ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.В.16 Объектно-ориентированное программирование

Направление подготовки (специальность) 27.03.04 Управление в технических системах

Профиль образовательной программы Интеллектуальные системы обработки информации и управления

Квалификация (степень) выпускника бакалавр

Форма обучения очная

СОДЕРЖАНИЕ

3.1 Практическая работа №1 Объекты формы в VBA: Список и Поле со списком
3.1 Практическая работа №1 Объекты формы в VBA: Рисунок
3.1 Практическая работа №1 Процедуры и функции
3.1 Практическая работа №1 Рекурсивные подпрограммы
3.1 Практическая работа №1 Класс TControl. Объект TEdit
3.1 Практическая работа №1 Список строк TStrings. Объект TMemo. Диалоговые окна сохранения и открытия файлов
3.1 Практическая работа №1 Группы радиокнопок
3.1 Практическая работа №1 Модули

1. КОНСПЕКТ ЛЕКЦИЙ

1.1. Лекция № 1 (2 часа)

Тема: «Эволюция методологий программирования »

1.1.1. Вопросы лекции:

1 Первое поколение языков программирования, развитие алгоритмических абстракций, модуль как единица построения программных систем, третье поколение языков.

2 Зарождение объектной модели, четвертое поколение языков.

3 Объектные и объектно-ориентированные языки программирования.

4 Парадигмы программирования

1.1.2. Краткое содержание вопросов:

1 Первое поколение языков программирования, развитие алгоритмических абстракций, модуль как единица построения программных систем, третье поколение языков.

В эволюции методологий разработки ПО можно выделить несколько периодов. На заре программирования каждая программа разрабатывалась вручную одним разработчиком, который и был полным ее хозяином. Возникшие по мере усложнения программ трудности в их программировании и отладке стимулировали развитие систем автоматизации программирования. Эти системы включали средства компиляции, а также ограниченный набор средств отладки и документирования. На этом этапе все средства были ориентированы на работу с текстами программ.

Второй период начался с момента, когда в связи с увеличением размеров программ стало резко возрастать число людей, разрабатывающих программу. Это потребовало организации коллективной работы на основе применения структурных методов и структурных методологий. Упорядочивая структуру и дисциплинируя разработку, они помогали стандартизации и систематизации разработки ПО и его сопровождения. Все это позволяло представить процесс создания программ как инженерную дисциплину, которая опиралась на формализованное понятие "жизненный цикл ПО". При этом в жизненном цикле центр тяжести начал перемещаться с процессов программирования и отладки программ на процесс проектирования общей структуры ПО и структуры его компонент, наряду с чем продолжалось активное развитие средств автоматизации программирования и отладки. Этот период может считаться временем, когда были сформулированы основные принципы методологии SE, понимаемые в смысле технологии разработки ПО или техники ПО , и начата реализация этих принципов.

Третий период в эволюции методологии характеризуется автоматизацией структурных методологий, т. е. созданием инструментальной поддержки трудоемких работ по проектированию ПО в целях максимального высвобождения времени разработчиков. Именно с этим периодом связывается появление первой генерации CASE-средств. Последние принесли новую жизнь структурным методологиям созданием автоматизированных графических средств для выпуска различных схем, диаграмм, экранных и бумажных изображений, наличием словарей данных и хранилищ, появлением средств анализа и контроля структуры и текстов, генераторов документов, генераторов программных кодов и др.

Следующий период связан с объединением средств автоматизации структурных методологий со средствами автоматизации программирования и отладки программ. По существу, речь идет о начале создания интегрированных систем поддержки полного жизненного цикла разработки ПО, который реализуется во второй генерации CASE-средств. В течение этого периода в рамках второй генерации CASE-средств активно развиваются и используются методы повторной разработки ПО. Имеющиеся достижения и результаты, по мнению специалистов, свидетельствуют о достаточно полной реализации принципов методологий SE. В то же время проблемы, возникающие при создании

современных программных систем, требуют перехода к более развитой методологии создания ПО.

С учетом сказанного последний период можно охарактеризовать как переход от методологии SE к методологии IE. На практике такой переход происходит достаточно плавно и постепенно путем ориентации методологии SE на централизованно-информационный подход к разработке ПО.

2 Зарождение объектной модели, четвертое поколение языков.

Методы структурного проектирования помогают упростить процесс разработки сложных систем за счет использования алгоритмов как готовых строительных блоков. Аналогично, методы объектно-ориентированного проектирования созданы, чтобы помочь разработчикам применять мощные выразительные средства объектного и объектно-ориентированного программирования, использующего в качестве блоков классы и объекты.

Объектно-ориентированный анализ и проектирование отражают эволюционное, а не революционное развитие проектирования; новая методология не порывает с прежними методами, а строится с учетом предшествующего опыта. К сожалению, большинство программистов в настоящее время формально и неформально натренированы на применение только методов структурного проектирования. Разумеется, многие хорошие проектировщики создали и продолжают совершенствовать большое количество программных систем на основе этой методологии. Однако алгоритмическая декомпозиция помогает только до определенного предела, и обращение к объектно-ориентированной декомпозиции необходимо. Более того, при попытках использовать такие языки, как C++ или Ada, в качестве традиционных, алгоритмически ориентированных, мы не только теряем их внутренний потенциал - скорее всего результат будет даже хуже, чем при использовании обычных языков C и Pascal. Дать электродрель плотнику, который не слышал об электричестве, значит использовать ее в качестве молотка. Он согнет несколько гвоздей и разобьет себе пальцы, потому что электродрель мало пригодна для замены молотка.

