**1.Классификация паттернов:**

### Порождающие

Отвечают за удобное и безопасное создание новых объектов или даже целых семейств объектов.

### Структурные

Отвечают за построение удобных в поддержке иерархий классов.

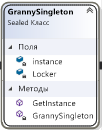
### Поведенческие

Решают задачи эффективного и безопасного взаимодействия между объектами программы.

**2.Одиночка (Singleton, Синглтон)** - порождающий паттерн, который гарантирует, что для определенного класса будет создан только один объект, а также предоставит к этому объекту точку доступа.

Применение:

Когда необходимо, чтобы для класса существовал только один экземпляр



**3.Потокобезопасная реализация благодаря свойствам статических полей:**

public sealed class SingletonThree

{

Private static readonly SingletonThree Instance= new SingletonThree();

Static SingletonThree() {}

private SingletonThree() {}

public static SingletonThree GetInstance()

{

Return Instance;

}

**С отложенной реализацией:**

public sealed class SingletonFour

{

Private static readonly Lazy<SingletonFour> Lazy =

New Lazy <SingletonFour>(() => new SingletonFour());

private SingletonFour() {}

public static SingletonFour GetInstance() => Lazy.Value;

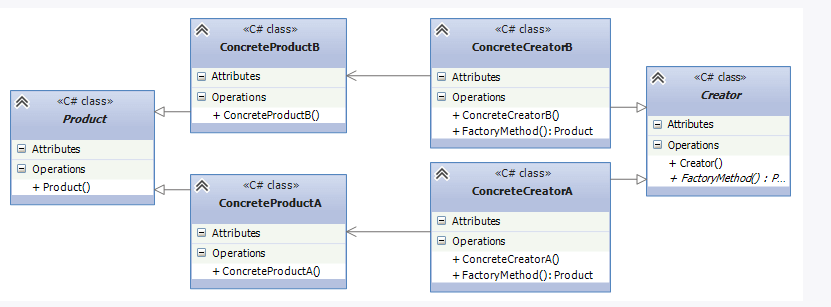
}

Lazy<T> гарантирует нам, что объект будет создан только тогда, когда в нем есть необходимость.

**4.Фабричный метод (Factory Method**) - это паттерн, который определяет интерфейс для создания объектов некоторого класса, но непосредственное решение о том, объект какого класса создавать происходит в подклассах. То есть паттерн предполагает, что базовый класс делегирует создание объектов классам-наследникам.

Применение:

1. Когда заранее неизвестно, объекты каких типов необходимо создавать
2. Когда система должна быть независимой от процесса создания новых объектов и расширяемой: в нее можно легко вводить новые классы, объекты которых система должна создавать.
3. Когда создание новых объектов необходимо делегировать из базового класса классам наследникам

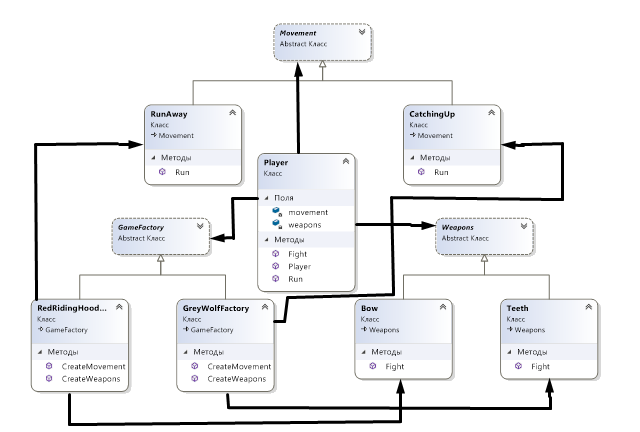


* Абстрактный класс Product определяет интерфейс класса, объекты которого надо создавать.
* Конкретные классы ConcreteProductA и ConcreteProductB представляют реализацию класса Product. Таких классов может быть множество
* Абстрактный класс Creator определяет абстрактный фабричный метод FactoryMethod(), который возвращает объект Product.
* Конкретные классы ConcreteCreatorA и ConcreteCreatorB - наследники класса Creator, определяющие свою реализацию метода FactoryMethod(). Причем метод FactoryMethod() каждого отдельного класса-создателя возвращает определенный конкретный тип продукта. Для каждого конкретного класса продукта определяется свой конкретный класс создателя.

