1. Основные идеи ООП: использование объекта в качестве основной компоненты программы и децентрализация управления, реализуемая представлением программы совокупностью взаимодействующих объектов

2. Сущность ООП: моделирование поведения объектов реального мира, наглядное представление объектов, создание и использование АТД, производные АТД

3. Концепции ООП. Инкапсуляция и расширяемость типов

Инкапсуляция и расширяемость типов представляют собой важные концепции объектно-ориентированного программирования (ООП), обеспечивающие безопасность, гибкость и поддерживаемость программного кода.

**Инкапсуляция:**

* Это механизм, позволяющий скрыть детали реализации объекта от внешнего мира и предоставить доступ к ним только через определенный интерфейс (методы).
* Объединение данных и методов, работающих с этими данными, внутри класса.
* Инкапсуляция позволяет обеспечить безопасность и упростить использование объектов, скрывая сложные детали реализации и предоставляя стандартизированный способ взаимодействия.

**Расширяемость типов:**

* Это способность типов быть расширенными или измененными путем добавления новой функциональности или изменения существующей без изменения самого типа.
* Расширяемость достигается через механизмы наследования и полиморфизма.
* Например, новые классы могут быть созданы на основе существующих классов путем расширения их функциональности или изменения их поведения.

4. Концепции ООП. Семантика объекта: данные как отражение состояния объекта и методы как средства обеспечения взаимодействия объектов. Представление данных. Классификация методов: конструкторы, деструкторы, селекторы, модификаторы

Семантика объекта включает в себя представление данных как состояния объекта и методы как средства взаимодействия между объектами.

**Отражение состояния объекта:**

* **Данные:** Представляют состояние объекта. Это переменные (или поля), которые хранят информацию о состоянии объекта в определенный момент времени. Например, для объекта "автомобиль" данные могут включать в себя его скорость, количество топлива и т.д.

**Методы как средства взаимодействия объектов:**

* **Методы:** Представляют собой функции, которые определены внутри класса и могут работать с данными объекта. Они обеспечивают возможность взаимодействия объектов между собой и изменения их состояния. Например, для объекта "автомобиль" методы могут включать в себя функции ускорения, торможения и заправки топливом.

**Представление данных:**

* **Инкапсуляция:** Механизм, позволяющий объединить данные и методы, работающие с ними, внутри класса. Это обеспечивает контролируемый доступ к данным и скрытие их реализации от внешнего мира.

**Классификация методов:**

1. **Конструкторы:** Методы, вызываемые при создании объекта. Они инициализируют начальное состояние объекта и могут принимать аргументы для установки начальных значений.
2. **Деструкторы:** Методы, вызываемые при уничтожении объекта. Они освобождают ресурсы, занимаемые объектом, и могут выполнять различные завершающие операции.
3. **Селекторы (геттеры):** Методы, предоставляющие доступ к данным объекта. Они возвращают значения полей объекта, но не изменяют его состояние.
4. **Модификаторы (сеттеры):** Методы, позволяющие изменять состояние объекта. Они обновляют значения полей объекта, изменяя его состояние.

5. Концепции ООП. Классы объектов: назначение и семантика, классы и абстрактные типы данных, наглядное представление классов. Объекты как экземпляры классов. Основные действия с объектами: создание, инициализация, обработка, разрушение

Классы объектов представляют собой шаблоны или чертежи для создания объектов в объектно-ориентированном программировании (ООП). Они объединяют данные (поля) и функции (методы), которые работают с этими данными.

**Назначение и семантика классов:**

* **Назначение:** Определение структуры данных и поведения объектов. Классы служат для создания объектов с определенными свойствами и функциональностью.
* **Семантика:** Классы предоставляют семантику для создания объектов определенного типа, обладающих определенными характеристиками и поведением.

**Классы и абстрактные типы данных:**

* Классы часто используются для создания абстрактных типов данных (ADT), которые определяются своими методами и интерфейсом, но скрывают свою реализацию. ADT предоставляют интерфейс для работы с данными, но не раскрывают внутренние детали реализации.