3 Объектные и объектно-ориентированные языки программирования.

- Smalltalk
- C++
- Common Lisp Object System (CLOS)
- Ada
- Eiffel
- Java
- Object Pascal

4 Парадигмы программирования.

Парадигма программирования — это совокупность подходов, методов, стратегий, идей и понятий, определяющая стиль написания программ.

Парадигма программирования в современной индустрии программирования очень часто определяется набором инструментов программиста (язык программирования и операционная система).

Парадигма программирования представляет (и определяет) то, как программист видит выполнение программы. Например, в объектно-ориентированном программировании программист рассматривает программу как набор взаимодействующих объектов, тогда как в функциональном программировании программа представляется в виде цепочки вычисления функций.

1.2. Лекция № 3 (2 часа)

Тема: «Понятие объекта»

1.2.1. Вопросы лекции:

1 Свойства, присущие объектам: состояние, поведение, идентичность.

1.2.2. Краткое содержание вопросов:

1 Свойства, присущие объектам: состояние, поведение, идентичность.

Состояние объекта характеризуется перечнем (обычно статическим) всех свойств данного объекта и текущими (обычно динамическими) значениями каждого из этих свойств.

К числу свойств объекта относятся присущие ему или приобретаемые им характеристики, черты, качества или способности, делающие данный объект самим собой. Например, для лифта характерным является то, что он сконструирован для поездок вверх и вниз, а не горизонтально.

Все свойства имеют некоторые значения. Эти значения могут быть простыми количественными характеристиками, а могут ссылаться на другой объект. Состояние лифта может описываться числом 3, означающим номер этажа, на котором лифт в данный момент находится.

Простые количественные характеристики (например, число 3) являются «постоянными, неизменными и непреходящими», тогда как объекты «существуют во времени, изменяются, имеют внутреннее состояние, преходящи и могут создаваться, уничтожаться и разделяться».

Тот факт, что всякий объект имеет состояние, означает, что всякий объект занимает определенное пространство (физически или в памяти компьютера).

Все объекты в системе инкапсулируют некоторое состояние, и все состояние системы инкапсулировано в объекты.

Поведение

Поведение — это то, как объект действует иреагирует; поведение выражается в терминах состояния объекта и передачи сообщений.

Иными словами, поведение объекта — это его наблюдаемая и проверяемая извне деятельность.

Операцией называется определенное воздействие одного объекта на другой с целью вызвать соответствующую реакцию. В чисто объектно-ориентированном языке, таком как Smalltalk, принято говорить о передаче сообщений между объектами. В языках типа C++ мы говорим, что один объект вызывает функцию-член другого. В основном понятие сообщение совпадает с понятием операции над объектами, хотя механизм передачи различен. В объектно-ориентированных языках операции, выполняемые над данным объектом, называются методами и входят в определение класса объекта. В C++ они называются функциями-членами.

Состояние объекта представляет суммарный результат его поведения.

Наиболее интересны те объекты, состояние которых не статично: их состояние изменяется и запрашивается операциями.

Идентичность

«Идентичность — это такое свойство объекта, которое отличает его от всех других объектов».

Они отмечают, что «в большинстве языков программирования и управления базами данных для различения временных объектов их именуют, тем самым путая адресуемость и идентичность. Большинство баз данных различают постоянные объекты по ключевому атрибуту, тем самым, смешивая идентичность и значение данных». Источником множества ошибок в объектно-ориентированном программировании является неумение отличать имя объекта от самого объекта.

1.3. Лекция № 3 (2 часа)

Тема: «Разработка Visual Basic-приложений».

1.3.1. Вопросы лекции:

1 Создание программного интерфейса пользователя.

1.3.2. Краткое содержание вопросов:

1 Создание программного интерфейса пользователя

После создания проекта открылась наша первая форма.

Форма это основной объект графического интерфейса (окно приложения), на её основе мы будем создавать интерфейс для пользователя.

Вы можете изменить размеры представленной формы и другие её свойства.

Свойства формы отображаются в окне Свойства в правой части среды разработки. Эти свойства разделены на группы и их довольно много. Некоторые свойства имеют свои подсвойства. Например, свойство Font (Шрифт) имеет подсвойства: Name, Size и др.

1.4. Лекция № 4 (2 часа)

Тема: «Сведения о классах и наследовании. Основы визуального программирования интерфейса».

1.4.1. Вопросы лекции:

1 Принцип замещения Лисковой.

2 Одиночное наследование: понятие производного класса, управление доступом в производных классах, конструкторы и деструкторы, абстрактные классы и виртуальные функции, виртуальный полиморфизм, информация о типе на этапе выполнения RTTI.

3 Зависимость по времени жизни.

4 Использование и зависимость от интерфейсов.

5 Объекты при передаче параметров и возврате из методов.

1.4.2. Краткое содержание вопросов:

Реализация поведения объектов на примере добавления функций-членов в структуры.

Пользователям, по-видимому, понадобится широкий набор операций над объектами типа Screen: возможность перемещать курсор, проверять и устанавливать области экрана и рассчитывать его реальные размеры во время выполнения, а также копировать один объект в другой. Все эти операции можно реализовать с помощью функций-членов.

Функции-члены класса объявляются в его теле. Это объявление выглядит точно так же, как объявление функции в области видимости пространства имен. (Напомним, что глобальная область видимости – это тоже область видимости пространства имен.

Структура как вырожденный класс.

