Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Оренбургский государственный университет»

Колледж электроники и бизнеса

Н.А. Уйманова, М.Г. Таспаева

ОСНОВЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

Часть II

Рекомендовано к изданию Редакционно-издательским советом федерального государственного бюджетного образовательного учреждения высшего профессионального образования «Оренбургский государственный университет» в качестве методических указаний для студентов, обучающихся по программам среднего профессионального образования по специальности 09.02.03 Программирование в компьютерных системах

Оренбург
2015

УДК 681.3.06:004.4(075.3)
ББК 32.973 Я73
У 35

Рецензент – кандидат педагогических наук, доцент А.Е.Шухман

Уйманова, Н.А.

У 35 Основы объектно-ориентированного программирования : методические указания к лабораторным работам: в 3 ч. Часть 2 / Н.А. Уйманова, М.Г. Таспаева; Оренбургский гос. ун-т. – Оренбург : ОГУ, 2015. – 34 с.

Методические указания предназначены для выполнения лабораторных работ, обеспечивающих учебный процесс по дисциплине «Основы объектно-ориентированного программирования», студентами 3 курса очной формы обучения специальности 09.02.03 «Программирование в компьютерных системах».

Методические указания составлены с учетом федерального государственного образовательного стандарта среднего профессионального образования по направлению подготовки дипломированных специалистов, утвержденного приказом № 804 от 28 июля 2014 года Министерством образования и науки Российской Федерации.

УДК 681.3.06:004.4(075.3)
ББК 32.973 Я73

Содержание

Введение

Учебная дисциплина «Основы объектно-ориентированного программирования» является дисциплиной из вариативной части ОПОП, обуславливающей знания для профессиональной деятельности выпускника.

Целью данного курса является формирование у будущего специалиста умений и навыков профессиональной направленности, написания программных продуктов, базирующихся на принципах объектно-ориентированного программирования, а так же сформировать представление о работе в среде визуального редактора.

У студентов необходимо сформировать такие умения и навыки, чтобы они могли в дальнейшем эффективно их использовать в своей профессиональной деятельности при программировании программных продуктов. Будущий специалист должен овладеть, прежде всего, базовыми принципами написания объектно-ориентированной программы, уметь работать в среде визуального программирования Delphi.

Задачами курса является:

- изучение основных понятий объектно-ориентированного программирования, их программное описание;
- изучение принципов объектно-ориентированного программирования;
- практическое освоение структуры написания классов с целью дальнейшего применения в профессиональной деятельности;
- изучение приемов организации распределения памяти для экземпляров класса;
- изучение технологии описания и применения виртуальных методов;
- выработка умений написания программных продуктов в среде визуального программирования Delphi.

Методические указания к лабораторным работам (часть 2) предназначены для проведения лабораторных занятий по дисциплине «Основы объектно-ориентированного программирования» по темам «Понятие виртуального метода» и «Динамическое распределение памяти».

1 Лабораторная работа №4. Динамическое распределение памяти

1.1 Цель работы

Научиться распределять память для экземпляров объекта, динамически создавать компоненты.

1.2 Ход работы

Задание общего уровня:

1 Изучить методические рекомендации к лабораторной работе;

2 Создать новый проект delphi, осуществляющий занесение данных об автопарке организации в список;

3 Спроектировать форму приложения, согласно рисунка 1, представленного в методических рекомендациях;

4 Описать родительский класс *tts* в секции *interface* программного модуля ниже описания класса *TForm1*;

5 В классе описать поля *fname* типа *string*, *fmosh_dvig* типа *integer*;

6 Описать конструктор *create*, осуществляющий присвоение полям объекта начальных значений;

7 Описать деконструктор *destroy*, осуществляющий уничтожение объекта.

8 Описать дочерний класс *tbus*;

9 В классе описать поле *fcount_places* типа *integer*;

10 Описать конструктор *create*, осуществляющий присвоение полям объекта начального значения;

11 Описать деконструктор *destroy*, осуществляющий уничтожение объекта;

12 Описать метод show для вывода списка автобусов;

13 В секции реализации implementation описать реализацию всех методов созданных классов (см. методические рекомендации);

14 Объявить секцию const после описания всех классов, задать переменную *count*, определяющую максимальное количество автобусов в списке;

15 В глобальном разделе описания переменных объявить массив автобусов *bus* типа *tbus*;

16 Объявить переменную «счетчик» количества автобусов;

17 Для компонента *button1* организовать событие *onclick*, осуществляющее создание списка;

18 Для компонента *button2* организовать событие *onclick*, осуществляющее вывод списка в компонент *memo1*;

19 Обработать дополнительные методы проекта;

20 Сохранить проект на диске в директории lab4_1;