**Наглядное представление классов:**

* Классы могут быть представлены в виде схемы, где показаны его поля и методы. **Объекты как экземпляры классов:**
* Объекты являются конкретными экземплярами классов, созданными на основе их описания (шаблона).

**Основные действия с объектами:**

1. **Создание:** Использование конструктора для выделения памяти и инициализации объекта.
2. **Инициализация:** Установка начальных значений для полей объекта, например, через методы или конструкторы.
3. **Обработка:** Выполнение операций с объектом, вызов методов для изменения его состояния или получения информации.
4. **Разрушение:** Использование деструктора для освобождения памяти, занятой объектом, при его уничтожении или выходе из области видимости.

6. Концепции ООП. Конструкторы и деструкторы, основное назначение

**Конструкторы:** Специальные методы класса, вызываемые при создании нового экземпляра объекта. Основное назначение - инициализация начальных значений свойств объекта.

**Деструкторы:** Также методы класса, вызываемые перед уничтожением объекта. Их основное назначение - освобождение ресурсов, выделенных объекту во время его жизни.

7. Концепции ООП. Перегрузка операторов и функций. Ключевые слова: friend, operator

**Перегрузка операторов:** Изменение поведения стандартных операторов для пользовательских типов данных.

**Перегрузка функций:** Создание нескольких функций с одним именем, но разными параметрами или типами возвращаемых значений.

**Ключевые слова:**

* **Friend:** Позволяет функциям или классам получать доступ к приватным членам других классов.
* **Operator:** Используется для перегрузки операторов, определяет новое поведение для операций между объектами классов

8. Концепции ООП. Шаблоны и обобщенное программирование. Ключевое слово: template. Контейнерные классы

Шаблоны и обобщенное программирование позволяют создавать универсальные классы и функции, работающие с разными типами данных без необходимости дублирования кода.

**Шаблоны (template):** Это механизм, позволяющий создавать обобщенные типы и функции, которые могут работать с различными типами данных. Ключевое слово **template** используется для определения шаблонов классов или функций в языках программирования, таких как C++.

**Обобщенное программирование:** Подход, при котором создаются абстрактные структуры данных или алгоритмы, способные работать с разными типами данных. Это позволяет писать универсальный код, который может быть переиспользован для различных типов.

**Контейнерные классы:** Это классы, реализующие структуры данных для хранения коллекций объектов. Они обычно используют обобщенное программирование и шаблоны для обеспечения универсальности и могут включать в себя списки, векторы, стеки, очереди и другие структуры данных. Например, в C++ контейнеры из стандартной библиотеки (**std::vector**, **std::list**, **std::map** и т. д.) основаны на шаблонах и позволяют хранить объекты различных типов.

9. Концепции ООП. Основные составляющие STL: контейнеры, итераторы, алгоритмы

STL (Standard Template Library) - это часть стандартной библиотеки языка программирования C++, которая предоставляет набор шаблонов классов и функций для обработки данных. Она включает в себя три основные составляющие:

**1. Контейнеры:**

* Предоставляют реализацию структур данных для хранения коллекций объектов.
* Включают в себя векторы, списки, множества, карты и другие структуры данных.
* Позволяют удобно хранить и управлять коллекциями объектов различных типов.

**2. Итераторы:**

* Используются для обхода элементов в контейнерах.
* Предоставляют унифицированный способ доступа к элементам контейнера и позволяют выполнять операции над ними.
* Подобно указателям, позволяют перемещаться между элементами структуры данных.

**3. Алгоритмы:**

* Предоставляют набор функций для выполнения различных операций над контейнерами.
* Включают алгоритмы сортировки, поиска, трансформации и многие другие.
* Работают с контейнерами через итераторы, что позволяет применять операции к элементам контейнера без знания его внутренней структуры.

10. Концепции ООП. Наследование: базовый и производный класс, иерархия классов

Концепция наследования в объектно-ориентированном программировании позволяет одному классу (производному классу) использовать свойства и методы другого класса (базового класса).

