**Паттерны (GoF) –** эффективные решения типовых проблем проектирования ПО.

*Типы шаблонов:*

Порождающие – отделяют логику создания объектов от остальной логики и предоставляют интерфейс для его создания.

Структурные – определяют отношения между классами и объектами указывая, как из классов и объектов создать более сложные структуры.

Поведенческие – определяют способы взаимодействия между классами и объектами.

**Creational:**

**Factory method (фабричный метод)** – определяет интерфейс для создания объектов некого базового класса, но решение о том, объект какого класса базового типа инстанциировать определяется наследниками фабричного метода.

Определяется интерфейс фабрики с фабричным методом, реализации этой фабрики будут переопределять этот метод и возвращать специфичный объект базового типа.

Преимущества: Создание объектов отделено от основного кода (ослабление связанности), использование базового класса (упрощает добавление новых классов базового типа).

* В приспособленце также используется для их хранения и определения необходимости создания их создания.

**Abstract Factory (абстрактная фабрика)** – определяет интерфейс для создания семейств связанных объектов, не привязываясь к конкретным классам создаваемых объектов.

Определяется интерфейс фабрики, каждый метод которой специфично реализован наследником, каждая реализация сама решает, какие конкретные типы этого семейства будут определены.

Преимущества: Те же, что и у фабричного метода, плюс гарантирует, что создаваемые семейства типов будут сочетаемы.

* Различие этих фабрик заключается в том, что Abstract Factory используется для создания семейств объектов, factory method для одного объекта. Каждый метод абстрактной фабрики может быть реализован через фабричный метод или прототип.

**Singleton (Одиночка) –** гарантирует, что класс имеет только один экземпляр с глобальной точкой доступа.

Скрывается конструктор, создается static метод, который контролирует ЖЦ одиночки.

Проблемы: Нарушает SRP, имеет проблемы с многопоточностью, при тестировании требует создания Mock-объекта, может иметь проблемы глобальной переменной: глобальный доступ нарушает модульность программы (повышает связанность), изменяемое состояние, если его поля не final.

* Остальные порождающие паттерны могут реализованы с помощью одиночки (кроме абстрактного метода), также может быть реализован фасад.
* Отличие от глобальной переменной в том, что ее состояние можно изменить, синглтон гарантирует его неизменность (саму ссылку не изменить, но состояние полей можно, если они не final, плюс, если они не финал, то проблема с многопоточностью).

**Builder (Строитель) –** предназначен для того, чтобы вынести конструирование объекта за пределы собственного класса, таким образом, чтобы в результате одного и того же процесса конструирования могли получаться разные представления. Позволяет создавать разные варианты одного и того же объекта с помощью пошагового вызова методов вместо использования конструктора. Или же: предоставляет API для конструирования сложного объекта.

Builder:

Определяется интерфейс builder’а, в котором объявляются методы для установки полей этого объекта и метод получения результата. Его реализации определяют специфичные случаи конструирования этого объекта. Опционально может вводится класс директор, который определяет порядок строительства объектов.

* + Чем-то похож на абстрактную фабрику, только на уровне объекта.

Fluent Builder:

Конфигурируется с помощью вложенного статического класса, который имеет те же поля, что и внешний класс и реализует методы setter’ы для их конфигурации и метод build, который устанавливает значения builder во внешний класс. Запрещает создание объекта вне билдера.  
Предпочтителен, так как имеет удобный API в виде цепочки вызываемых методов.

**Prototype (Прототип) –** предназначен для создания объектов класса с помощью клонирования, не вдаваясь в подробности реализации.

Создание интерфейса с методом copy() или имплементация интерфейса Cloneable и метода clone().

Преимущества: Копирование объекта инкапсулировано в самом копируемом классе (уменьшение связанности, решается проблема недоступности приватных полей внешнему классу, позволяет избежать использования конструктора класса).

* Проблема всех порождающих паттернов, кроме Singleton и Fluent Builder в том, что они не инкапсулируют логику создания своих объектов (возможно создание объектов с помощью их конструкторов).

**Structural:**

**Adapter (Адаптер) –** позволяет конвертировать интерфейс класса (публичные методы) в другой интерфейс, ожидаемый клиентом. Т.е. выступает посредником между классом, который имеет нужный и реализованный функционал и интерфейсом, который необходим клиенту.

Реализует интерфейс, ожидаемый клиентом, на основе интерфейса или класса, который нужно адаптировать.

**\*Bridge (Мост) –** разделяетодин или несколько классов на две отдельные иерархии – абстракцию (слой делегирующий работу реализации) и реализацию, которые можно расширять независимо друг от друга.

(заменяет наследование агрегацией или композицией) - преимущество

Абстракция не должна зависеть от реализации, только от другой абстракции

Типичный пример – класс расширяется в 2х разных плоскостях и эти плоскости имеют множество комбинаций

Для получения преимуществ наследования без потери гибкости

**Decorator (Декоратор)** – предоставляет возможность динамического добавления нового поведения объекту, путем создания оболочки для расширяемого объекта, которая будет расширять функционал базового класса. Использует агрегацию или композицию взамен создания подклассов.

Должен предоставлять тот же (или расширенный) интерфейс, что и класс, функционал которого он расширяет (с помощью наследования), должен быть ассоциирован с объектом, который расширяет (иметь поле этого объекта).

Преимущества: Добавление нового функционала, не изменяя сам класс, использованию наследования предпочитает агрегацию или композицию.

**Facade (Фасад)** – предоставляет простой (иногда урезанный) интерфейс, который скрывает сложность системы и инкапсулирует логику взаимодействия этих классов, делая систему проще для использования.

**Преимущества**: создает точку доступа к сложной системе, изолируя клиентов от ее компонентов.

**Composite (Компоновщик) –** шаблон контейнер,позволяет сгруппировать множество объектов в древовидную структуру и работать с ней, как с одним объектом.

Агрегирует все компоненты в виде коллекции этих объектов и расширяет интерфейс этой группы объектов, путем делегирования работы дочерним компонентам (в реализации вызывает метод интерфейса компонентов).

**Flyweight (Легковес/Приспособленец) –** позволяет использовать разделяемые объекты (приспособленцы) с общим и неизменным состоянием сразу в нескольких контекстах.

Легковес **–** класс, в который выделено общее переиспользуемое состояние (должно быть immutable), фабрика легковесов – представлена Object pool (создает, хранит, если уже создан – выдает имеющийся).

* Возможно, string pool отражает идеи этого паттерна.

Информация, которая не хранится в легковесе является уникальной для каждого объекта

Преимущества: помогает избегать порождение одинаковых объектов (экономит оперативную память).

**Proxy (Заместитель/Суррогат) –** позволяет создавать объекты-прослойки между клиентом и сервисным объектом, который выполняет свою функцию, а затем делегирует вызов сервисному объекту. Управляет ЖЦ проксируемого класса.

Расширяет тот же интерфейс, что и сервисный класс, а также ассоциирован с ним. Метод расширяемого интерфейса выполняет свою функцию до или после передачи вызова сервисному объекту.

Применяется для кэширования, логирования, ленивой инициализации, защите доступа, удаленный прокси

Отличия от декоратора: Контролирует ЖЦ объекта (не передается извне – в конструкторе, а создается в классе, то есть не расширяет интерфейс, а следовательно клиент не видит разницу в использовании), не добавляется новый функционал к проксируемому классу (*поведение проксируемого класса не изменяется,* добавляется дополнительная механика типа кэширования, логирования и др.).