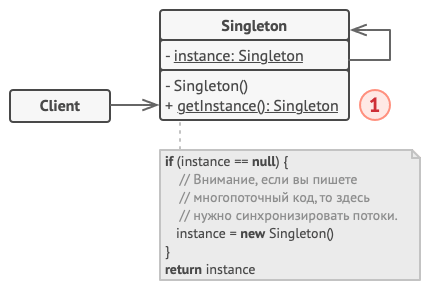
1. **Какие типы паттернов бывают?**

Параждающие

Архитектурные

Пведенчиские

1. **Нарисуете диаграмму классов и поясните принцип работы паттерна Singleton. Назовите условия применения.  
   **Одиночка**** — это порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.



определяет статический метод getInstance, который возвращает единственный экземпляр своего класса.

Пример:  
Правительство государства — хороший пример одиночки. В государстве может быть только одно официальное правительство. Вне зависимости от того, кто конкретно заседает в правительстве, оно имеет глобальную точку доступа «Правительство страны N»

1. **Как сделать потокобезопасную, с отложенной реализацией версию Singleton.**

public sealed class Singleton

{

/\*Явный статический конструктор сообщает компилятору C#не помечать тип как beforefieldinit\*/

static Singleton() { }

private Singleton() { }

private static readonly Singleton source = new Singleton();

public static Singleton Source

{

get

{

return source;

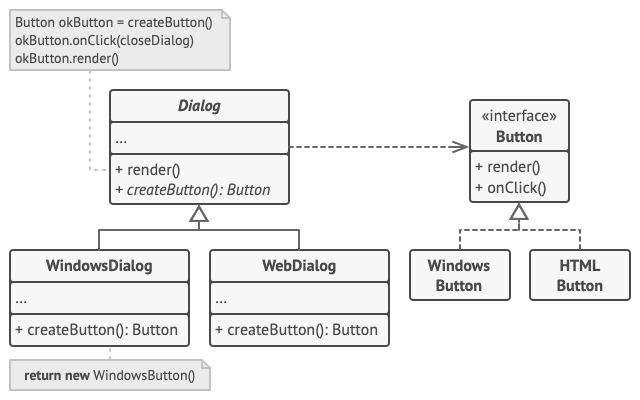
}

}

}

1. **Нарисуете диаграмму классов и поясните принцип работы паттерна Factory Method. Назовите условия применения**

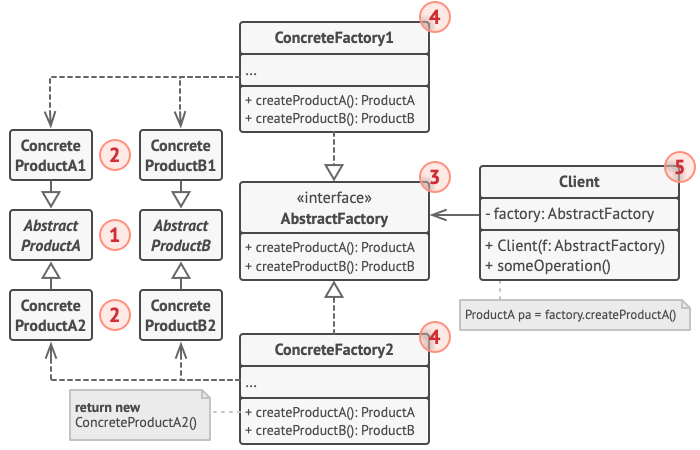
****Фабричный метод**** — это порождающий паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов



Фабричный метод объявлен в классе диалогов. Его подклассы относятся к различным операционным системам. Благодаря фабричному методу, вам не нужно переписывать логику диалогов под каждую систему. Подклассы могут наследовать почти весь код из базового диалога, изменяя типы кнопок и других элементов, из которых базовый код строит окна графического пользовательского интерфейса

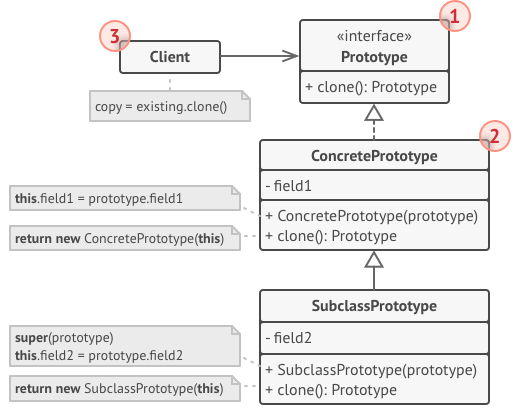
1. **Нарисуете диаграмму классов и поясните принцип работы паттерна Abstract Factory. Назовите условия применения**

****Абстрактная фабрика**** — это порождающий паттерн проектирования, который позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.