**Базовый класс:** Это класс, который передает свои свойства и методы производному классу. Он может содержать общую функциональность, которая может быть использована несколькими производными классами.

**Производный класс:** Это класс, который наследует свойства и методы от базового класса. Он может расширять функциональность базового класса, добавляя новые свойства или методы, или переопределяя существующие.

**Иерархия классов:** Это структура, в которой классы организованы в виде дерева, где базовый класс является родительским для одного или нескольких производных классов, которые, в свою очередь, могут быть родительскими для других классов.

11. Концепции ООП. Полиморфизм. Абстрактный базовый класс

Полиморфизм в объектно-ориентированном программировании позволяет объектам различных классов использовать одно и то же имя для выполнения различных задач. Абстрактный базовый класс играет ключевую роль в реализации полиморфизма.

**Полиморфизм:** Это способность объекта использовать методы базового класса для выполнения своих особых задач. Это может происходить благодаря наследованию и переопределению методов в производных классах.

**Абстрактный базовый класс:** Это класс, который содержит хотя бы один чисто виртуальный метод (без реализации) и не может быть инстанциирован (т.е., создан экземпляр этого класса). Он служит в качестве шаблона для производных классов и определяет интерфейс для этих классов.

12. Концепции ООП. Обработка исключительных ситуаций, проверка утверждений

Обработка исключительных ситуаций и проверка утверждений - это методы обеспечения безопасности и надежности программы путем управления потенциальными ошибками.

**Обработка исключительных ситуаций:** Это механизм, позволяющий программе обрабатывать ошибки или исключительные ситуации, возникающие во время выполнения, и принимать соответствующие действия. Если происходит ошибка, программа может сгенерировать исключение и передать управление блоку кода, предназначенному для обработки этого типа ошибок.

**Проверка утверждений:** Это специальные инструкции или механизмы, используемые для проверки утверждений или предположений в коде программы. Они представляют собой выражения, которые должны быть истинными во время выполнения программы. Если проверка утверждения не проходит успешно (логическое выражение оценивается как ложное), программа может аварийно завершиться с сообщением об ошибке или предупреждением.

13. Концепции ООП. Преимущества и недостатки ООП. Основные свойства языка ООП. Место языка Си++ во внедрении и развитии средств ООП. Значение языка Си++ для профессиональной подготовки программистов

**Преимущества и недостатки ООП:**

**Преимущества:**

1. **Модульность:** Разделение кода на модули повышает его читаемость и управляемость.
2. **Повторное использование кода:** Возможность использовать и расширять готовые компоненты.
3. **Гибкость и расширяемость:** Упрощение изменения и добавления нового функционала без изменения всей программы.
4. **Упрощение сопровождения:** Более понятная структура кода и уменьшение его сложности.
5. **Наследование и полиморфизм:** Позволяют более гибко организовывать код.

**Недостатки:**

1. **Сложность:** Более сложная архитектура и понимание, особенно для начинающих.
2. **Переиспользование:** Может быть затруднено из-за сложности иерархий наследования.
3. **Производительность:** Излишнее использование ООП-концепций может привести к потере производительности.

**Основные свойства языка ООП:**

1. **Инкапсуляция:** Скрытие деталей реализации и предоставление интерфейса.
2. **Наследование:** Возможность создания новых классов на основе уже существующих.
3. **Полиморфизм:** Возможность объектов с одним интерфейсом иметь различные формы.

**Место языка C++ во внедрении и развитии средств ООП:**

C++ играет значительную роль в мире ООП. Он считается одним из основополагающих языков, который успешно интегрировал в себя концепции ООП. Благодаря поддержке инкапсуляции, наследования, полиморфизма и других принципов ООП, C++ является мощным инструментом для разработки приложений, особенно в области системного программирования, игровой индустрии и разработки операционных систем.

**Значение языка C++ для профессиональной подготовки программистов:**

C++ представляет большую ценность для профессиональной подготовки программистов по нескольким причинам:

* **Гибкость:** Обучение C++ обеспечивает понимание базовых концепций ООП, что упрощает изучение других языков программирования.
* **Производительность:** Разработка на C++ требует понимания работы с памятью, что полезно при работе с более высокоуровневыми языками.
* **Широкое применение:** C++ широко используется в различных областях разработки, что делает его знание востребованным для программистов.

