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

**S: Single Responsibility Principle (Принцип единственной ответственности).**

Класс должен быть ответственен лишь за одну часть программы.

**O: Open-Closed Principle (Принцип открытости-закрытости).**

Программные сущности (классы, модули, функции) должны быть открыты для расширения, но закрыты для модификации. Это достигается за счет абстракций и полиморфизма.

**L: Liskov Substitution Principle (Принцип подстановки Барбары Лисков).**

Необходимо, чтобы подклассы могли бы служить заменой для своих суперклассов.

Цель этого принципа заключаются в том, чтобы классы-наследники могли бы использоваться вместо родительских классов, от которых они образованы, не нарушая работу программы. Если оказывается, что в коде проверяется тип класса, значит принцип подстановки нарушается.

**I: Interface Segregation Principle (Принцип разделения интерфейса).**

Не должно быть нереализованных интерфейсов. Этот принцип направлен на устранение недостатков, связанных с реализацией больших интерфейсов. Нужно разбивать на более маленькие.

**D: Dependency Inversion Principle (Принцип инверсии зависимостей).**

*Объектом зависимости должна быть абстракция, а не что-то конкретное.*

1. Модули верхних уровней не должны зависеть от модулей нижних уровней. Оба типа модулей должны зависеть от абстракций.
2. Абстракции не должны зависеть от деталей. Детали должны зависеть от абстракций.
3. Поясните детально, каким образом ***Dependency***

***Inversion Principle*** применен в реализации класса ***Employee***.

В данной реализации класса Employee, принцип инверсии зависимостей (Dependency Inversion Principle, DIP) применяется путем инъекции зависимости через конструктор.

Интерфейс IBonus:

Во-первых, есть интерфейс IBonus, который определяет контракт для любого класса, реализующего его. Этот интерфейс вероятно имеет метод calc, который принимает количество отработанных часов и возвращает величину бонуса.

Класс Employee:

Класс Employee имеет свойство Bonus, тип которого - интерфейс IBonus. Это позволяет классу Employee работать с любым объектом, который реализует интерфейс IBonus, не зависимо от его конкретной реализации. Таким образом, Employee зависит от абстракции (IBonus), а не от конкретной реализации бонуса.

Конструктор Employee:

Конструктор класса Employee принимает параметр типа IBonus. Это означает, что при создании объекта Employee необходимо предоставить объект, реализующий интерфейс IBonus. Таким образом, зависимость Employee от конкретной реализации бонуса инвертирована - не класс Employee создает объект бонуса, а он получает его извне.

Метод CalcBonus:

Метод CalcBonus класса Employee использует метод calc объекта Bonus, который был передан через конструктор. Таким образом, Employee не знает, как именно рассчитывается бонус, и ему не важно, какая конкретная реализация IBonus используется.

Это позволяет легко заменять конкретные реализации бонусов без изменения класса Employee. При необходимости добавления новых типов бонусов или изменения существующих, можно создать новый класс, реализующий интерфейс IBonus, и передать его в конструктор Employee, не затрагивая его внутреннюю логику. Таким образом, принцип инверсии зависимостей обеспечивает гибкость и расширяемость кода.

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.

Суть паттерна Abstract Factory заключается в создании интерфейса, который определяет методы для создания семейств взаимосвязанных или взаимозависимых объектов без указания их конкретных классов. При этом каждая конкретная реализация интерфейса фабрики будет создавать свои собственные объекты, образующие семейство.

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

Factory Method: Этот паттерн определяет интерфейс для создания объекта, но делегирует сам процесс создания подклассам. Таким образом, каждый подкласс может реализовать свой собственный способ создания объекта. Это позволяет расширять или изменять логику создания объектов, не изменяя кода клиента. Factory Method работает с одним продуктом.

Abstract Factory: В отличие от Factory Method, Abstract Factory определяет интерфейс для создания семейств взаимосвязанных объектов, а не отдельных объектов. То есть, он предоставляет абстрактную фабрику, которая содержит методы для создания различных объектов. Это позволяет создавать семейства объектов, в которых все компоненты взаимосвязаны друг с другом. Abstract Factory работает с несколькими продуктами, образуя семейство.

Таким образом, основное различие между ними заключается в том, что Factory Method работает с одним продуктом и делегирует создание этого продукта подклассам, а Abstract Factory работает с семействами продуктов и предоставляет интерфейс для создания всех компонентов этого семейства.

1. Как (какие классы, методы и как) необходимо доработать библиотеку ***Lec03LibN:***

*-*если появится новый тип вознаграждения *D(x,y,z)= (wH+a)\*cH\*x+y\*z, действующий, только для уровня 2;*

*Создаем класс и в уровень 2 добавляем*

*- если появится новый уровень вознаграждения 4(a,b,c)= (wH+a\*c)\*(cH+b), действующий, только для уровня 1;*

Только в уровне 1 создаем класс

*- если перестанет применяться уровень вознаграждения типа B уровня 3.*

Будет ругаться программа, заглушка, public, можно исключения вызывать. Удалить логику