Конструкция struct языка C не была принята в языке программирования Java потому, что класс выполняет все то же самое, что может делать структура, и даже более того. Структура группирует несколько полей данных в один общий объект, тогда как класс связывает с полученным объектом операции, а также позволяет скрывать поля данных от пользователей объекта. Иными словами, класс может инкапсулировать encapsulate) свои данные в объекте, доступ к которому осуществляется только через его методы.

Структура объявления класса.

Классы и объекты в C++ являются основными концепциями объектно-ориентированного программирования — ООП. Объектно-ориентированное программирование — расширение структурного программирования, в котором основными концепциями являются понятия классов и объектов. Основное отличие языка программирования C++ от C состоит в том, что в C нет классов, а следовательно язык C не поддерживает ООП, в отличие от C++.

Чтобы понять, для чего же в действительности нужны классы, проведём аналогию с каким-нибудь объектом из повседневной жизни, например, с велосипедом. Велосипед — это объект, который был построен согласно чертежам. Так вот, эти самые чертежи играют роль классов в ООП. Таким образом классы — это некоторые описания, схемы, чертежи по которым создаются объекты. Теперь ясно, что для создания объекта в ООП необходимо сначала составить чертежи, то есть классы. Классы имеют свои функции, которые называются методами класса. Передвижение велосипеда осуществляется за счёт вращения педалей, если рассматривать велосипед с точки зрения ООП, то механизм вращения педалей — это метод класса. Каждый велосипед имеет свой цвет, вес, различные составляющие — всё это свойства. Причём у каждого созданного объекта свойства могут различаться. Имея один класс, можно создать неограниченно количество объектов (велосипедов), каждый из которых будет обладать одинаковым набором методов, при этом можно не задумываться о внутренней реализации механизма вращения педалей, колёс, срабатывания системы торможения, так как всё это уже будет определено в классе. Разобравшись с назначением класса, дадим ему грамотное определение.

Доступ к членам класса.

Поддержка свойства инкапсуляции в классе дает два главных преимущества. Во-первых, класс связывает данные с кодом. И во-вторых, класс предоставляет средства для управления доступом к его членам. Именно эта, вторая преимущественная особенность и будет рассмотрена в данной статье.

В языке C#, по существу, имеются два типа членов класса: открытые и закрытые, хотя в действительности дело обстоит немного сложнее. Доступ к открытому члену свободно осуществляется из кода, определенного за пределами класса. А закрытый член класса доступен только методам, определенным в самом классе. С помощью закрытых членов и организуется управление доступом.

Ограничение доступа к членам класса является основополагающим этапом объектно-ориентированного программирования, поскольку позволяет исключить неверное использование объекта. Разрешая доступ к закрытым данным только с помощью строго определенного ряда методов, можно предупредить присваивание неверных значений этим данным, выполняя, например, проверку диапазона представления чисел. Для закрытого члена класса нельзя задать значение непосредственно в коде за пределами класса. Но в то же время можно полностью управлять тем, как и когда данные используются в объекте. Следовательно, правильно реализованный класс образует некий "черный ящик", которым можно пользоваться, но внутренний механизм его действия закрыт для вмешательства извне.

Поля данных класса как механизм реализации состояния объекта.

Как уже говорилось выше, в современных объектно-ориентированных языках программирования каждый объект является значением, относящимся к определённому классу. Класс представляет собой объявленный программистом составной тип данных, имеющий в составе:

Поля данных

Параметры объекта (конечно, не все, а только необходимые в программе), задающие его состояние. Поля данных объекта называют свойствами объекта. Физически свойства представляют собой значения (переменные, константы), объявленные как принадлежащие классу.

Методы

Процедуры и функции, связанные с классом. Они определяют действия, которые можно выполнять над объектом такого типа, и которые сам объект может выполнять.

Функции члена класса как механизм реализации поведения объекта.

Функция-член класса может содержать спецификатор virtual. Такая функция называется виртуальной. Спецификатор virtual может быть использован только в объявлениях нестатических функций-членов класса.

Если некоторый класс содержит виртуальную функцию, а производный от него класс содержит функцию с тем же именем и типами формальных параметров, то обращение к этой функции для объекта производного класса вызывает функцию, определённую именно в производном классе. Функция, определённая в производном классе, вызывается даже при доступе через указатель или ссылку на базовый класс. В таком случае говорят, что функция производного класса подменяет функцию базового класса. Если типы функций различны, то функции считаются разными, и механизм виртуальности не включается. Ошибкой является различие между функциями только в типе возвращаемого значения.

Спецификаторы доступа для обеспечения инкапсуляции.

Начиная с этой лекции мы будем подробно рассматривать принципы объектно-ориентированного программирования только на примере языка C#. Язык C# является полноценным объектно-ориентированным языком программирования со строгим контролем типов. Мы будем рассматривать объектно-ориентированное программирование только для языка C#, поскольку этот язык поддерживает все основные конструкции объектно-ориентированного подхода за исключением множественного наследования.

В C# объектно-ориентированное программирование поддерживается с помощью классов. Класс - это пользовательский тип данных, который включает в себя как данные, так и программный код в виде функций. Данные хранимые в классе называются полями, а функции класса называются методами. Также класс может содержать свойства и индексаторы, которые в определенном смысле объединяют поля и методы. После того, как класс описан, можно создавать переменные с типом этого класса.

Средства управления жизнью объекта.

Методология управления, менеджмента – это логическая схема управленческой деятельности, предполагающая взаимосвязанное понимание целей, ориентиров, а также средств и способов их достижения. Это еще и умение видеть, распознавать, понимать, оценивать и учитывать зависимости, которые и раскрывают содержание проблем, и подсказывают пути их решения.