21 Провести отладку и компиляцию проекта;

22 Реализовать созданные методы, при закрытии проекта вызвать их на выполнение;

23 Сохранить проект, выполнить отладку и компиляцию;

24 Создать новый проект delphi, демонстрирующий динамическое создание объекта *tmemo* на форме (см. методические указания п. 1.5.3);

25 Провести отладку и компиляцию проекта;

26 Сохранить проект на диске в директории lab4_2;

27 Оформить отчет о проделанной работе;

Задание повышенного уровня:

28 Выполнить индивидуальное задание согласно выданного варианта;

29 В проекте описать собственные классы, организовать методы, описать и реализовать конструкторы и деструкторы для объектов описанных классов;

30 Сохранить проект на диске в директории lab4_3;

31 Выполнить отладку и компиляцию проекта;

32 Оформить отчет о проделанной работе;

33 Защитить работу.

1.3 Содержание отчета

1 Цель, ход работы;

2 Постановка задачи, листинг программного модуля, результат работы приложения;

3 Постановка задачи на создание динамического компонента delphi, листинг программного модуля, результат работы приложения до создания объекта и после;

4 Формулировка индивидуального задания;

5 Листинг программного модуля индивидуального задания, результат работы приложения;

6 Вывод.

1.4 Контрольные вопросы

1 Охарактеризуйте основные понятия объектно-ориентированного программирования;

2 Охарактеризуйте принципы объектно-ориентированного программирования (полиморфизм, инкапсуляция, наследование);

3 Привести примеры программного описания принципов объектно-ориентированного программирования;

4 Каковы функции использования конструкторов и деструкторов?

5 Правила объявления конструкторов и наследуемых конструкторов;

6 Для чего используется директива inherited?

7 Какова функция указателя self?

1.5 Методические рекомендации

1.5.1 Понятие конструктора и деструктора

Методы, которые предназначены для создания и удаления объектов называются конструкторами и деструкторами соответственно. Описание данных методов отличается от обычных тем, что в их заголовках стоят ключевые слова *constructor* и *destructor*. В качестве имен конструкторов и деструкторов в базовом классе *TObject* и многих других классах используются имена *Create* и *Destroy*.

Конструкторы и деструкторы отвечают за существование объекта в памяти, т.е. выделяют память для экземпляра класса, затем и освобождают ее.

Конструктор – это специальный вид подпрограммы, присоединенный к классу. Его назначение – создавать представителей (экземпляры) класса. Он ведет себя как функция, которая возвращает ссылку на вновь созданный экземпляр класса, т.е. на объект. Одновременно выделяется память для хранения значений полей экземпляра класса.

Деструктор – это специальная разновидность подпрограммы, присоединенной к классу. Его назначение заключается в уничтожении экземпляра класса, т.е. объекта и освобождении памяти, выделенной под экземпляр.

Синтаксис объявления конструкторов и деструкторов:

Type

*<имя класса>=**Class**[{Имя родительского класса>)]*

Constructor *<Имя конструктора>[(<параметры>)]; [**Override**;]*

Destructor *<Имя деструктора>[(<параметры>)}; [**Override**;]*

***End**;*

П р и м е ч а н и я :

1 Объявляются конструкторы и деструкторы, как правило, в разделе *Public* класса;

2 В классе может быть объявлено несколько конструкторов, однако чаще бывает один конструктор. Общепринятое имя для единственного конструктора *Create*;

3 В одном классе может быть объявлено несколько деструкторов, но чаще бывает один деструктор без параметров с именем *Destroy*;

4 За объявлением деструктора по имени *Destroy* следует указывать ключевое слово-директиву *Override*, разрешающее выполнение предусмотренных по умолчанию действий для уничтожения экземпляра объекта, если при его создании возникла какая-либо ошибка. Фактически *Override* переопределяет метод предка;

5 Метод *Free* так же удаляет (разрушает) экземпляры класса, предварительно проверяя их на *Nil*.

Реализация конструктора

В задачу конструктора входит создание экземпляра класса и выполнение операторов, содержащихся в его теле. Назначение кода внутри конструктора – инициализировать только что созданный экземпляр объекта. Синтаксис реализации конструктора:

Constructor *<имя класса>.<имя конструктора>[(<параметры>}};*

[<блок объявлений>}

Begin

<Исполняемые операторы>

End*;*

Реализация наследуемых конструкторов.

