







**Вторая группа вопросов:**

1. ООП декомпозиция - это процесс разбиения сложной системы на более простые и управляемые части, которые называются объектами. В объектно-ориентированном программировании (ООП) декомпозиция позволяет создавать модульные и расширяемые системы, где каждый объект выполняет определенную функцию и взаимодействует с другими объектами для достижения общей цели. Декомпозиция также способствует повторному использованию кода и облегчает поддержку и модификацию программного обеспечения.
2. Статический полиморфизм - это механизм в языках программирования, позволяющий использовать одно и то же имя функции или оператора для различных типов данных или аргументов. Статический полиморфизм достигается через перегрузку функций и операторов, шаблоны (в C++), а также через обобщенные типы (generics) в языках программирования, таких как Java и C#.

Перегрузка функций позволяет определять несколько функций с одинаковым именем, но различающимися по количеству или типу параметров. Компилятор выбирает нужную функцию на этапе компиляции, основываясь на типах аргументов.

Шаблоны (или обобщения) в языке C++ позволяют создавать обобщенные функции и классы, которые могут работать с различными типами данных без необходимости написания отдельного кода для каждого типа.

1. Инкапсуляция – принцип ООП, который заключается в том, что объект должен скрывать свою внутреннюю реализацию от других объектов и предоставлять только необходимый интерфейс для взаимодействия с ним. Инкапсуляция позволяет изолировать изменения внутренней реализации объекта от клиентского кода, что обеспечивает более гибкую и надежную систему. Также она способствует повышению безопасности и защите данных, так как она позволяет скрыть приватные данные и методы от несанкционированного доступа.

**Третья группа вопросов:**

1. Абстрактная фабрика (Abstract Factory) - это паттерн проектирования, который обеспечивает интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов без указания их конкретных классов. Он использует принцип инкапсуляции, чтобы скрыть детали создания объектов от клиента.

Пример использования на практике:

Предположим, у нас есть приложение, которое работает с различными видами баз данных (например, SQL и NoSQL). Мы можем использовать абстрактную фабрику для создания набора объектов, таких как Connection, Statement и ResultSet, для каждой конкретной базы данных. Клиентский код будет использовать интерфейс абстрактной фабрики для создания этих объектов, не зная конкретных классов баз данных.

1. Адаптер (Adapter) - это паттерн проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе. Он использует принцип инкапсуляции, чтобы скрыть детали преобразования интерфейса от клиента.

Пример использования на практике:

Предположим, у нас есть старая библиотека, которая предоставляет функциональность через устаревший интерфейс, и новый код должен использовать современный интерфейс. Мы можем создать адаптер, который реализует новый интерфейс и делегирует вызовы старому объекту через его устаревший интерфейс. Таким образом, клиентский код может использовать новый интерфейс, не зная о деталях адаптации старой библиотеки.

Оба этих паттерна проектирования демонстрируют использование инкапсуляции для скрытия деталей реализации от клиентского кода и обеспечивают гибкость и расширяемость системы.