Паттерны:

Можно выделить три типа шаблонов:

Порождающие, Структурные и Поведенческие.

Перечислим основные паттерны проектирования для каждого из них.

# **Порождающие (Creational)** Данные паттерны выделяют процесс инстанцирования. С помощью них система перестает зависеть от метода формирования композиции и представления объектов.

## **Прототип (Prototype)**. Такой шаблон задаёт типы создаваемых объектов посредством экземпляра-прототипа. Более того, копируя прототип, паттерн формирует новые объекты. При использовании этого шаблона вы сможете отойти от реализации и придерживаться принципа «программирование через интерфейсы». В роли возвращающего типа выступает указанный интерфейс/абстрактный класс на вершине иерархии. При этом классы-наследники могут подставить в это место наследника, выполняющего реализацию данного типа. Иными словами, прототип позволяет создать объект посредством клонирования, а не с помощью конструктора.

* 1. **Factory (Фабрика)**. Это не считается паттерном в строгом смысле слова. Правильнее будет обозначить Factory как подход, в рамках которого логика создания объектов выносится в отдельный класс.
  2. **Фабричный метод (Factory Method)**. Шаблон, который определят общий интерфейс для формирования объектов в суперклассе. Данный метод даёт подклассам возможность изменять вид создаваемых объектов.
  3. **Абстрактная фабрика (Abstract factory)**. Это порождающий паттерн проектирования, который предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, без непосредственной спецификации их конкретных классов. Реализация такого шаблона обеспечивается за счёт создания абстрактного класса Factory. Этот класс выступает в качестве интерфейса для создания компонентов системы (скажем, применительно к оконному интерфейсу он может формировать окна и кнопки). После всего этого пишутся классы, которые реализуют данный интерфейс.
  4. **Одиночка (Singleton)**. Очередной порождающий паттерн. С помощью него обеспечивается наличие единственного экземпляра класса с глобальной точкой доступа в однопоточном приложении.
  5. **Строитель (Builder)**. Полезный порождающий паттерн, который, по сути, является методом создания составного объекта. Он дифференцирует сложный объект на конструирование и представление. Благодаря этому при выполнении одной и той же операции конструирования вы можете получить разные представления.

# **Структурные (Structural)** Такие шаблоны определяют всевозможные структуры высокого уровня сложности, которые вносят изменения в интерфейс уже созданных объектов или его реализацию. Это позволяет упростить процесс разработки и сделать программу более оптимизированной. Шаблоны, входящие в эту разновидность, облегчают проектирование за счёт того, что они выявляют простейший метод реализации отношений между субъектами.

* 1. **Мост (Bridge)**. Применяется в проектировании ПО. Он разделяет абстракцию и реализацию таким образом, чтобы они могли меняться независимо. Данный паттерн работает с помощью инкапсуляции, агрегирования и может применять наследование в целях распределения межклассовой ответственности.
  2. **Декоратор (Decorator)**. Данный шаблон был сформирован для динамического подключения дополнительного поведения к объекту. С его помощью практика создания подклассов получает гибкую альтернативу. Это позволяет сделать функционал более широким.
  3. **Адаптер (Adapter)**. Он необходим для организации применения функций объекта, который нельзя модифицировать, посредством специального интерфейса.
  4. **Компоновщик (Composite pattern)**. Такой шаблон способен объединять объекты в древовидную структуру. Это полезно для представления иерархии от частного к целому. Тем самым Composite pattern даёт клиентам возможность обращаться к отдельным объектам и к группам объектов одинаковым образом.
  5. **Заместитель (Proxy)**. Он предоставляет объект, контролирующий доступ к другому объекту, перехватывая при этом все вызовы. Иными словами, он выступает в качестве контейнера Фасад (Facade). Он помогает скрыть сложность системы. Принцип работы довольно прост: все возможные внешние вызовы сводятся к одному и тому же объекту, который передаёт эти вызовы соответствующим объектам системы.
  6. **Приспособленец (Flyweight, «легковесный (элемент)»)**. В процессе применения данного паттерна объект представляет себя как уникальный экземпляр в разных частях программы, однако, на самом деле это не так.

# **Поведенческие (behavioral)** Такие паттерны определяют алгоритмы и методы реализации взаимодействия каких-либо объектов и классов. Они выявляют общие закономерности связей между объектами, которые реализуют данные шаблоны. Использование поведенческих паттернов позволяет снизить уровень связности системы и упростить взаимодействие между объектами. Это позволяет сделать программный продукт более гибким.

* 1. **Итератор (iterator)**. Это интерфейс, который даёт доступ к элементам коллекции (массива или контейнера), а также навигацию по ним. В зависимости от конкретной системы итераторы носят разные названия. Если говорить о системах управления, то это курсоры.
  2. **Интерпретатор (Interpreter)**. Он решает одну очень известную, но постоянно меняющуюся задачу. Альтернативное название — Little (Small) Language.
  3. **Цепочка обязанностей (Chain of responsibility)**. Такой шаблон нужен для организации уровней ответственности в системе.
  4. **Хранитель (Memento)**. С его помощью можно выполнить закрепление и сохранение внутреннего состояния объекта без нарушения инкапсуляции. Это нужно для того, чтобы в последующем можно было восстановить его в это самое состояние.
  5. **Команда (Command)**. Он представляет действие и применяется в рамках объектно-ориентированного программирования. Объект команды содержит в себе как само действие, так и его параметры.
  6. **Посредник (Mediator)**. Данный паттерн проектирования позволяет обеспечить взаимодействие нескольких объектов. В процессе этого создается слабая связанность и объекты избавляются от потребности в явных ссылках друг на друга.
  7. **Состояние (State)**. Применяется в ситуациях, когда в процессе выполнения программы объект должен изменять свое поведение, в зависимости от того, в каком состоянии он находится.
  8. **Наблюдатель (Observer)**. Его также называют «подчинённые» (Dependents). Формирует особый механизм у класса, который дает возможность экземпляру объекта этого класса получать уведомления от остальных объектов. В оповещения содержится информация об изменении их состояния. Таким образом, происходит наблюдение за объектом.
  9. **Посетитель (visitor)**. Отвечает за описание операции, которая выполняется над объектами остальных классов. Если внести изменения в visitor, то вам не придётся корректировать обслуживаемые классы.
  10. **Стратегия (Strategy)**. Этот шаблон необходим для того, чтобы определять семейства алгоритмов и инкапсуляции каждого из них. Кроме того, данный паттерн позволяет обеспечить их взаимозаменяемость. Благодаря этому появляется возможность подбора алгоритма посредством определения соответствующего класса. С помощью Strategy можно изменять применяемый алгоритм вне зависимости от использующих его объектов-клиентов.
  11. Шаблонный метод (Template method). Определяет «фундамент» алгоритма и даёт наследникам возможность переопределять те или иные шаги алгоритма, оставляя его общую структуру нетронутой.