Оглавление

Введение 2

1. Система программирования Delphi 3

1.1 Состав проекта 3

1.2 Описание класса 4

2. Класс VCL 10

2.1 Компоненты 11

2.2 Основы создания компонентов 12

Заключение 15

Приложение А. Первый шаг создания нового компонента 17

Приложение Б. Логическая структура сети университета 18

Приложение В. Система ДОТ «Platonus» 19

Введение

Цель производственной практики: закрепление теоретических знаний, полученных в университете, развитие необходимых практических умений и навыков в соответствии с темой дипломной работы «Создание нового компонента в визуальной среде разработки приложений».

Для решения поставленной цели были определены следующие задачи:

Углубленное изучение систему программирования Delphi;

изучение и сбор необходимых материалов для выполнения дипломной работы;

приобретение практических навыков в работе эксплуатации локальной вычислительной сети тьюторской службы университета

выполнение работы по обслуживанию компьютерной техники и специальных программ.

Дата прохождения практики: с 25.10.2010 по 20.11.2010 г.

Место прохождения практики: отдел инженерной службы Казахстанско-Российского университета.

1. Система программирования Delphi

1.1 Состав проекта

Delphi – это система визуального объектно-ориентированного программирования, в которой в качестве языка программирования используется язык Object Pascal. Язык Object Pascal – строго типизированный объектно-ориентированный язык, в основе которого лежит Turbo Pascal.

В состав Delphi входит библиотека VCL (Visual Component Library), содержащая компоненты для построения интерфейса приложения, а также структуры для хранения и обработки данных. Библиотека представляет собой иерархию объектов, имеющую общего предка предка – TObject.

Проектируемое с помощью Delphi приложение хранится в нескольких файлах. Рассмотрим основные файлы:

.dpr – файл проекта. Это главная программа, в которой содержится код инициализации, завершения и запускается цикл обработки событий. Файл содержит ссылки на все формы и модули, входящие в состав проекта.

.pas – исходный код модуля и формы. Большинство модулей связаны с формами и содержат тексты обработчиков событий элементов, размещенных на формах.

.dfm – файл формы. Содержит описание внешнего вида формы и расположенных на ней элементов.

.cfg,.dsk,.dof – файлы настроек компилятора, среды и проекта.

.res – файл ресурсов (создается автоматически). Включает значок программы, номер версии и т.д.

.exe – исполняемый файл созданного приложения.

.~dpr,.~pas,.~dfm – резервные файлы проекта, модуля и формы. Создаются, если включен режим резервного копирования.

1.2 Описание класса

Объект, называемый в Delphi классом, может кроме разделов private и public иметь разделы protected, published и automated.

Элементы класса, расположенные после ключевого слова protected, называются защищенными. Они доступны внутри модуля, содержащего описание класса, и внутри его потомков.

В разделе published содержатся элементы, отображаемые в инспекторе объектов. Для них установлен тот же тип доступа, что и для элементов public. Этот раздел используется только при разработке нестандартных компонентов. Раздел automated содержит элементы, которые добавляются к интерфейсу объектов, использующих OLE. Интерфейс OLE (Object Linking and Embedding — связывание и внедрение объектов) поддерживается множеством различных программ и используется для помещения документа, созданного в одной программе, в другую программу. Например, можно вставить документ Microsoft Word в Microsoft Excel. Таким образом, OLE - это технология связи программ, позволяющая совместно использовать данные.

В потомках объекта доступность унаследованных элементов можно изменять.

Основная особенность объектов в Delphi состоит в том, что они по умолчанию являются динамическими, но операция разыменовывания для доступа к ним не используется. Если предок класса не указан, им считается TObject.

Рекомендуется для конструктора класса использовать имя Create, для деструктора – Destroy и вызывать из них соответствующие методы предка с помощью ключевого слова inherited. Вызов конструктора предка выполняется перед первым исполняемым оператором конструктора класса, а деструктора – после последнего. Деструктор объявлен в TObject как виртуальный, поэтому в любом потомке требуется объявлять его с ключевым словом override, которое используется в Delphi для переопределения виртуальных методов.