Constructor *<имя класса>.<имя конструктора>[(<параметры>)];*

[<блок обьявлений>}

Begin

Inherited *<имя конструктора>[(<параметры>)};*

<инициализация собственных полей>

End*;*

Реализация деструкторов

Деструктор уничтожает экземпляр класса, который был использован при его

вызове, автоматически освобождая любую динамическую память, которая ранее была зарезервирована конструктором, закрывает файлы и т.п. операции. Программист ответственен за вызов деструкторов для всех экземпляров класса, если были зарезервированы подчиненные объекты. Часто деструктор не выполняет других действий и представляет собой пустую процедуру. Синтаксис реализации деструктора:

Destructor *<имя класса>.<имя деструктора>[(<параметры>)];*

[<блок объявлений>]

Begin

<исполняемые операторы>

End*;*

Реализация наследуемых деструкторов

Если использовать механизм наследования деструкторов, то можно упростить задачу уничтожения экземпляров класса, таким образом, чтобы каждый раз заботиться лишь об уничтожении тех полей, которые были добавлены в данном классе. Всю работу по очистке наследуемых полей можно возложить на наследуемые деструкторы. Для вызова наследуемого деструктора необходимо используется ключевое слово *Inherited*.

Синтаксис объявления наследуемого деструктора следующий:

Destructor *<имя класса>.<имя деструктора>[(<параметры>)};*

[<блок объявлений>]

Begin

<уничтожение собственных полей>

Inherited *<имя деструктора>[{<параметры>)];*

End*;*

Вызов конструкторов

Для того чтобы вызвать конструктор, необходимо правильно объявить и определить класс. Объявление класса должно быть доступно, т.е. он должен находится в области видимости из того места, где конструктор будет вызываться.

Если в пользовательском классе не определен конструктор, то по умолчанию

будет использоваться конструктор, унаследованный от класса-потомка. В любом случае у всех объектов есть доступ к конструктору *Create*, определенному в классе *TObject*.

Определение класса создает активную структуру, способную создавать представителей этого класса. Объекты, которые создаются с помощью определения класса, способны хранить ссылки на вновь создаваемые объекты.

Var *<имя обьекта>: <имя класса>; // Объявление переменной–указателя*

Begin

<имя обьекта>:=<имя класса>.<имя конструктора>[(<параметры>)];

Если вызвать конструктор от имени объекта, то новый объект не будет создан (память не выделяется), но будут выполнены операторы, указанные в коде конструктора. Конструктор может также вызываться с помощью переменной типа указателя на класс.

Вызов деструкторов

Деструкторы вызываются точно так же, как и большинство других методов класса – через его действующий экземпляр. Синтаксис вызова конструктора следующий:

<имя объекта>.<имя деструктора>[(<параметры>)];

Для примера опишем родительский класс транспортных средств *TTs*, дочерний класс *Автобусы* (*TBus*).

Интерфейс разрабатываемого приложения может иметь вид, представленный на рисунке 1.

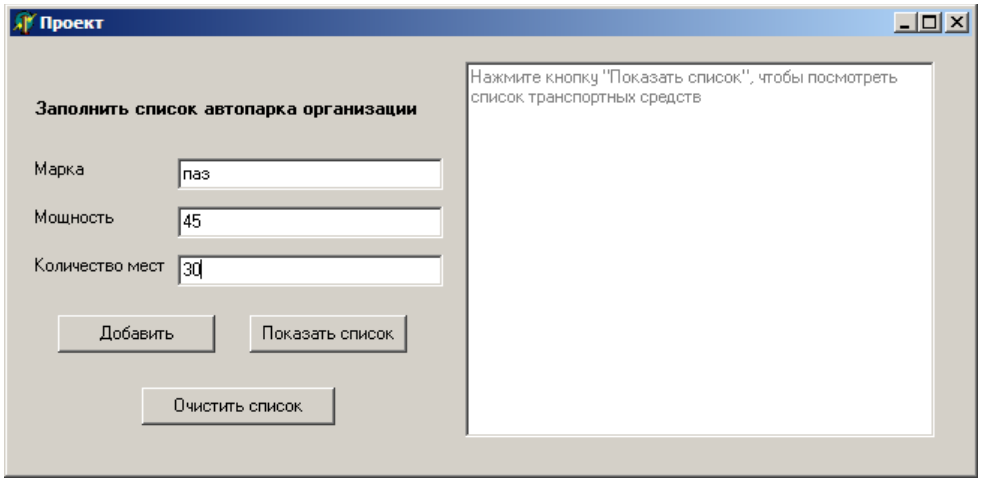

Рисунок 1– Интерфейс приложения

В разделе Type определим родительский класс *TTs*. Класс содержит поля *наименование* (*fname*) и *мощность двигателя (fmosh_dvig)*, конструктор *Create*.

```
type TTs = class
  fname:string[20];
  fmosh_dvig: integer;
  constructor Create (name:string; mosh_dvig:integer);
  destructor Destroy;
end;
```