14. Классы: компонентные данные, компонентные функции, способы задания доступа, синтаксическая структура определения класса, объявление, определение и инициализация объектов, объекты объявленные как константы

15. Область видимости класса: оператор разрешения области видимости ( : : ) - унарная форма, бинарная форма; вложенные классы - классы, вложенные в классы, классы, вложенные в блоки, вложенные определения функций

16. Компонентные данные: назначение, основные свойства, синтаксис и семантика определений компонентных данных, статические компонентные данные, статические константы, изменяемые компонентные данные (mutable)

17. Компонентные функции: назначение, основные свойства, доступ к компонентным функциям, доступ компонентных функций к компонентным данным - функции селекторы, функции модификаторы

18. Встроенные компонентные функции: назначение, синтаксис и семантика определения, рекомендации по использованию; компонентные функции, определяемые вне класса: назначение, синтаксис и семантика определения и объявления, рекомендации по использованию

19. Указатель this, компонентные функции типа static и const (статические и постоянные компонентные функции): назначение, синтаксис и семантика определения и объявления, рекомендации по использованию

20. Конструкторы: назначение, наличие нескольких конструкторов у одного класса - причины, разделение функций, способы задания конструктора: по умолчанию или явное

21. Явное описание (определение или объявление) конструктора, размещение определения в теле класса и влияние размещения на функции конструктора, размещение определения вне своего класса, параметры конструктора и возможности их использования

22. Конструкторы для инициализации объектов, инициализаторы конструктора; конструкторы для преобразования типов, ключевое слово explicit

23. Конструкторы умолчания: назначение, способы задания, правила описания

24. Конструкторы копирования: назначение, способы задания, правила описания

25. Вызов конструктора: назначение, способы реализации, задание аргументов; явный и автоматический вызовы: ситуации использования, рекомендации по применению

26. Деструкторы: назначение, основные свойства, способы задания: по умолчанию или явное

27. Явное описание (определение или объявление) деструктора, размещение определения в теле класса и влияние размещения на функции деструктора, размещение определения вне своего класса, параметры деструктора и возможности их использования

28. Вызов деструктора:

назначение, способы реализации, явный и автоматический вызовы: ситуации использования, рекомендации по применению

29. Специальный полиморфизм – преобразование АТД к встроенному типу

30. Специальный полиморфизм – перегрузка функций, алгоритм выбора перегруженной функции

31. Дружественные функции: основные свойства, номенклатура, необходимость, полезность и опасность применения, наборы дружественных функций, дружественные классы

32. Дружественные функции: объявление – размещение, синтаксис, семантика, требования к параметрам и типу возвращаемого значения; определение – размещение, область видимости; вызов функции

33. Специальный полиморфизм – перегрузка операторов: синтаксис, семантика, приоритет и порядок выполнения, доступные и недоступные для перегрузки операторы, перегружаемые как обычные функции, только как компонентные функции

34. Перегрузка унарных и бинарных операторов: синтаксис и семантика

35. Перегрузка операторов присваивания и индексирования: синтаксис и семантика

36. Перегрузка операторов ввода-вывода («, »): синтаксис и семантика

37. Перегрузка оператора () для индексирования: синтаксис и семантика

38. Перегрузка операторов доступа к элементу класса через указатель на объект : синтаксис и семантика

39. Перегрузка операторов new и delete: синтаксис и семантика

40. Параметрический полиморфизм – шаблонные классы и шаблонные функции: назначение, параметризованные типы данных, синтаксис и семантика

41. Шаблоны классов: назначение, синтаксис и семантика, объявление и определение шаблона, дружественные шаблонному классу функции, статические элементы шаблонного класса, аргументы шаблона класса

42. Шаблоны функций: назначение, синтаксис и семантика, объявление, определение, вызов и перегрузка шаблонных функций, аргументы шаблона функции