Специалисты выделяют следующие компоненты, характеризующие содержание методологии управления: подходы, парадигмы, проблемы, приоритеты, ориентиры, критерии, альтернативы, процедуры выбора, средства и методы управления, а также ограничения.

Подход – наиболее принципиальный компонент методологии, определяющий выбор и использование остальных ее компонентов. Среди подходов в управлении особо выделяется системный подход. Весьма популярны в управлении программно-целевой и проектный подходы. Для современного менеджмента характерен маркетинговый подход, ориентированный на потребителя, кроме того, для него свойственны кибернетический, информационный, гуманистический подход – при этом заметен акцент на преимущественном использовании отдельных наук, типов ресурсов и соответствующей методологии в целом. Многие исследователи выделяют интеграционный и сетевой подходы. Принимая на вооружение тот или иной подход, можно говорить о науке управления и об управлении как искусстве; выбранный подход в таком случае обычно конкретизируется через те или иные принципы управления.

Парадигма (от греч. paradeigma пример, образец) – исходная концептуальная схема, система понятий, отражающая осмысление существенных черт действительности, модель постановки проблем и их решения, выбора соответствующих методов, господствующая в научном сообществе в течение определенного исторического периода и знаменующая

собой определенный этап в развитии теории (управленческий рационализм Ф.Тейлора, функциональная дифференциация А. Файоля и М. Вебера и др.) В современной парадигме управления, несмотря на разнообразие конкретных формулировок, предпочтение отдается человеческой личности, с учетом процессов глобализации и с акцентом на управление знаниями, на сетевые, партнерские принципы взаимодействия.

Конструкторы и деструкторы.

Конструктор - функция, предназначенная для инициализации объектов класса.Рассмотрим класс date:

```
class date {
  int day, month, year;
public:
  set(int, int, int);
};
```

Нигде не утверждается, что объект должен быть инициализирован, и программист может забыть инициализировать его или сделать это дважды.

ООП дает возможность программисту описать функцию, явно предназначенную для инициализации объектов. Поскольку такая функция конструирует значения данного типа, она называется конструктором. Конструктор всегда имеет то же имя, что и сам класс. Когда класс имеет конструктор, все объекты этого класса будут инициализироваться.

```
class date  {
  int day, month, year;
public:
  date(int, int, int); // конструктор
};
```

10 Описание селекторов и модификаторов.

Селектор — имя элемента разметки

Когда автор Web-узла хочет определить общий стиль всех страниц, он просто прописывает стили для всех элементов HTML-разметки, которые будут использоваться на страницах. Это дает возможность скомпоновать страницы из логических элементов, а стиль отображения элементов описать во внешнем файле.

Такой способ создания сайта позволяет автору изменять внешний вид всех страниц путем внесения изменений в файл описания стилей, а не в файлы HTML-страниц.

Внешний файл при этом может выглядеть следующим образом:

```
I, EM {color:#003366;font-style:normal;}
A I {font-style:normal;font-weight:bold;
     text-decoration:line-through;}
```

В первой строке этого описания перечислены селекторы-элементы, которые будут отображаться одинаково:

```
<I>Это курсив</I> и это тоже <EM>курсив</EM>
```

Последняя строка определяет стиль отображения вложенного в гипертекстовую ссылку курсива:

```
<A NAME=empty><I>intuit</I></A>
```

В данном случае переопределение состоит в том, что текст отображается внутри гипертекстовой ссылки перечеркнутым, причем жирным шрифтом.

Селектор — имя класса

Имя класса не является каким-либо стандартным именем элемента HTML-разметки. Оно определяет описание класса элементов разметки, которые будут отображаться одинаково. Для того, чтобы отнести элемент разметки к тому или иному классу, нужно воспользоваться его атрибутом CLASS (открыть):

```
<STYLE>
.test {color:white;background-color:black;}
</STYLE>
...
<P CLASS="test">
Этот параграф мы отобразим белым цветом по
черному фону
</P>
...
<P>
Эту <A CLASS="test">гипертекстовую ссылку</A>
мы отобразим белым цветом по черному фону.
</P>
```

Ассоциация и агрегация объектов и классов.

Отношения между классами (объектами)

По моей практике преподавания вопрос отношений между классами (или объектами) почему-то вызывает проблемы. Из-за чего так происходит, не могу сказать. Когда я изучал ООП, мне это показалось настолько очевидным, что не стоило даже упоминания — вся логика ООП не просто кричит, она вопиет об очевидности таких отношений и правилах их построения. Но тем не менее этот вопрос часто требует объяснений. Чем мы сейчас и займемся, прежде чем продолжим изучение новых конструкций языка Java.

Теория ООП выделяет три основных отношения между классами:

Ассоциация

Агрегация и композиция

Обобщение/Расширение (наследование)

Последний пункт мы с вами уже рассматривали, так что сосредоточимся на первых двух.

Ассоциация

Ассоциация означает, что объекты двух классов могут ссылаться один на другой, иметь некоторую связь между друг другом. Например Менеджер может выписать Счет. Соответственно возникает ассоциация между Менеджером и Счетом. Еще пример — Преподаватель и Студент — т.е. какой-то Студент учится у какого-то Преподавателя. Ассоциация и есть описание связи между двумя объектами. Студент учится у Преподавателя. Идея достаточно простая — два объекта могут быть связаны между собой и это надо как-то описать.

Агрегация и композиция

Агрегация и композиция на самом деле являются частными случаями ассоциации. Это более конкретизированные отношения между объектами.