Определим дочерний класс *TBus*. Класс содержит поля (*count_places*), конструктор *Create*, метод *Show* для вывода информации об автобусах в компонент *TMemo*.

```
TBus = class (TTs)
fcount_places: integer;
constructor Create (name:string; mosh_dvig:integer; count_places:integer);
destructor Destroy;
procedure show (Mem:TMemo);
end;
```

В разделе реализации определяем конструктор класса *TTs*.

```
constructor TTs.Create (name:string; mosh_dvig:integer);
begin
 fname:=name;
 fmosh_dvig:=mosh_dvig;
end;
```

Реализация конструктора класса *TBus* выглядит следующим образом.

```
constructor TBus.Create (name:string; mosh_dvig:integer; count_places:integer);
begin
  inherited create (name, mosh_dvig);
 fcount_places:=count_places;
end;
```

Реализация деструкторов классов *TTs* и *TBus* соответственно.

```
destructor TTs.Destroy;
begin
end;
destructor TBus.Destroy;
begin
  inherited;
end;
```

Реализация метода *Show* класса *TBus*.

```
procedure tbus.show (Mem:TMemo);
begin
        mem.lines.add(IntToStr(i)+')'+ fname+' '+'Мощность двигателя
='+IntToStr(fmosh_dvig)+' л.с.'+' Кол-во посад.мест= '+IntToStr(fcount_places));
end;
```

В разделе объявления констант *const* (данный раздел располагается между разделом описания типов и разделом переменных) объявляем значение константы *count* – максимальное число элементов массива (списка автобусов).

```
const count=5;
```

В разделе переменных *Var* объявляем массив объектов (список автобусов) *Bus* класса *TBus*.

```
var
  Form1: TForm1;
  Bus:array [1..count] of TBus;
  i,n:integer; //счетчики элементов массива Bus
```

Обрабатываем событие *OnCreate* для формы *Form1*. При создании формы переменной (счетчику) *n* присваивается значение равное *1*.

```
procedure TForm1.FormShow(Sender: TObject);
begin
   n:=1;
end;
```

Обработчик события *OnClick* для объекта *Button1*. Осуществляет создание объектов класса *TBus*, помещает данные об автобусах в список.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 if  n<=count then
 begin
     Bus[n]:=TBus.Create(edit1.Text,StrToInt(edit2.Text),StrToInt(edit3.text));
     n:=n+1;
 end
else ShowMessage('Список заполнен!');
   edit1.Text:='';
   edit2.Text:='';
   edit3.Text:='';
end;
```

Вывод списка осуществляется применением метода *show* к элементам массива (обработчик события *OnClick* для объекта *Button2*):

```
procedure TForm1.Button2Click(Sender: TObject);
begin
```

```
  memo1.Lines.Clear;
  for i:=1 to count do
  if Bus[i] <> NIL then Bus[i].show(Memo1);
end;
```

Обработчик события *OnClick* для объекта *Button3*. Осуществляет уничтожение объектов класса *TBus*

```
procedure TForm1.Button3Click(Sender: TObject);
begin
  for i:=1 to count do
  if Bus[i]<>NIL then
  begin
     Bus[i].Destroy;
     Bus[i]:=nil;
  end;
  memo1.Lines.Clear;
  n:=1;
end;
```

1.5.2 Динамическое создание компонентов

Вызовы конструкторов и деструкторов визуальных компонентов Delphi. Любой компонент, попавший приложение при визуальном проектировании, включается в определенную иерархию объектов, которая замыкается на форме (класс *TForm*). Поэтому вызов конструкторов и деструкторов всех компонентов формы производится автоматически при инициализации и удалении формы, незримо для программиста.

Сами формы создаются и уничтожаются приложением – глобальным объектом с именем *Application*. В файле-проекте с расширением **.dpr* можно увидеть вызов конструктора формы в виде строки:

Application.CreateForm(Tform1, Form1);

Динамически создаваемые компоненты – это компоненты, место в памяти под которые выделяется по мере необходимости в процессе работы приложения. Этим они и отличаются от компонентов, которые помещаются на Форму при проектировании приложения. Возможность создавать компоненты динамически это очень большое удобство для программиста. Например, можно создавать в цикле сразу много однотипных компонентов, формируя из них массив, которым в дальнейшем очень просто управлять.

Все компоненты, как объекты, имеют множество свойств, определяющих их работу. При установке компонента на Форму из палитры большинство этих свойств определяются системой Delphi автоматически. При создании динамического компонента программист должен описать и настроить их вручную.

Прежде всего, для появления динамически создаваемого компонента нужно выделить под него место в памяти. Выделением места в памяти компьютера под любой компонент занимается конструктор типа объекта этого компонента – метод *Create*. Для этого сначала нужно описать переменную нужного типа, а затем для выделения памяти воспользоваться методом *Create*. Метод *Create* имеет параметр *Owner*, определяющий так называемого «владельца» для создаваемого компонента.

При обычной установке компонента из палитры система делает владельцем этого компонента Форму. Проще всего поступать так же. Однако можно указать в качестве владельца сам этот компонент, воспользовавшись в качестве параметра ключевым словом *Self*.

Когда компонент создан, то есть место в памяти под него выделено, можно задавать значения параметрам этого объекта. Прежде всего, это ещё один компонент, так называемый «родитель». Компонент-родитель будет отвечать за отрисовку нашего динамически создаваемого компонента. Это значит, что новый компонент появится в границах компонента-родителя.

Если компонент-владелец имеет тип *Tcomponent*, то есть может быть любым компонентом, то компонент-родитель уже имеет тип *TwinControl*. То есть это должен

быть «оконный» компонент, умеющий принимать и обрабатывать сообщения от системы Windows. Это необходимо, так как компонент должен находиться в некоторой иерархии компонентов, принимающих и передающих сообщения от системы Windows. Нашему динамическому компоненту сообщения будут передаваться через компонент-родитель.

Компонент не может быть родителем для самого себя. Имя компонента-родителя просто присваивается свойству *Parent* создаваемого динамически компонента. Общая схема «конструирования» динамически создаваемого компонента:

