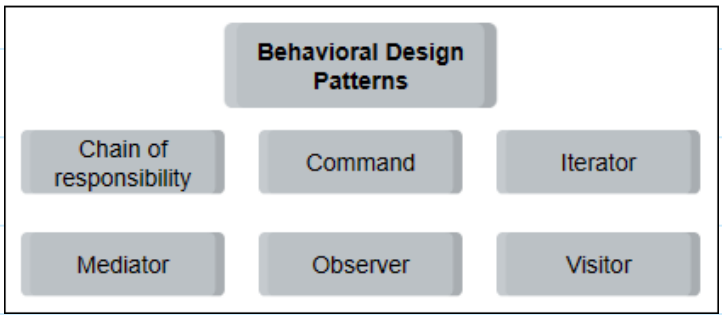
Friday, January 19, 2024 6:10 PM



Structural Design Patterns	When to Use
Decorator	• To modify or extend the functionality of an object without changing its base code. • To implement additional functionalities of similar objects instead of reusing the same code.
Facade	• To simplify a client's interaction with a system by hiding the underlying complex code. • To interact with the methods present in a library without knowing the processing that happens in the background.
Adapter	• To enable old APIs to work with new refactored ones. • To allow an object to cooperate with a class that has an incompatible interface. • To reuse the existing functionality of classes.
Bridge	• To extend a class in several independent dimensions. • To change the implementation at run time. • To share the implementation between objects.
Composite	• To allow the reuse of objects without worrying about their compatibility. • To develop a scalable application that uses plenty of objects. • To create a tree-like hierarchy of objects.
Flyweight	• To share a list of immutable strings across the application. • To prevent load time as it allows caching.
Proxy	• To reduce the workload on the target object.

Behavior Design Pattern

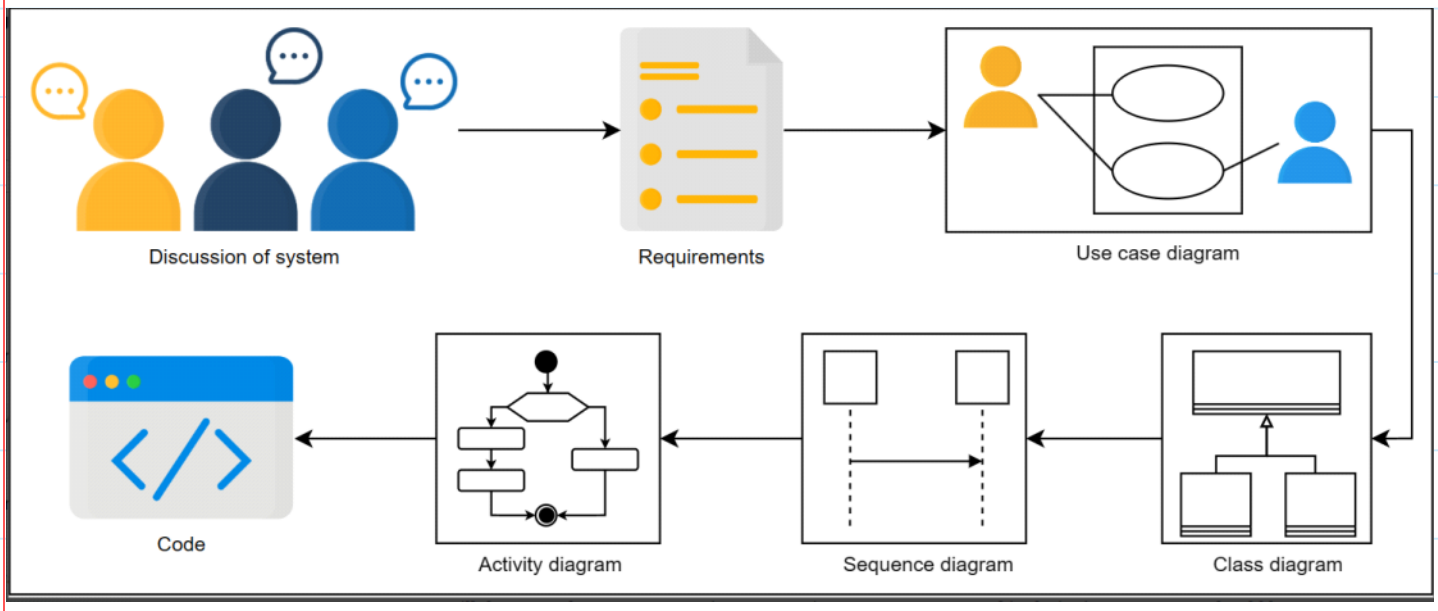
Friday, January 19, 2024 6:10 PM



Behavioral Design Patterns	When to use
Chain of Responsibility pattern	• It can be used where a program is written to handle various requests in different ways without knowing the sequence and type of requests beforehand. • It can be used in the process of *event bubbling* in the DOM, where the event propagates through the nested elements, one of which may choose to handle the event.
Command pattern	• It can be used to queue and execute requests at different times. • It can be used to perform operations such as “reset” or “undo”. • It can be used to keep a history of requests made.
Iterator pattern	• This pattern can be used when dealing with problems explicitly related to iteration, for designing flexible looping constructs and accessing elements from a complex collection without knowing the underlying representation. • It can be used to implement a generic iterator that traverses any collection independent of its type efficiently.
Mediator pattern	• It can be used to avoid the tight coupling of objects in a system with a lot of objects. • It can be used to improve code readability. • It can be used to make code easier to maintain.
Observer pattern	• It can be used to improve code management by breaking down large applications into a system of loosely-coupled objects. • It can be used to improve communication between different parts of the application. • It can be used to create a one-to-many dependency between objects that are loosely coupled.
Visitor pattern	• It can be used to perform similar operations on different objects of a data structure. • It can be used to perform specific operations on different objects in the data structure. • It can be used to add extensibility to libraries or frameworks.

Real-Word Design Process

Friday, January 19, 2024 6:10 PM



Top-down	Bottom-up
This approach constructs high-level objects, and then designs the smaller subcomponents.	This approach identifies the smallest components and uses those components as a base to design bigger components.
It's a backward-looking approach.	It's a forward-looking approach.
It's mainly used in structural programming.	It's mainly used in object-oriented design.
It allows for a high amount of data redundancy.	It allows for a minimum data redundancy.