Агрегация — отношение когда один объект является частью другого. Например Студент входит в Группу любителей физики.

Композиция — еще более «жесткое отношение, когда объект не только является частью другого объекта, но и вообще не может принадлежат еще кому-то. Например Машина и Двигатель. Хотя двигатель может быть и без машины, но он вряд ли сможет быть в двух или трех машинах одновременно. В отличии от студента, который может входить и в другие группы тоже. Такие описания всегда несколько условны, но тем не менее.

Зависимость по времени жизни.

«Внедрение зависимостей» и «Статическую типизацию» трудно назвать лучшими друзьями на все времена. Некоторые наработки в этом направлении существуют и легко гуглятся, однако интересно, насколько реально создать собственную простую реализацию баз хаков, ловких ухищрений и подключения внешних библиотек. Без особой гибкости, вот чтобы буквально два действия — настройка и внедрение. Вопросы многопоточности затрагиваться не будут, чтобы не отвлекаться от основной идеи. Итак, что же лично я хочу от внедрения зависимостей.

Постановка

Управление временем жизни объектов. Нюансы, касающиеся времени жизни, не должны разбавлять основную логику приложения. Чаще всего мне не нужно знать, что требуемый объект является экземпляром класса-одиночки или создается с помощью какого-то фабричного метода. Просто нужен пригодный для использования экземпляр. Совсем хорошо, если существует возможность изменять правила управления временем жизни объекта, не затрагивая основную логику приложения.

Тестирование. Одна из первоочередных целей внедрения зависимостей. Подразумевает возможность в подменять определенные объекты тестовыми заглушками, опять же, не затрагивая основную реализацию.

Упрощенный доступ к объектам. Информация, какой объект откуда можно получить чаще всего совершенно неинтересна. Более того, она отвлекает от базовой функциональности и способствует более сильному связыванию подсистем проекта, чем того хочется. Также нежелание программиста продумывать адекватные точки доступа к добавляемым сервисам может негативно сказаться на общей архитектуре системы. «В последние дни я получал почти все нужные мне объекты из модуля номер N, закину и эти туда же...».

Использование и зависимость от интерфейсов.

Когда вы разрабатываете класс в C++, первое, что вы должны продумать — открытый интерфейс для класса. Открытый интерфейс определяет, как ваш класс будет использоваться другими программистами (или вами), и после разработки и реализации он должен, как правило, оставаться постоянным. Вы можете добавлять что-то в интерфейс, но как только вы начали использовать класс, будет трудно удалить функции из общего интерфейса (если только они не используются и не были необходимы в первую очередь).

Но это не означает, что вы должны включать больше функциональности в своем классе, только чтобы позже решить, что удалить из интерфейса. Если вы так делаете, вы просто усложняете использование класса. Люди будут задавать вопросы вроде: «Почему есть четыре способа сделать это? Какой из них лучше? Какой способ выбрать мне?» Как правило, легче делать все просто и обеспечить один способ, если нет веских причин предложить вашему классу несколько методов с одинаковой функциональностью.

В то же время, только потому, что добавление методов для открытого интерфейса (вероятно) ничего не испортит, это не означает, что вы должны начинать с крошечного интерфейса. Прежде всего, если кто-нибудь решит унаследоваться от вашего класса, а в последствии вы выберете функцию с тем же именем, то возникнет путаница. Во-первых, если вы не объявите функцию виртуальной, то объект подкласса выберет функцию в зависимости от статического типа указателя. Это может создать проблемы. Кроме того, если вы объявите ее виртуальной, то у вас, возможно, возникнет проблема с тем, что она может предоставлять другую функциональность, отличную от оригинальной реализации

этой функции. И, наконец, вы просто не можете добавлять чисто виртуальные функции классу, который уже используется, потому что никто из унаследовавших от него не будет реализовывать эту функцию.

Открытый интерфейс должен оставаться постоянным как можно дольше. В самом деле, хороший подход к проектированию классов — написать интерфейс до реализации, потому что это то, что определяет, как ваш класс взаимодействует с остальным миром (что более важно для программы в целом, чем фактическая реализация класса). Кроме того, если вы сначала пишете интерфейс, вы можете увидеть, как класс будет работать с другими классами, прежде чем вы на самом деле погрузитесь в детали реализации.

Объекты при передаче параметров и возврате из методов.

Параметры методов могут быть переданы по значению, по ссылке или как выходные параметры. Когда параметр передается по значению, метод получает копию данных вызывающего объекта и не может изменить эту копию. При передаче параметра по ссылке метод получает указатель на данные вызывающего объекта. Эти данные используются совместно с вызывающим объектом. Если метод производит изменение значения параметра, передаваемого по ссылке, изменяются данные вызывающего объекта. При использовании параметра, передаваемого по ссылке, метод может использовать начальное значение данных. Выходные параметры похожи на параметры, передаваемые по ссылке, за исключением того что выходные параметры используются только для возврата данных вызывающему объекту, в то время как параметры, передаваемые по ссылке, могут использоваться как для передачи данных методу, так и для получения данных из метода.

Варианты реализации отношения клиент-сервер. Внутренние классы.

Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты, потребители этих функций.

Практические реализации такой архитектуры называются клиент-серверными технологиями. Каждая технология определяет собственные или использует имеющиеся правила взаимодейстия между клиентом и сервером, которые называются протоколом обмена (протоколом взаимодействия).

Принцип замещения Лисковой.

