Подобедов Роман 6 группа 7 вариант

Нулевой блок:

1. Win32AP. Для использования данного API в языке c++ подключаются библиотеки Windows.h, conio.h, process.h. В них определено множество директив #define, необходимых для работы с API (напр. переопределены базовые типы данных)
2. В операционной системе Windows термин "поток" (thread) означает выполнение инструкций программы в контексте определенного процесса. Потоки представляют собой базовые единицы выполнения внутри процесса и делят ресурсы процесса, такие как память и файлы. Каждый процесс в Windows содержит по крайней мере один поток, называемый главным потоком. Потоки внутри одного процесса могут совместно использовать данные и ресурсы, но у каждого потока есть свой собственный указатель на инструкции, стек вызовов и регистры процессора. Это позволяет потокам параллельно выполняться, улучшая общую производительность многозадачных приложений. Операционная система Windows поддерживает множество функций для работы с потоками, таких как создание, управление приоритетами выполнения, синхронизация и т. д. Потоки часто используются для распараллеливания работы в приложениях, улучшения отзывчивости и обеспечения эффективного использования ресурсов компьютера.
3. Мьютекс (Mutex) представляет собой объект синхронизации в операционных системах, в том числе в Windows. Название "мьютекс" происходит от термина "взаимное исключение" (mutual exclusion), что означает, что он используется для предотвращения конфликтов доступа к ресурсам, когда несколько потоков или процессов пытаются одновременно получить доступ к общим данным. Мьютекс имеет два основных состояния: заблокирован (locked) и разблокирован (unlocked). Когда поток (или процесс) захватывает мьютекс, он переводится в заблокированное состояние, и другие потоки, пытающиеся захватить этот мьютекс, будут ожидать, пока он не освободится. Это обеспечивает взаимное исключение, предотвращая одновременный доступ к критическим секциям кода или ресурсам.
4. Событие (Event) в операционных системах, включая Windows, представляет собой объект синхронизации, используемый для оповещения или синхронизации между потоками или процессами. Событие может находиться в одном из двух состояний: сигнальном или несигнальном. Основная идея событий заключается в том, что один поток (или процесс) может устанавливать событие в сигнальное состояние, а другие потоки (или процессы) могут ждать, когда оно станет сигнальным, чтобы продолжить выполнение своей работы. Событие может использоваться для сигнализации о том, что определенное условие выполнилось, и другие потоки могут приступить к выполнению своей задачи. В Windows события могут быть созданы с использованием функции API, такой как CreateEvent.

Второй блок:

1. ООП-декомпозиция относится к процессу разделения сложных систем на более простые, более управляемые компоненты, используя принципы объектно-ориентированного программирования. Декомпозиция в ООП предполагает разбиение системы на объекты, которые взаимодействуют друг с другом для выполнения задач. Основные принципы декомпозиции в ООП включают:

* Идентификация объектов: Выделение основных сущностей, которые являются частью системы. Эти сущности становятся объектами в программе.
* Выделение функциональности: Разделение функциональности системы на методы и операции, которые выполняются объектами. Каждый объект обладает своими собственными методами для выполнения определенных действий.
* Определение интерфейсов: Установление взаимодействия между объектами через интерфейсы. Интерфейс определяет, какие операции объект может выполнить, но не предоставляет деталей реализации.
* Инкапсуляция: Скрытие деталей реализации объекта от внешнего мира. Объекты предоставляют только необходимые интерфейсы для работы с ними, что упрощает управление кодом и обеспечивает модульность.
* Наследование и полиморфизм: Использование наследования для создания отношений между объектами и возможности переопределения методов (полиморфизм) для адаптации поведения в подклассах.

Декомпозиция в ООП позволяет разработчикам разбивать сложные системы на более мелкие, самодостаточные и переиспользуемые компоненты. Это способствует повышению читаемости, поддерживаемости и масштабируемости кода, а также облегчает работу в команде, где разные члены могут быть ответственны за отдельные компоненты системы.

1. Статический полиморфизм - это форма полиморфизма в программировании, при которой решение о выборе конкретной реализации метода или функции принимается на этапе компиляции, а не во время выполнения программы. Он также называется компиляционным полиморфизмом или ранним связыванием. Одним из основных механизмов статического полиморфизма является перегрузка функций и методов. В языках программирования, таких как C++ или Java, можно объявить несколько функций с одинаковыми именами, но с разными типами параметров или разными количествами параметров. Компилятор различает эти функции на этапе компиляции и выбирает подходящую реализацию, когда вызывается функция в коде.
2. Инкапсуляция — один из ключевых принципов объектно-ориентированного программирования, который означает объединение данных и методов, которые работают с этими данными, в единый компонент, называемый классом. Одной из целей инкапсуляции является сокрытие внутренних деталей реализации класса от внешнего мира и предоставление интерфейса для взаимодействия с объектом.

Основные принципы инкапсуляции:

* Скрытие данных: Инкапсуляция позволяет объединять данные и методы работы с ними внутри класса. Таким образом, детали реализации остаются скрытыми от внешнего кода.
* Интерфейс доступа: Класс предоставляет интерфейс, через который другие части программы могут взаимодействовать с его данными и методами. Этот интерфейс определяет, какие операции можно выполнять с объектом, но не раскрывает, как именно они реализованы.
* Контроль доступа: Инкапсуляция обеспечивает уровень контроля доступа к данным класса. Методы могут быть открытыми (public), защищенными (protected) или закрытыми (private), определяя, кто и как может использовать или изменять данные.

Третий блок:

1. Фабричный метод: Фабричный метод — это паттерн проектирования, который предоставляет интерфейс для создания объекта в суперклассе, но позволяет подклассам изменять тип создаваемых объектов. В контексте инкапсуляции, фабричный метод помогает скрыть детали создания объектов от клиентского кода.

Ключевые аспекты инкапсуляции:

* Интерфейс суперкласса: Суперкласс предоставляет абстрактный метод (фабричный метод), который должен быть реализован подклассами. Этот метод используется для создания конкретных объектов.
* Создание объекта в подклассах: Каждый подкласс реализует фабричный метод таким образом, чтобы он создавал экземпляр конкретного объекта, соответствующего контексту подкласса.
* Клиентский код использует интерфейс суперкласса: Клиентский код взаимодействует с объектами через интерфейс суперкласса, вызывая фабричный метод. Это позволяет клиенту работать с абстракцией создания объекта, не заботясь о конкретных классах.

1. Template Method: Template Method — это паттерн проектирования, который определяет "скелет" алгоритма в суперклассе, перекладывая реализацию некоторых шагов на подклассы. В контексте инкапсуляции, Template Method помогает инкапсулировать детали реализации алгоритма, разделяя его на основные шаги.

Ключевые аспекты инкапсуляции:

* Абстрактный метод (шаги алгоритма): Суперкласс определяет абстрактные методы, представляющие шаги алгоритма. Эти методы должны быть реализованы подклассами.
* Шаблон метода: Суперкласс содержит финальный метод, представляющий "скелет" алгоритма. Этот метод вызывает абстрактные методы в определенной последовательности, формируя конечный алгоритм.
* Инкапсуляция деталей реализации: Подклассы реализуют абстрактные методы, предоставляя конкретные детали реализации для каждого шага алгоритма. Детали реализации скрыты от клиентского кода, который взаимодействует только с шаблоном метода.