Рассмотрим пример, в котором объявляется класс «целое число».

type

TNumber = class

n:integer;

constructor Create(n\_:integer);

end;

constructor TNumber.Create(n\_:integer);

begin

inherited Create;

n:=n\_;

end;

Переменная класса TNumber создается и удаляется так:

var

num:TNumber;

…

num:=TNumber.Create(10);

Label1.Caption:=IntToStr(num.n);

num.Destroy;

Здесь описан указатель на класс TNumber. Конструктор Create выделяет место в динамической памяти, достаточное для размещения объекта, и инициализирует его. В предпоследней строке значение поля num преобразуется в строку и выводится на форму через метку (элемент управления типа TLabel). Обратите внимание на то, что операция разыменовывания для доступа к объекту не используется.

Если конструктор или деструктор в классе задан, оба они наследуются от предка, а в конечном счете – от TObject, в соответствующих методах которого задается размещение объекта в памяти и ее освобождение.

Если требуется переопределить виртуальный метод, он описывается с ключевым словом override, а не virtual. Если же указать в потомке слово virtual, это будет воспринято компилятором как начало новой иерархии виртуальных методов.

Конструктор в Delphi может быть виртуальным, поскольку он размещает объект в динамической памяти только при первом вызове, а во время всех последующих вызовов выполняется как обычный метод.

В классах, предназначенных для создания не объектов, а потомков, разрешается определять методы с ключевым словом abstract. Для таких методов реализация задается только в потомках класса. Класс, содержащий хотя бы один абстрактный метод, называется абстрактным. Создавать экземпляры объектов такого класса запрещается, а в потомках, предназначенных для создания объектов, все абстрактные методы надо обязательно переопределить. Абстрактные методы предназначены для задания общего интерфейса иерархии.

При переопределении обычных методов в производном классе непосредственный доступ к одноименному методу предка невозможен. Чтобы обойти это ограничение, при описании метода потомка можно задать ключевое слово overload (перегруженный). У перегруженных методов должен различаться список параметров, поскольку именно по типам аргументов компилятор выбирает, какой из одноименных методов вызвать. Перегруженные методы не могут объявляться в одном и том же классе – только в предке и потомке, например:

type

a = class

procedure max(a,b:integer):overload;

end;

b = class(a)

procedure max(a:real; b:integer; s:char):overload;

end;

…

var

x:b;

…

x.max(20,3); {вызывается метод из класса a}

x.max(0.1,10,’u’); {вызывается метод из класса b}

Виртуальные методы также можно перегружать, но при этом становится невозможным доступ к методу предка и компилятор выдает предупреждение. Чтобы подтвердить компилятору свои намерения, можно использовать ключевое слово reintroduce. Рассмотрим варианты переопределения виртуальных методов на примере:

type

a = class

procedure p1; virtual;

procedure p2; virtual;

procedure p3; virtual;

procedure p4; virtual;

procedure p5; virtual;

end;

b = class(a)

procedure p1;

procedure p1; reintroduce;

procedure p1; override;

procedure p1; virtual;

procedure p1; reintroduce; virtual;

end;

…

var

x:a;

…

x:=b.create;

x.p1; x.p2; x.p3; x.p4; x.p5;

В производном классе b процедуре р1 переопределяет виртуальный метод на простой (выдается предупреждение о том, что доступ к соответствующему методу предка закрыт), процедура р2 делает то же самое, но предупреждение не выдается, процедура р3 переопределяет виртуальный метод традиционным способом (для обеспечения полиморфизма), р4 начинает новое семейство виртуальных методов (доступ к методу предка закрывается, выдается предупреждение), и, наконец, процедура р5 делает то же самое, что и р4, но предупреждение не выдается.

При вызове этих методов для объекта производного класса через указатель на базовый класс метод потомка вызывается только в случае x.p3. Свойства (доступ к которым обеспечивается через вкладку properties инспектора объектов) служат для организации доступа к полям класса. Как правило, свойство связано с полем класса и определяет методы его получения и установки. Упрощенно определение свойства выглядит так:

property имя: тип

[read имя\_метода\_или\_поля] {метод получения}

[write имя\_метода\_или\_поля] {метод устанвки}

Например:

type

a = class

private FSize: word;

function GetSize:word;

procedure SetSize(x\_:word);

property Size:word read GetSize write SetSize;

end;

Здесь свойство Size определяет интерфейс доступа к полю FSize. Если установка или получение значения поля запрещены, соответствующие части определения опускают. Метод записи обычно содержит действия по проверке допустимости устанавливаемого значения, метод чтения может содержать, например, поддержку счетчика обращений к полю. Если никаких дополнительных действий не требуется, после read или write вместо имени метода может указываться просто имя поля.

