**Composite:**

The Composite design pattern allows you to compose objects into tree structures and treat individual objects and compositions of objects uniformly.

**Flyweight:**

The Flyweight design pattern is used to minimize the memory usage and the number of objects created by an application by sharing common data among objects.

**Factory Method:**

The Factory Method design pattern defines an interface for creating an object, but lets subclasses decide which class to instantiate. It is used to create objects in a super class, but allow subclasses to alter the type of objects that will be created.

**Decorator:**

The Decorator design pattern is used to add new behavior to existing objects dynamically, without modifying their source code. It is an alternative to subclassing, which allows you to extend the functionality of an object at runtime.

**Singleton:**

The Singleton design pattern ensures that a class has only one instance, and provides a global point of access to it.

**Facade:**

The Facade design pattern provides a unified interface to a set of interfaces in a subsystem, hiding the complexity of the subsystem from the client.

**Adapter:**

The Adapter design pattern converts the interface of a class into another interface that the client expects. It is used to allow two incompatible classes to work together by wrapping an instance of one class with an adapter that conforms to the interface of the other class.

**Proxy:**

The Proxy design pattern provides a surrogate or placeholder for another object, which controls access to that object. It is used to provide a level of indirection, which can be used to perform tasks such as lazy loading, access control, or caching.

**Strategy:**

The Strategy design pattern allows you to define a family of algorithms, and make them interchangeable, so that the client can choose the algorithm to use at runtime.

**State:**

The State design pattern allows an object to alter its behavior when its internal state changes. It appears as if the object has changed its class.

**Observer:**

The Observer design pattern defines a one-to-many dependency between objects, so that when one object changes state, all its dependents are notified and updated automatically.

**Memento:**

The Memento design pattern allows you to capture the internal state of an object and restore it later, without violating the object's encapsulation.

**Iterator:**

The Iterator design pattern provides a way to access the elements of an aggregate object sequentially, without exposing its underlying representation.

**Template Method:**

The Template Method design pattern defines the skeleton of an algorithm in a method, deferring some steps to subclasses. It allows you to define the steps of an algorithm, and allow subclasses to provide the implementation for one or more of those steps.