```
var Component: Tcomponent;//Описать переменную для компонента
begin
  Component:=Tcomponent.Create(Owner);//Задать владельца
      Component.Parent:=Parent;//Задать родителя
  end;
```

На этом создание компонента можно считать законченным, и он успешно появляется (или «не появляется», если он не визуальный) в приложении. Остальные свойства будут присвоены ему по умолчанию самой системой Delphi.

Для примера динамически создадим многострочный редактор, компонент *Memo*. Пусть он появляется на форме по нажатию кнопки:

```
procedureTForm1.Button1Click(Sender: Tobject);
var Memo: Tmemo;
begin
Memo:=Tmemo.Create(Form1);
Memo.Parent:=Form1;
Memo.Left:=50;
Memo.Top:=50;
Memo.Width:=250;
Memo.Height:=100;
Memo.Text:='Компонент Memo динамически создан!';
end;
```

По умолчанию Delphi присвоит ему типовое имя с присвоением очередного порядкового номера: *Memo1*. Программист при создании компонента также может присвоить свойству *Name* нужное значение, например:

Memo.Name:='DynamMemo';

К данному компоненту можно обращаться как по этому имени, так и с указанием переменной, с помощью которой он был создан: *Memo*. В последнем случае переменная должна быть глобальной.

Создадим кнопку удаления динамически созданного объекта Memo. Формируем обработчик события *OnClick*.

If Memo<>nil then a.Destroy;

1.6 Варианты индивидуальных заданий

1 Сформировать список городов, объявив в качестве родительского класс Tterritoria;

2 Сформировать список отделений больницы, объявив в качестве родительского класс Thospital;

3 Сформировать список молочных продуктов, объявив в качестве родительского класс Tproduct;

4 Сформировать список кафедр, объявив в качестве родительского класс Tpodrazdel;

5 Сформировать список ноутбуков, объявив в качестве родительского класс Tcomputer;

6 Сформировать список журналов, объявив в качестве родительского класс Tizdanie;

7 Сформировать список заводов Оренбургской области, объявив в качестве родительского класс Tenterprise;

8 Сформировать список программистов, объявив в качестве родительского класс Tsotrudnik;

9 Сформировать список крылатых насекомых, объявив в качестве родительского класс Tnasekom;

10 Сформировать список служебных программ, объявив в качестве родительского класс Tpo;

11 Сформировать список квартир, объявив в качестве родительского класс Tnedvizh;

12 Сформировать список озер Оренбургской области, объявив в качестве родительского класс Tvodoem;

13 Сформировать список фигуристов, объявив в качестве родительского класс Tsportsman;

14 Сформировать список рекламных агентств, объявив в качестве родительского класс Tservice;

15 Сформировать список радиостанций, объявив в качестве родительского класс Tsmi.

2 Лабораторная работа №5. Создание виртуального метода в классе

2.1 Цель работы

Научиться описывать классы, содержащие виртуальные методы, согласно правил вызова.

2.2 Ход работы

Задание общего уровня:

1 Изучить методические рекомендации к лабораторной работе;

2 Создать новый проект delphi, осуществляющий занесение данных о

животных зоопарка в список;

3 Спроектировать форму приложения, согласно рисунка 3, представленного в методических рекомендациях;

4 Описать родительский класс tanimal в секции interface программного модуля ниже описания класса tform1:

1) в классе описать поле NameAnimal типа string;

2) описать конструктор Create, осуществляющий присвоение полю объекта начального значения;

3) описать метод GetInfo, который будет формировать строки списка, метод сделать виртуальным;

5 Описать дочерний класс Tmammal:

1) в классе описать поле age типа real;

2) описать конструктор Create, осуществляющий присвоение полям объекта начального значения;

