**Design Patterns** In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

**Uses of Design Patterns**  Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.In addition, patterns allow developers to communicate using well-known, well understood names for software interactions. Common design patterns can be improved over time, making them more robust than ad-hoc designs.

**Creational design patterns** This design patterns is all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.

* Abstract Factory - creates an instance of several families of classes
* Builder - separates object construction from its representation
* Factory Method - creates an instance of several derived classes
* Object Pool - avoid expensive acquisition and release of resources by recycling objects that are no longer in use
* Prototypy - a fully initialized instance to be copied or cloned
* Singleton - a class of which only a single instance can exist
* Structural design patterns

This design patterns is all about Class and Object composition. **Structural class-creation** patterns use inheritance to compose interfaces. Structural object-patterns define ways to compose objects to obtain new functionality.

* Adapter - match interfaces of different classes
* Bridge - separates an object’s interface from its implementation
* Composite - a tree structure of simple and composite objects
* Decorator - add responsibilities to objects dynamically
* Façade - a single class that represents an entire subsystem
* Flyweight - a fine-grained instance used for efficient sharing
* Private Class Date - restricts accessor/mutator access
* Proxy - an object representing another object

**Behavioral design patterns** - this design patterns is all about Class's objects communication. Behavioral patterns are those patterns that are most specifically concerned with communication between objects.

* Chain of responsibility - a way of passing a request between a chain of objects
* Command - encapsulate a command request as an object
* Interpreter - a way to include language elements in a program
* Iterator - sequentially access the elements of a collection
* Mediator - defines simplified communication between classes
* Memento - capture and restore an object's internal state
* Null Object - designed to act as a default value of an object
* Observer - a way of notifying change to a number of classes
* State - alter an object's behavior when its state changes
* Strategy - encapsulates an algorithm inside a class
* Template method - defer the exact steps of an algorithm to a subclass
* Visitor - defines a new operation to a class without change