В программе свойство выглядит как поле класса, например:

var

x:a;

…

x:=100;

При обращении к свойству автоматически вызываются указанные в нем методы чтения и установки.

Рассмотренные свойства являются простыми. Кроме простых можно определять процедурные свойства, свойства-массивы и индексируемые свойства.

2. Класс VCL

Использование стандартных объектов при создании приложений не только уменьшает сроки разработки, но и снижает количество ошибок. Отрицательным же следствием их применения является увеличение размера программы. Библиотека VCL содержит несколько сотен классов, образующих древовидную иерархию. Рассмотрим фрагмент такой иерархии: TObject – базовый класс иерархии. В нем нет полей, но заданы методы, определяющие основу поведения любого объекта, – создание, уничтожение объекта, обработку сообщений. Все классы, как стандартные, так и создаваемые программистом, прямо или косвенно наследуются от TObject. Их можно разделить на визуальные и невизуаьные. Визуальные классы предназначены для организации взаимодействия с пользователем, невизуальные – для хранения и обработки данных.

Exception – является базовым классом для обработки исключительных ситуаций, которые возникают, например, при попытке деления на нуль, выходе за пределы выделенной области памяти, обращении к не существующему файлу и т.д. В Object Pascal реализован механизм обработки исключений, а VCL включает почти сотню классов для представления всех исключительных ситуаций.

TPersistent – является предком всех классов, которые могут иметь раздел published (элементы, заданные в этом разделе, отображаются в инспекторе объектов). Опубликованные свойства объектов сохраняются при записи на диск.

TList – предназначен для хранения произвольного количества элементов разных типов и доступа к ним по индексу аналогично массиву. Для хранения данных предназначены и другие потомки TPersistent, например классы TStack, TCollection, TStrings и их потомки. Для сохранения данных и объектов на внешних носителях предназначен класс TStream и его производные.

TComponent – является предком визуальных и невизуальных компонентов (таких, как объект-приложение TApplication). Потомки TComponent могут отображаться в палитре компонентов, быть владельцами других компонентов и помещаться на форму.

TControl – задает основные свойства всех визуальных компонентов. От него наследуются так называемые элементы управления – кнопки, метки, списки, флажки. Через эти элементы осуществляется диалог с пользователем. Те элементы, которые могут получить фокус ввода (воспринимают ввод с клавиатуры), наследуются от TControl через TWinControl

2.1 Компоненты

Компоненты Delphi являются частью классовой иерархии Delphi, которая называется библиотекой визуальных компонентов (Visual Component Library, VCL). Библиотека визуальных компонентов Delphi, несмотря на название, содержит в своем составе как визуальные (видимые во время выполнения приложения), так и невизуальные (видимые только на этапе проектирования) компоненты. К визуальным компонентам относятся TButton, TEdit, TLabel и др. К невизуальным относятся такие компоненты, как TTimer, TTable, TQuery И др.

Компонент - это небольшая часть приложения, обеспечивающая создание пользовательского интерфейса. Компоненты можно представить как небольшие "кирпичики", из которых строится "дом" - приложение.

Компоненты могут находиться на панели компонентов. Разработчик может брать компоненты из панели компонентов и располагать их на форме.

Компоненты могут сильно отличаться по степени своей сложности. Можно использовать в своем приложении как простые компоненты (как, например, TLabel, TEdit), так и достаточно сложные.

Типы компонентов

В Delphi имеется четыре базовых типа компонентов.

Первый тип компонентов - стандартный.

Стандартные компоненты включают в себя поведение элементов управления Windows. К таким компонентам относятся: TListView, TRichEdit, TImageList и др. Все эти компоненты находятся на вкладке Win32 панели компонентов Delphi. Каждый из данных компонентов представляет собой так называемую оболочку (wrapper) стандартного элемента управления WindowsВторой тип компонентов - пользовательский.