В объектно-ориентированном программировании принцип замены Лискова (LSP) является особым определением отношения подпечатания, названного (сильным) поведенческим подпечатанием, которое было первоначально введено Барбарой Лисковой в программной речи конференции 1987 года под названием абстракция Данных и иерархия. Это - семантическое, а не просто синтаксическое отношение, потому что это намеревается гарантировать семантическую способность к взаимодействию типов в иерархии, типов объекта в частности. Лисков и Джиннетт Венг сформулировали принцип кратко в газете 1994 года следующим образом:

Позвольте Неудавшийся, чтобы разобрать (Отсутствующий texvc выполнимый).

Одиночное наследование: понятие производного класса, управление доступом в производных классах, конструкторы и деструкторы, абстрактные классы и виртуальные функции, виртуальный полиморфизм, информация о типе на этапе выполнения RTTI.

Моделируя словарь системы, вам часто придется работать с классами, похожими на другие по структуре и поведению. В принципе их можно моделировать как различные, независимые друг от друга абстракции. Но лучше выделить одинаковые свойства и сформировать на их основе общие классы, которым наследуют специализированные.

Моделирование отношений наследования осуществляется в таком порядке:

Найдите атрибуты, операции и обязанности, общие для двух или более классов из данной совокупности

Вынесите эти элементы в некоторый общий класс (если надо, создайте новый, но следите, чтобы уровней не оказалось слишком много).

Отметьте в модели, что более специализированные классы наследуют более общим, включив отношение обобщения, направленное от каждого потомка к его родителю.

Здесь показано отношение обобщения, которое от четырех классов - РасчетныйСчет, Акция, Облигация и Собственность - направлено к более общему классу ЦенныеБумаги. Он является родителем, а остальные -его потомками. Каждый специализированный класс - это частный случай класса ЦенныеБумаги. Обратите внимание, что в классе ЦенныеБумаги есть две операции - presentValue (текущаяСтоимость) и history (история).

3 Множественное наследование.

Если порожденный класс наследует элементы одного базового класса, то такое наследование называется одиночным. Однако, возможно и множественное наследование. Множественное наследование позволяет порожденному классу наследовать элементы более, чем от одного базового класса. Синтаксис заголовков классов расширяется так, чтобы разрешить создание списка базовых классов и обозначения их уровня доступа:

```
class X
{...};
class Y
{...};
class Z
{...};
class A : public X, public Y, public Z
{...};
```

4 Пространства имен.

Пространство имен — это декларативная область, в рамках которой определяются различные идентификаторы (имена типов, функций, переменных, и т. д.).Пространства имен используются для организации кода в виде логических групп и с целью избежания конфликтов имен, которые могут возникнуть, особенно в таких случаях, когда база кода включает несколько библиотек.Все идентификаторы в пределах пространства имен доступны друг другу без уточнения.Идентификаторы за пределами пространства имен могут получить доступ к членам, используя полное имя идентификатора, например std::vector<std::string> vec;, используя Объявление using для отдельного идентификатора (using std::string) или Директива using (C++) для всех идентификаторов в пространстве имен (using namespace std;).Код в файлах заголовков всегда должен содержать полное имя в пространстве имен.

5 Обработка исключений.

Обработка исключений (exception handling) позволяет упорядочить обработку ошибок времени исполнения. Используя обработку исключений C++, программа может автоматически вызвать функцию-обработчик ошибок тогда, когда такая ошибка возникает. Принципиальным достоин-ством обработки исключений служит то, что она позволяет автоматизировать большую часть кода для обработки ошибок, для чего раньше требовалось ручное кодирование.

1.5. Лекция № 10 (2 часа)

Тема: «Природа классов».

1.5.1. Вопросы лекции:

1 Интерфейс и реализация

2 Отношение между классами
3 Ассоциация
4 Наследование. Множественное наследование

1.5.2. Краткое содержание вопросов:

1 Интерфейс и реализация

нтерфейсы COM описаны на языке IDL (Interface Definition Language – язык определений интерфейса) и имеют логические имена, которые указывают на моделируемые ими функциональные возможности. Ниже приведено IDL-определение COM-интерфейса IApe:

```
[object, uuid(753A8A7C-A7FF-11d0-8C30-0080C73925BA)]

interface IApe : Unknown

{

import «unknwn.idl»;

HRESULT EatBanana(void);

HRESULT SwingFromTree(void);
[propget] HRESULT Weight([out, retval] long *plbs);

}
```

Сопровождающая IApe документация должна специфицировать примерную семантику трех операций: EatBanana, SwingFromTree и Weight. Все объекты, раскрывающие IApe посредством QueryInterface , должны гарантировать, что их реализации этих методов придерживаются семантического контракта IApe. В то же время определения интерфейса почти всегда специально оставляют место для интерпретации разработчиком объекта. Это означает, что клиенты никогда не могут быть полностью уверены в точном поведении любого заданного метода, а только в том, что его поведение будет следовать схематическим правилам, описанным в документации к интерфейсу. Эта контролируемая степень неопределенности является фундаментальной характеристикой полиморфизма и одной из основ развития объектно-ориентированного программного обеспечения.

2 Отношение между классами

Класс в ООП может включать в себя другой класс, если он определяет атрибуты, являющиеся объектами другого класса. Такое отношение соответствует отношению включения между объектами. Например, класс автомобилей определяет, что у автомобиля есть атрибут-мотор, принадлежащий классу «Моторы». В свою очередь класс «Моторы» задает один из атрибутов-карбюраторов, являющимся экземпляром класса «Карбюраторы».

