Турсунов Баходурхон

**1.  Как принцип единой ответственности (SRP) помогает улучшить разработку программного обеспечения и в чем его основная идея?**

В целом принципы Solid – это не паттерны, их нельзя назвать какими-то определенными догмами, которые надо обязательно применять при разработке, однако их использование позволят улучшить код программы, упростить возможные его изменения и поддержку.

Принцип единой ответственности (Single Responsibility Principle) можно сформулировать так:

Каждый компонент должен иметь одну и только одну причину для изменения.

В C# в качестве компонента может выступать класс, структура, метод. А под обязанностью здесь понимается набор действий, которые выполняют единую задачу. То есть суть принципа заключается в том, что класс/структура/метод должны выполнять одну единственную задачу. Весь функционал компонента должен быть целостным, обладать высокой связностью.

**2.  Почему SRP иногда трудно реализовать, особенно когда обязанности класса не сразу ясны?**

Нередко принцип единой обязанности нарушает при смешивании в одном классе функциональности разных уровней. Например, класс производит вычисления и выводит их пользователю, то есть соединяет в себе бизнес-логику и работу с пользовательским интерфейсом. Либо класс управляет сохранением/получением данных и выполнение над ними вычислений, что также нежелательно. Класс следует применять только для одной задачи – либо бизнес-логика, либо вычисления, либо работа с данными.

Другой распространенный случай – наличие в классе или его методах абсолютно несвязанного между собой функционала.

**3.  Как принцип открытости/закрытости (OCP) способствует расширению программных модулей, не требуя их модификации?**

1. Открытость для расширения. Это означает, что программные сущности должны быть гибкими для расширения новым функционалом путем добавления нового кода, а не изменения существующего.

2. закрытость для модификации. Существующий код должен быть стабильным и не подвержен изменениям при добавлении нового функционала. Это позволяет избежать нарушения работы существующего кода и минимизирует риск внесения ошибок.

3. использование полиморфизма. Полиморфизм позволяет создавать абстрактные интерфейсы или базовые классы, которые определяют общие методы. Затем новые классы могут реализовывать этот интерфейс или наследоваться от базового класса, чтобы добавить новую функциональность.

**4.  Почему в соответствии с OCP важно, чтобы модули были открыты для расширения, но закрыты для модификации?**

Открытость для расширения позволяет добавлять новый функционал без изменения существующего кода.

Закрытость для модификации обеспечивает стабильность и защищает от внесения ошибок в существующий код. Это важно для сохранения стабильности и гибкости, и расширяемости

В результате, следуя принципу OCP, мы стремимся к созданию гибкого, расширяемого и стабильного программного обеспечения.

**5.  Какая фундаментальная концепция лежит в основе принципа замещения Лискова (LSP), когда речь идет о подклассах и базовых классах?**

Фундаментальная концепция, лежащая в основе принципа замещения Лискова, когда речь идет о подклассах и базовых классах, заключается в следующем:

Подтипы должны быть заменяемы на свои базовые типы. Это означает, что объекты подклассов можно использовать вместо объектов базовых классов без изменения корректности программы. Другими словами, поведение подклассов должно быть согласовано с поведением базовых классов таким образом, чтобы их можно было использовать вместо базовых классов в любом месте программы, где используется базовый класс, и при этом не нарушать работу программы.

Иными словами, подтипы (подклассы) должны сохранять контракт базовых типов (базовых классов). Это гарантирует, что использование наследования не нарушает функциональность программы и не вводит нежелательные побочные эффекты.

**6.  Как принцип замещения Лискова (LSP) гарантирует, что родительские экземпляры могут заменять дочерние экземпляры без неожиданного поведения?**

Принцип замещения Лискова гарантирует, что родительские экземпляры могут заменять дочерние экземпляры без неожиданного поведения благодаря следующим моментам:

**Согласованное поведение**: Все подклассы должны согласованно реализовывать интерфейсы и контракты базовых классов. Это означает, что методы подклассов должны вести себя так же, как и методы базовых классов, не нарушая предполагаемого поведения.

**Сохранение инвариантов:** Инварианты, определенные базовым классом, должны сохраняться в подклассах. Это означает, что предусловия, постусловия и инварианты, определенные базовым классом, должны также выполняться и для подклассов.

**Не добавление новых предусловий:** Подклассы не должны вводить новые предусловия для методов, которые уже определены в базовых классах. Это означает, что подклассы могут уточнять предусловия, но не менять или ужесточать их.

**Не ухудшение постусловий:** Подклассы не должны ухудшать постусловия для методов, которые уже определены в базовых классах. Это означает, что подклассы могут уточнять постусловия или возвращать более конкретные типы данных, но не менять или ослаблять постусловия.

**Преобразование результатов методов:** Результаты методов подклассов должны быть совместимы с результатами методов базовых классов. Это гарантирует, что код, использующий базовые классы, будет корректно работать и с подклассами.

Practice:

Реализуйте небольшой программный проект или модифицируйте существующий, чтобы продемонстрировать применение принципов SOLID, уделяя особое внимание SRP, OCP и LSP. Ниже приведены шаги для практического занятия:

1.  Выберите небольшой программный проект или модуль, над которым вы уже работали, или создайте новый, если это необходимо.

2.  Определите класс или компонент в проекте, который мог бы выиграть от следования принципу единой ответственности (SRP). Переделайте этот класс так, чтобы он в большей степени соответствовал SRP, разделив его обязанности на более мелкие, более сфокусированные классы.

3.  Если возможно, найдите возможность применить в проекте принцип открытости/закрытости (OCP). Внесите изменения, чтобы обеспечить расширяемость без изменения существующего кода.

4.  Проверьте соблюдение принципа замещения Лискова (LSP) в иерархии классов вашего проекта. Убедитесь, что подклассы могут быть заменены на базовые классы, не вызывая неожиданного поведения.