1. Какие типы паттернов бывают?

Паттерн представляет определенный способ построения программного кода для решения часто встречающихся проблем проектирования.

**Порождающие паттерны** — это паттерны, которые абстрагируют процесс инстанцирования или, иными словами, процесс порождения классов и объектов. Среди них выделяются следующие:

* **Абстрактная фабрика (Abstract Factory)**
* **Строитель (Builder)**
* **Фабричный метод (Factory Method)**
* **Прототип (Prototype)**
* **Одиночка (Singleton)**

**Структурные паттерны** рассматривают, как классы и объекты образуют более крупные структуры - более сложные по характеру классы и объекты. К таким шаблонам относятся:

* **Адаптер (Adapter)**
* **Мост (Bridge)**
* **Компоновщик (Composite)**
* **Декоратор (Decorator)**
* **Фасад (Facade)**
* **Приспособленец (Flyweight)**
* **Заместитель (Proxy)**

**Поведенческие паттерны** определяют алгоритмы и взаимодействие между классами и объектами, то есть их поведение. Среди подобных шаблонов можно выделить следующие:

* **Цепочка обязанностей (Chain of responsibility)**
* **Команда (Command)**
* **Интерпретатор (Interpreter)**
* **Итератор (Iterator)**
* **Посредник (Mediator)**
* **Хранитель (Memento)**
* **Наблюдатель (Observer)**
* **Состояние (State)**
* **Стратегия (Strategy)**
* **Шаблонный метод (Template method)**
* **Посетитель (Visitor)**

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

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

Когда надо использовать Синглтон? Когда необходимо, чтобы для класса существовал только один экземпляр

Синглтон позволяет создать объект только при его необходимости. Если объект не нужен, то он не будет создан. В этом отличие синглтона от глобальных переменных.

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

Если вы используете версию .NET Framework 4 (или выше), вы можете использовать **тип System.Lazy<T>**, чтобы реализовать ленивую загрузку очень просто. Все, что вам нужно сделать, это передать делегат конструктору, который вызывает конструктор Singleton, которому передается лямбда-выражение:

1. public sealed class Singleton
2. {
3. private Singleton() { }
4. private static readonly Lazy<Singleton> lazy =
5. new Lazy<Singleton>(() => new Singleton());
6. public static Singleton Source { get { return lazy.Value; } }
7. }
8. Это довольная простая реализация, которая хорошо работает. Она также позволяет вам проверить, был ли экземпляр создан с использованием *свойства IsValueCreated*, если вам это нужно.

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

**Когда надо применять паттерн**

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

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

### Когда использовать абстрактную фабрику

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

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

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

Когда использовать Прототип?

* 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.

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

### 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

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

[Пул объектов](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D1%8B%D0%B9_%D0%BF%D1%83%D0%BB) — порождающий шаблон проектирования, набор инициализированных и готовых к использованию объектов.

Основная идея паттерна Object Pool состоит в том, чтобы избежать создания новых экземпляров класса в случае возможности их повторного использования.

**UML-диаграмма классов паттерна Object Pool**

* **Reusable** - экземпляры классов в этой роли взаимодействуют с другими объектами в течение ограниченного времени, а затем они больше не нужны для этого взаимодействия.
* **Client** - экземпляры классов в этой роли используют объекты Reusable.
* **ReusablePool** - экземпляры классов в этой роли управляют объектами Reusable для использования объектами Client.

9. Какое основное назначение паттерна отложенная инициализация (Lazy initialization)?

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