Отношение включение может использовать ссылку на другой класс, т.е. использовать вместо объекта указатель ссылку на объект в языке C++:Engine* carEngine;

Суть включения не изменяется, все равно у автомобиля имеется мотор. И в том и в другом случае класс-автомобиль включает атрибут-мотор. С учетом объема информации, хранящейся в объекте класса, положение не изменяется — информация хранящаяся в классе, включает информацию включаемого класса. Также не изменяется доступность

информации — и в том и в другом случае доступность информации определяется интерфейсом включаемого класса.

3 Ассоциация

Отношение ассоциации устанавливает двустороннюю связь между объектами разных классов. Предположим, что существует класс, описывающий преподавателей в университете, и класс, описывающий студентов. Студент может выбрать руководителя своего курсового проекта среди преподавателей. Между студентами и преподавателями устанавливается ассоциация «руководит — имеет руководителя курсового проекта», иными словами, преподаватель может руководить несколькими студентами а у студента есть руководитель. Если преподаватель отказывается от руководства каким-либо студентом (например, из-за хронического невыполнения заданий), он разрывает ассоциацию. При этом происходит следующее. Во-первых, данный студент удаляется из списка студентов, которыми руководит преподаватель (изменяется атрибут объекта преподаватель), а во-вторых, студент лишается преподавателя (изменяется атрибут объекта студент).

4 Наследование. Множественное наследование

Наследования как способ задания интерфейса. Механизм абстрактных классов и полиморфизма в ООП позволяет решать многие задачи в терминах базовых классов, не заботясь о том, какие конкретно классы участвуют в той или иной операции. Представим себе класс фигур, которые можно изображать на экране терминала в графическом редакторе. Из этого класса выделены конкретные классы квадрата, эллипса, треугольника, трехмерного шара и т.д. Базовый класс задает действия, которые можно производить с фигурами: сдвинуть, повернуть, масштабировать, закрасить и т.д. Большинство действий графического редактора можно запрограммировать в терминах действий на базовым классом (Например, при нажатии определенной клавиши сдвинуть фигуру к левому краю страницы). Полиморфизм в ООП обеспечит правильное выполнение конкретных действий, а добавление новых типов фигур путем введения нового конкретного класса из класса «Фигура» не нарушит алгоритмов работы редактора.

Приведем фрагмент кода программы на языке C++, который иллюстрирует использование базового класса «Фигура» для задания интерфейса. Ключевое слово class отмечает начало определения класса. В части, отмеченной меткой public, задается внешняя часть класса или его интерфейс. Ключевое слово virtual перед методом определяет то, что этот метод полиморфен (виртуален в терминах C++), т.е. может быть переопределен в порожденных классах. Все методы возвращают логический результат (bool), сообщающий об успешном или неуспешном завершении операции.

2. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ПРОВЕДЕНИЮ ПРАКТИЧЕСКИХ (СЕМИНАРСКИХ) ЗАНЯТИЙ

2.1. Практическое занятие №1 (2 часа).

Тема: «Эволюция методологий программирования».

2.1.1. Вопросы к занятию:

1. прецедент в объектном моделировании (этапы).
2. структура диаграммы.

2.1.2. Краткое описание проводимого занятия:

2.1.2.1. Ответы на вопросы семинарского (практического) занятия.

2.1.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Алгоритмом называют

+1) набор предписаний над исходными и промежуточными данными для получения конечного результата;

2) графическое изображение, где каждый этап процесса представлен в виде геометрической фигуры

3) последовательность команд, описывающая выполнение действий на понятном языке и приводящая к получению результата

4) систему обозначений, служащую для точного описания программ на ЭВМ

2. Указать наиболее полный перечень способов записи алгоритмов:

1) словесный, графический, псевдокод, программный

2) словесный

3) графический, программный

4) словесный, программный

5) псевдокод

3. Суть такого свойства алгоритма как результативность заключается в том, что:

1) алгоритм должен иметь дискретную структуру

2) записывая алгоритм для исполнителя, используют команды, входящие в систему

3) алгоритм должен обеспечивать решение некоторого класса задач данного типа;

4) при исполнении всех команд процесс должен прекратиться за конечное число шагов

5) исполнитель алгоритма не должен принимать решения, не предусмотренные составителем алгоритма

4. Блок-схема – это:

1) ориентированная сеть с вершинами типов: функциональные, предикатные и объединяющие

2) рисунок с изображением алгоритма

3) семантический граф операторов алгоритма

4) семантическая диаграмма

2.2. Практическое занятие №2 (2 часа).

Тема: «Понятие объекта».

2.2.1. Вопросы к занятию:

1 Свойства, присущие объектам: состояние, поведение, идентичность.

2.2.2. Краткое описание проводимого занятия:

2.2.2.1. Ответы на вопросы семинарского (практического) занятия.

2.2.2.2. Проведение текущего контроля успеваемости

Состояние объекта характеризуется перечнем (обычно статическим) всех свойств данного объекта и текущими (обычно динамическими) значениями каждого из этих свойств.

К числу свойств объекта относятся присущие ему или приобретаемые им характеристики, черты, качества или способности, делающие данный объект самим собой. Например, для лифта характерным является то, что он сконструирован для поездок вверх и вниз, а не горизонтально.

Все свойства имеют некоторые значения. Эти значения могут быть простыми количественными характеристиками, а могут ссылаться на другой объект. Состояние лифта может описываться числом 3, означающим номер этажа, на котором лифт в данный момент находится.

Простые количественные характеристики (например, число 3) являются «постоянными, неизменными и непреходящими», тогда как объекты «существуют во времени, изменяются, имеют внутреннее состояние, преходящи и могут создаваться, уничтожаться и разделяться».