К пользовательским компонентам относятся все компоненты, которые невходят в стандартную библиотеку компонентов Delphi

Свойства

Свойства компонента предназначены для доступа пользователя к внутренним полям компонента. Свойства позволяют изменять атрибуты компонента, в том числе и вычисляемыми значениями. Определение свойства компонента должно содержать его имя и тип, а также как минимум одно объявление способа доступа к данному свойству (описания). Синтаксис определения свойства компонента имеет вид:

property Имя свойства: тип описания;

- имя свойства - должно быть уникальным;

- тип - определяет, значения какого типа могут быть записаны в данном свойстве;

- описания - объявления способов доступа к этому свойству.

2.2 Основы создания компонентов

Опишем создание компонента. Для создания собственного компонента важно иметь представление о библиотеке визуальных компонентов Delphi, об иерархии компонентов.

Решение о создании новых компонентов может быть принято по ряду причин, среди которых:

- разработка нового пользовательского интерфейса, с дальнейшим использованием его в других приложениях;

- создание принципиально нового класса, которого нет в стандартной библиотеке Delphi и среди элементов ActiveX;

- упрощение кода приложения, путем введения новых компонентов; - распространение своих компонентов среди других программистов;

- и, наконец, желание глубоко изучить Delphi, разобраться с тонкостями программирования.

Описание процесса создания:

- определить действия компонента;

- разработать краткий алгоритм, по которому будет работать компонент;

- разбить всю конструкцию компонента на независимые части;

- возможность дальнейшей разработки;

- код компонента, этот пункт разбивается на такие этапы:

выбор предка для вашего компонента;

создание заготовки (модуля) компонента;

создание свойств, событий и методов компонента;

отладка и тестирование;

- регистрация компонента в среде Delphi;

- создание справки для вашего компонента.

Рис. 1. Применение свойства Align

Рис. 2 Разбиение меню приложения на столбцы разделов с помощью свойства Break

Обладая, мастерством рисования, и зная методику написания визуальных компонентов для delphi написать элемент интерфейса приложений для Windows.

Заключение

Во время прохождения практики были преодолены все задачи, предусмотренные программой практики:

углубил знания по программированию, технологии программирования;

подготовил структуру ВКР;

приобрел опыт работы по обслуживанию и эксплуатации оргтехники и периферийных устройств;

приобрел практические навыки в работе эксплуатации локальной вычислительной сети тьюторской службы университета;

сопровождал программу «Platonus» для отдела дистанционного обучения по стандарту РК;

Считаю что выполненные мною работы и практические задания при прохождении практики, помогли мне закрепить навыки, умения и знания, полученные за время учебы в университете.

Литература

1. Ахаян В., Горев А., Макашарипов С. Эффективная работа с СУБД. С-П.: ПИТЕР, 2002 г.
2. Ворощук А. Н. Основы ЦВМ и программирование. Главная редакция физико-математической литературы изд-ва “Наука”, М., 2003 г.
3. Вычислительные машины, системы и сети: Учебник/А.П. Пятибратов, С.Н. Беляев, Г.М. Козырева и др.; Под ред. проф. А.П. Пятибратова. - М.: Финансы и статистика, 2001 г.
4. Горев А., Макашарипов С., Владимиров Ю. Microsoft SQL Server для профессионалов. СПб.: Питер, 2003 г.
5. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов. М., 2001 г.
6. Дарахвелидзе П., Марков Е. Программирование в Delphi. Спб, 2005 г.
7. Дейт К. Введение в системы баз данных. М: Наука, 2002 г.
8. Каган Б. М. Электронные вычислительные машины и системы: Учеб. пособие для вузов. - 2-е изд., перераб. и доп. М.: Энергоатомиздат, 2005 г.
9. Макдона Р. Основы микрокомпьютерных вычислений: Пер. с англ./ Т. Г. Никольской; Под ред. В.Ф. Шаньгина. М.: Высш. шк., 2002 г.
10. Мейер Б. Бодуэн К. Методы программирования. М., 2002 г.
11. Ахо А.В., Хопкрофт Д.Э., Ульман Д.Д. Структуры данных и алгоритмы. – М.: Издательский дом “Вильямс”, 2000.
12. Кнут В. Искусство программирования. Т.1. Основные алгоритмы. – 3-е изд. - Издательский дом “Вильямс”, 2000.

Приложение А

Первый шаг создания нового компонента

Рис.1 Окно NewItems

Рис.2 Эксперт для создания нового компонента

Приложение Б

Логическая структура сети университета

Приложение В

Система ДОТ «Platonus»