**5.Паттерн "Абстрактная фабрика"** (Abstract Factory) предоставляет интерфейс для создания семейств взаимосвязанных объектов с определенными интерфейсами без указания конкретных типов данных объектов.

Применение:

* Когда система не должна зависеть от способа создания и компоновки новых объектов
* Когда создаваемые объекты должны использоваться вместе и являются взаимосвязанными

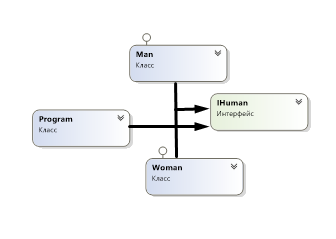


* Абстрактные классы Movement и Weapons определяют интерфейс для классов, объекты которых будут создаваться в программе.
* Конкретные классы RunAway /CatchingUp и Bow /Teeth представляют конкретную реализацию абстрактных классов
* Абстрактный класс фабрики GameFactory определяет методы для создания объектов.
* Конкретные классы фабрик RedRidingHoodFactory и GreyWolfFactory реализуют абстрактные методы базового класса и непосредственно определяют какие конкретные продукты использовать
* Класс клиента Player использует класс фабрики для создания объектов. При этом он использует исключительно абстрактный класс фабрики GameFactory и абстрактные классы продуктов Weapons и Movement и никак не зависит от их конкретных реализаций

**6.Паттерн Прототип (Prototype**) позволяет создавать объекты на основе уже ранее созданных объектов-прототипов. То есть по сути данный паттерн предлагает технику клонирования объектов.

Применение:

* Когда конкретный тип создаваемого объекта должен определяться динамически во время выполнения
* Когда нежелательно создание отдельной иерархии классов фабрик для создания объектов-продуктов из параллельной иерархии классов (как это делается, например, при использовании паттерна Абстрактная фабрика)
* Когда клонирование объекта является более предпочтительным вариантом нежели его создание и инициализация с помощью конструктора. Особенно когда известно, что объект может принимать небольшое ограниченное число возможных состояний.

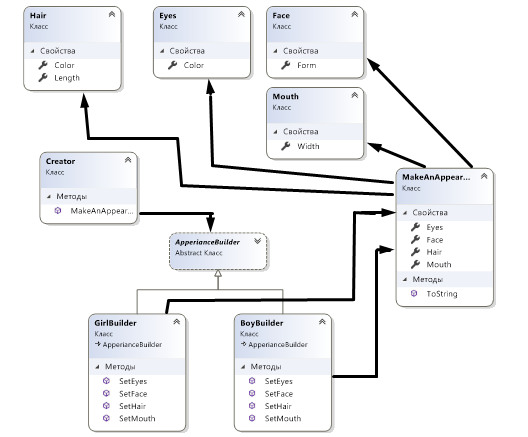


* IHuman: определяет интерфейс для клонирования самого себя, который, как правило, представляет метод Clone()
* Man и Woman: конкретные реализации прототипа. Реализуют метод Clone()
* Program: создает объекты прототипов с помощью метода Clone()

**7.Строитель (Builder)** - шаблон проектирования, который инкапсулирует создание объекта и позволяет разделить его на различные этапы.

### Когда использовать паттерн Строитель?

* Когда процесс создания нового объекта не должен зависеть от того, из каких частей этот объект состоит и как эти части связаны между собой
* Когда необходимо обеспечить получение различных вариаций объекта в процессе его создания



* MakeAnApperiance: представляет объект, который должен быть создан.
* ApperianceBuilder: определяет интерфейс для создания различных частей объекта MakeAnAppriance
* GirlBuilder / BoyBuilder: конкретная реализация Buildera. Создает объект MakeAnApperiance и определяет интерфейс для доступа к нему
* Сreator: распорядитель - создает объект, используя объекты Builder

**8.** Пул –это набор готовых к использованию объектов. Если клиенту требуется объект, он берёт его из пула. Когда объект больше не нужен, он не уничтожается, а возвращается в пул (при этом состояние объекта сбрасывается до начального).

Шаблон, в основном, применяется для повышения производительности.

9.Отложенная инициализация (Lazy initialization)

**Назначение**: паттерн позволяет отсрочить действия, связанные с созданием объекта, до момента, когда непосредственно потребуется результат этих действий.  
Паттерн используется, если создание объекта связано с большими затратами ресурсов, или если есть вероятность, что объект или его часть не будут использованы.