1. **Нарисуете диаграмму классов и поясните принцип работы паттерна Prototype. Назовите условия применения**

****Прототип**** — это порождающий паттерн проектирования, который позволяет копировать объекты, не вдаваясь в подробности их реализации.



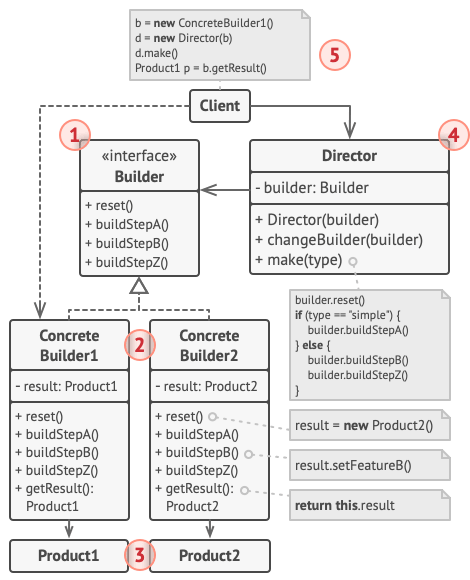
****Интерфейс прототипов**** описывает операции клонирования. В большинстве случаев — это единственный метод clone.

****Конкретный прототип**** реализует операцию клонирования самого себя. Помимо банального копирования значений всех полей, здесь могут быть спрятаны различные сложности, о которых не нужно знать клиенту. Например, клонирование связанных объектов, распутывание рекурсивных зависимостей и прочее.

****Клиент**** создаёт копию объекта, обращаясь к нему через общий интерфейс прототипов.

1. **Нарисуете диаграмму классов и поясните принцип работы паттерна Builder. Назовите условия применения**

****Строитель**** — это порождающий паттерн проектирования, который позволяет создавать сложные объекты пошагово. Строитель даёт возможность использовать один и тот же код строительства для получения разных представлений объектов.



****Интерфейс строителя**** объявляет шаги конструирования продуктов, общие для всех видов строителей.

****Конкретные строители**** реализуют строительные шаги, каждый по-своему. Конкретные строители могут производить разнородные объекты, не имеющие общего интерфейса.

****Продукт**** — создаваемый объект. Продукты, сделанные разными строителями, не обязаны иметь общий интерфейс.

****Директор**** определяет порядок вызова строительных шагов для производства той или иной конфигурации продуктов.

Обычно ****Клиент**** подаёт в конструктор директора уже готовый объект-строитель, и в дальнейшем данный директор использует только его. Но возможен и другой вариант, когда клиент передаёт строителя через параметр строительного метода директора. В этом случае можно каждый раз применять разных строителей для производства различных представлений объектов.

1. **Что такое Пул объектов (Object pool), в чем суть паттерна?**

Объектный пул (англ. object pool) — порождающий паттерн (шаблон) проектирования, набор инициализированных и готовых киспользованию объектов. Когда системе требуется объект, он не создаётся, а берётся из пула. Когда объектбольше не нужен, он не уничтожается, а возвращается в пул.

1. **Какое основное назначение паттерна отложенная инициализация (Lazy initialization)**

 когда некоторая ресурсоёмкая операция (создание объекта, вычисление значения) выполняется непосредственно перед тем, как будет использован её результат. Таким образом, инициализация выполняется «по требованию», а незаблаговременно. Аналогичная идея находит применение в самых разных областях: например, компиляция «на лету»и логистическая концепция «Точно в срок».

**https://prog-cpp.ru/uml-classes/**

**<https://refactoring.guru/ru/design-patterns/factory-method>**

**<https://refactoring.guru/ru/design-patterns/abstract-factory>**

**<https://refactoring.guru/ru/design-patterns/prototype>**

**https://refactoring.guru/ru/design-patterns/builder**