3) описать метод GetInfo, согласно всех правил вызова виртуальных методов;

6 Описать дочерний класс Treptiles:

1) в классе описать поле Length типа real;

2) описать конструктор Create, осуществляющий присвоение полям объекта начального значения;

3) описать метод GetInfo, согласно всех правил вызова виртуальных методов;

7 В секции реализации implementation описать реализацию всех методов трех созданных классов (см. методические рекомендации);

8 Объявить секцию const после описания всех классов, задать переменную, определяющую максимальное количество животных в списке;

9 В глобальном разделе описания переменных объявить массив животных типа tanimal;

10 Объявить переменную «счетчик» количества животных;

11 Для компонента button1 организовать событие onclick, осуществляющее

создание списка;

12 Для компонента button2 организовать событие onclick, осуществляющее вывод списка на экран;

13 Обработать дополнительные методы проекта;

14 Сохранить проект на диске в директории lab5_1;

15 Провести отладку и компиляцию проекта;

16 Дописать деструктор в каждом из трех созданных классов;

17 Реализовать созданные методы, при закрытии проекта вызвать их на выполнение;

18 Сохранить проект, выполнить отладку и компиляцию;

19 Оформить отчет о проделанной работе;

Задание повышенного уровня:

20 Выполнить индивидуальное задание согласно выданного варианта;

21 В проекте описать собственные классы, организовать виртуальные методы, описать и реализовать конструкторы и деструкторы для объектов описанных классов;

22 Сохранить проект на диске в директории lab5_2;

23 Выполнить отладку и компиляцию проекта;

24 Оформить отчет о проделанной работе;

25 Защитить работу.

2.3 Содержание отчета

1 Цель, ход работы;

2 Постановка задачи №1, листинг программного модуля, результат работы приложения;

3 Формулировка индивидуального задания;

4 Листинг программного модуля индивидуального задания, результат работы приложения;

5 Вывод.

2.4 Контрольные вопросы

1 Охарактеризуйте основные понятия объектно-ориентированного программирования, опишите программное обращение;

2 Охарактеризуйте принципы объектно-ориентированного программирования, приведите примеры программного описания;

3 Что представляет собой раннее и позднее связывание?

4 В чем заключается смысл вызова виртуального метода, отличие виртуальных, динамических и статических методов;

5 Перечислите правила вызова виртуальных методов, особенности объявления виртуального метода в дочерних классах;

6 Каково назначение конструктора и деструктора?

7 Для чего используется директива inherited?

2.5 Методические рекомендации

2.5.1 Понятие полиморфизма. Раннее и позднее связывание

Полиморфизм – это свойство объектов разных классов выполнять одно и то же действие согласно своего типа.

Два или более класса, которые являются производными одного и того же базового класса, называются полиморфными. Это означает, что они могут иметь общие характеристики, но так же обладать собственными свойствами.

В рамках ООП поведенческие свойства объекта определяются набором входящих в него методов. Изменяя алгоритм того или иного метода в потомках объекта, программист может придавать этим потомкам отсутствующие у родителя

специфические свойства. Для изменения метода необходимо перекрыть его в потомке, т.е. объявить в потомке одноименный метод и реализовать в нем нужные действия. В результате чего в объекте-родителе и объекте-потомке будут действовать два одноименных метода, имеющих разную алгоритмическую основу и, следовательно, придающие объектам разные свойства. Это и называется полиморфизмом объектов.

