1. Перечислите и поясните принципы ***SOLID***.

**Single responsibility**

Каждый класс должен иметь только одну причину для изменения.

**Open-closed**

классы должны быть расширяемы (наследование), но неизменяемы

**Liskov substitution**

Должна быть возможность вместо базового типа (класса) подставить любой его (класс-наследник), при этом работа программы не должна измениться.

**Interface segregation**

слишком «толстые» интерфейсы необходимо разделять на более маленькие

**Dependency inversion**

Модули высокого уровня не должны зависить от модулей низкого уровня, все они должны зависить от абстракций. Абстракции не должны зависить от деталей, детали должны зависить от абстракций.

1. Поясните детально, каким образом ***Dependency Inversion Principle*** применен в реализации класса ***Employee***.

Согласно принципу инверсии зависимостей, модули высокого уровня, такие как Employee, не должны напрямую зависеть от модулей низкого уровня, таких как IBonus, Bonus1A и Bonus1C. Вместо этого они должны зависеть от абстракций, что и реализовано в данном примере через использование интерфейса IBonus.

1. Поясните суть паттерна ***Abstract Factory***, что дает его применение.

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

1. Сравните два паттерна ***Factory Method*** и ***Abstract Factory.***

Краткое сравнение паттернов Factory Method и Abstract Factory:

1. \*\*Factory Method (Фабричный метод)\*\*:

- Основной целью паттерна является создание объектов определенного типа внутри субклассов.

- Он определяет метод, который подклассы должны реализовать для создания экземпляров классов.

- Реализация паттерна применяется на уровне одиночного метода.

- Позволяет делегировать ответственность за создание объектов подклассам, сохраняя при этом общий интерфейс.

2. \*\*Abstract Factory (Абстрактная фабрика)\*\*:

- Основной целью паттерна является создание семейств взаимосвязанных объектов без указания их конкретных классов.

- Он определяет интерфейс для создания семейств связанных или зависимых объектов, не специфицируя их конкретных классов.

- Реализация паттерна применяется на уровне семейств связанных объектов.

- Позволяет создавать объекты различных типов, причем все создаваемые объекты соответствуют одной и той же абстракции.

Итак, основное различие между этими паттернами заключается в том, что Factory Method фокусируется на создании одного объекта, позволяя подклассам решать, какой именно объект создавать, в то время как Abstract Factory фокусируется на создании семейства взаимосвязанных объектов, предоставляя абстрактный интерфейс для их создания.

1. Порождающие паттерны.

Factory Method (Фабричный метод)

паттерн проектирования, который определяет общий интерфейс для создания объектов в суперклассе, позволяя подклассам изменять тип создаваемых объектов.

Пример с методом IFactory где возвращаем IBonus а в фабриках это меняется на конкретные типы.

Abstract Factory (Абстрактная фабрика)

паттерн проектирования, который позволяет создавать семейства связанных объектов, не привязываясь к конкретным классам создаваемых объектов.

Builder (Строитель)

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

Prototype (Прототип)

паттерн проектирования, который позволяет копировать объекты, не вдаваясь в подробности их реализации.

Singleton(Одиночка)

паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

Отношение ассоциации используют, чтобы показать, что между классами (например, между двумя классами) существует некоторая связь. Обычно с помощью него на диаграмме классов показывают, что один класс пользуется функционалом другого класса.

Отношение композиции является частным случаем отношения агрегации. Однако у него есть одно отличие – классы-части, которые он соединяет с классом-целым, **не могут существовать обособленно.**