Тот факт, что всякий объект имеет состояние, означает, что всякий объект занимает определенное пространство (физически или в памяти компьютера).

Все объекты в системе инкапсулируют некоторое состояние, и все состояние системы инкапсулировано в объекты.

Поведение

Поведение — это то, как объект действует иреагирует; поведение выражается в терминах состояния объекта и передачи сообщений.

Иными словами, поведение объекта — это его наблюдаемая и проверяемая извне деятельность.

Операцией называется определенное воздействие одного объекта на другой с целью вызвать соответствующую реакцию. В чисто объектно-ориентированном языке, таком как Smalltalk, принято говорить о передаче сообщений между объектами. В языках типа C++ мы говорим, что один объект вызывает функцию-член другого. В основном понятие сообщение совпадает с понятием операции над объектами, хотя механизм передачи различен. В объектно-ориентированных языках операции, выполняемые над данным объектом, называются методами и входят в определение класса объекта. В C++ они называются функциями-членами.

Состояние объекта представляет суммарный результат его поведения.

Наиболее интересны те объекты, состояние которых не статично: их состояние изменяется и запрашивается операциями.

Идентичность

«Идентичность — это такое свойство объекта, которое отличает его от всех других объектов».

Они отмечают, что «в большинстве языков программирования и управления базами данных для различения временных объектов их именуют, тем самым путая адресуемость и идентичность. Большинство баз данных различают постоянные объекты по ключевому атрибуту, тем самым, смешивая идентичность и значение данных». Источником множества ошибок в объектно-ориентированном программировании является неумение отличать имя объекта от самого объекта.

2.3. Практическое занятие № 3 (2 часа).

Тема: «Разработка Visual Basic-приложений**».**

2.3.1. Вопросы к занятию:

1. заполнение списка и удаление его элементов
2. список с несколькими столбцами
3. многоэлементный выбор из списка
4. добавление и удаление данных

2.3.2. Краткое описание проводимого занятия:

2.3.2.1. Ответы на вопросы семинарского (практического) занятия.

2.3.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Оператором выбора (переключателем) в языке VBA является:
   1) Select Case;
   2) If;
   3) Case;
   4) IIF.
2. Процедура ShowMessage служит для отображения в Delphi:

1) окна сообщений;
2) окна помощи;
3) простого окна;
4) системного окна;
5) окна с картинкой.

3. Простейшее диалоговое окно в Delphi, отображаемое ShowMessage, содержит кнопку:
1) OK, Cancel;
2) OK;
3) OK, YES;
4) OK, NO;
5) OK, Help.

4. Какая функция в Delphi служит для отображения диалогового окна для ввода данных пользователем:
1) Showmessage;
2) MessageDlg;
3) Show;
4) InputBox();
5) Input.

2.4. Практическое занятие № 4 (2 часа).

Тема:«Сведения о классах и наследовании. Основы визуального программирования интерфейса**».**

2.4.1. Вопросы к занятию:

1. создание двоичного файла
2. чтение файла
3. использование диалоговых окон сохранения и чтения файла

2.4.2. Краткое описание проводимого занятия:

2.4.2.1. Ответы на вопросы семинарского (практического) занятия.

2.4.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Тестирование программы – это:
1) оценивание ресурсов компьютера, на котором будет работать программа
2) перевод проекта в форму программы для конкретного компьютера
3) системный подход к построению алгоритма с использованием декомпозиции и синтеза
4) процесс исполнения программы с целью выявления ошибок

2. Оператор – это:
1) функция, которая оперирует с данными
2) законченная фраза языка, предписание, команда
3) алгоритм действия программы, написанной на данном языке
4) процедура обработки данных

3. Переменная – это:
1) объект, способный принимать различные значения
2) значения чисел
3) меняющееся число
4) динамический объект

4. Модуль – это:
1) отдельная программа, которая взаимодействует с другими программами
2) набор символов и идентификаторов
3) специальная программная единица для создания библиотек
4) вспомогательная процедура

2.5. Практическое занятие № 5 (2 часа).

Тема: «Природа классов**».**

2.5.1. Вопросы к занятию:

1. TControl – базовый класс визуальных компонентов
2. Окно редактирования Edit

2.5.2. Краткое описание проводимого занятия:

2.5.2.1. Ответы на вопросы семинарского (практического) занятия.

2.5.2.2. Проведение текущего контроля успеваемости

Задания для проведения текущего контроля успеваемости

1. Индуктивный принцип – это:

1) когда определяется связь между входными, выходными данными и процессами обработки

2) принцип построения модели от частного к общему

3) упрятывание информации и абстрактных типов данных

4) принцип построения модели от общего к частному

2. Линейный связный список – это:

1) конечный набор пар, состоящих из информативных и указующих частей

2) рекурсивная конструкция алгоритма

3) совокупность динамических переменных

4) массив указателей

3. Сортировка – это:

1) процесс нахождения в заданном множестве объекта

2) процесс перегруппировки заданного множества объектов в некотором порядке

3) установка индексов элементов в возрастающем порядке

4) обработка элементов в алфавитном порядке

4. Композиция – это:

1) циклическая конструкция алгоритма, состоящая из многократного повторения одного блока действий

2) линейная конструкция алгоритма, состоящая из последовательно следующих друг за другом функциональных вершин

3) конструкция ветвления, имеющая предикатную вершину

4) механизм языка, позволяющий описать новый класс на основе существующего (родительского) класса