```
Type
TAnimal=class
...
Procedure GetEat;
End;
TMammal=class(TAnimal)
...
Procedure GetEat(count:real);
End;
```

Методы объектов бывают статическими, виртуальными и динамическими.

Статические методы включаются в код программы при компиляции. Это означает, что до использования программы определено, какая процедура будет вызвана в данной точке. Компилятор определяет, какого типа объект используется при данном вызове, и подставляет метод этого объекта.

Объекты разных типов могут иметь одноименные статические методы. В этом случае нужный метод определяется по типу экземпляра объекта.

Это удобно, так как одинаковые по смыслу методы разных типов объектов можно и назвать одинаково, а это упрощает понимание и задачи и программы. Статическое перекрытие – первый шаг полиморфизма. Одинаковые имена – вопрос удобства программирования, а не принцип.

Процесс, с помощью которого вызовы статических методов однозначно разрешаются компилятором во время компиляции в один метод, называется **ранним связыванием**. При раннем связывании вызывающий и вызываемый методы связываются при первой же возможности, т.е. во время компиляции. При **позднем**

связывании вызывающий и вызываемый методы не могут быть связаны во время компиляции, поэтому включается механизм, позволяющий осуществить связывание несколько позднее, когда вызов действительно произойдет.

Виртуальные методы в отличие от статических, подключаются к основному коду на этапе выполнения программы. Виртуальные методы дают возможность определить тип и конкретизировать экземпляр объекта в процессе исполнения, а затем вызвать методы этого объекта.

Описание виртуального метода отличается от описания обычного метода добавлением после заголовка метода служебного слова **virtual** .

procedure Method (список параметров); ***virtual****;*

Объявление виртуального метода в базовом классе выполняется с помощью ключевого слова **virtual**, а его перекрытие в производных классах - с помощью ключевого слова **override**. Перекрытый метод должен иметь точно такой же формат (список параметров, а для функций еще и тип возвращаемого значения), что и перекрываемый.

```
Type
TAnimal=class
...
Procedure GetEat;
Function GetInfo : string; virtual;
End;
TMammal=class(TAnimal)
...
Function GetInfo : string; override;
End;
```

Использование виртуальных методов в иерархии типов объектов имеет определенные ограничения:

1 Если метод объявлен как виртуальный, то в типе потомка его нельзя перекрыть статическим методом;

2 Объекты, имеющие виртуальные методы, инициализируются специальными процедурами, которые, в сущности, также являются виртуальными и носят название **constructor**;

3 Списки переменных, типы функций в заголовках перекрывающих друг друга виртуальных процедур и функций должны совпадать полностью;

Обычно на **конструктор** возлагается работа по инициализации экземпляра объекта: присвоение полям исходных значений, первоначальный вывод на экран и т.п.

Помимо действий, заложенных в него программистом, конструктор выполняет подготовку механизма позднего связывания виртуальных методов. Это означает, что еще до вызова любого виртуального метода должен быть выполнен какой-нибудь конструктор.

Конструктор – это специальный метод, который инициализирует объект, содержащий виртуальные методы. Заголовок конструктора выглядит так:

constructor Method (список параметров);

Зарезервированное слово **constructor** заменяет слова procedure и virtual .

Основное и особенное назначение конструктора – установление связей с таблицей виртуальных методов (VMT) – структурой, содержащей ссылки на виртуальные методы. Таким образом, конструктор инициализирует объект установкой связи между объектом и VMT с адресами кодов виртуальных методов. При инициализации и происходит позднее связывание.

У каждого объекта своя таблица виртуальных методов VMT . Именно это и позволяет одноименному методу вызывать различные процедуры.

Упомянув о конструкторе, следует сказать и о **деструкторе**. Его роль противоположна: выполнить действия, завершающие работу с объектом, закрыть все файлы, очистить динамическую память, очистить экран и т.д.

Заголовок деструктора выглядит таким образом:

destructor Done ;

Основное назначение деструкторов – уничтожение VMT данного объекта. Часто деструктор не выполняет других действий и представляет собой пустую процедуру.

```
destructor Done ;
begin
end ;
```

Пример описания корректного описания виртуального метода:

```
Type
TAnimal=class
…
Constructor Create(NameAnimal:string);
Function GetInfo : string; virtual;
End;
TMammal=class(TAnimal)
…
Constructor Create(NameAnimal:string; Age:real);
Function GetInfo : string; override;
End;
….
Var
AnyAnimal:TAnimal;
```

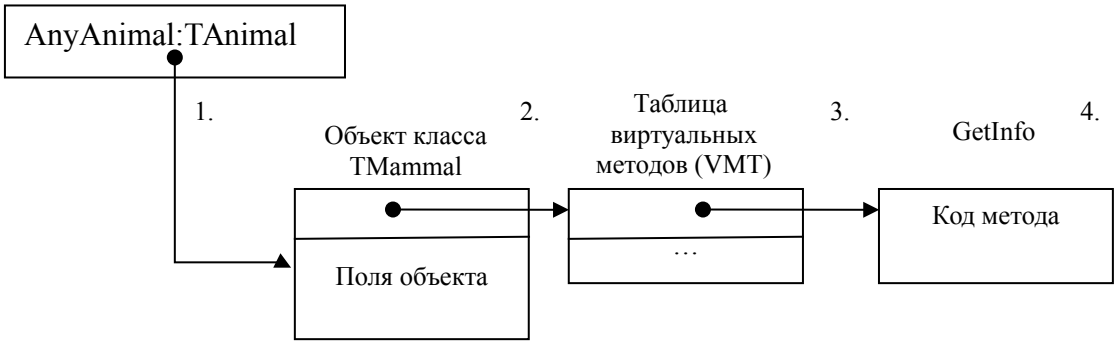

Рисунок 2 – Механизм вызова виртуального метода

Вызов виртуального метода осуществляется следующим образом:

1 Через объектную переменную выполняется обращение к занятому объектом блоку памяти;

2 Далее из этого блока извлекается адрес таблицы виртуальных методов (он записан в четырех первых байтах);

3 На основании порядкового номера виртуального метода извлекается адрес соответствующей подпрограммы;

4 Вызывается код, находящийся по этому адресу.

В реализации метода конструктора в дочернем классе используется директива **inherited**. Ключевое слово **Inherited** используется, чтобы вызвать родительский конструктор или метод деструктора, как соответствующий для текущего класса.

Оно вызывается в начале конструктора, и в конце деструктора. Это не является обязательным, но рекомендуется.

Без параметров Inherited вызывает так же названный метод родительского класса, с теми же самыми параметрами.

1 Create;

```
Begin
Inherited; // Всегда вызывается в начале конструктора
...
end;
```

2 Create(arguments);

```
Begin
Inherited Create(arguments); // Всегда вызывается в начале конструктора
...
end;
```

3 Destroy

```
Begin
...
Inherited; // Всегда вызывается в конце деструктора
```

end;

2.5.2 Задание для выполнения

Постановка задачи: Спроектировать приложение, в котором осуществить запись данных о животных зоопарка в список. Вывод списка животных на экран по щелчку на кнопку.

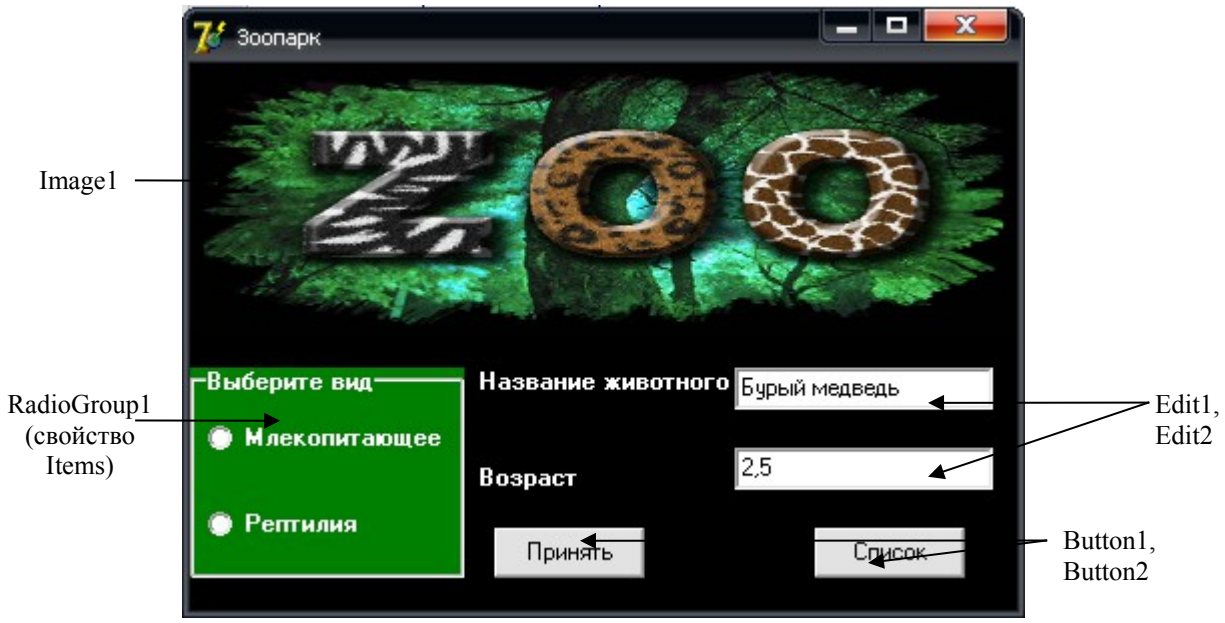

Рисунок 3 – Примерный внешний вид формы

Родительский класс:

type

TAnimal=class

NameAnimal:string;

constructor Create (FNameAnim:string) ;

function GetInfo: string; virtual;

end;

Дочерний класс (Млекопитающие)

TMammal=class(TAnimal)

Age:real;

```
constructor Create(FNameAnim:string; FAge:real);
function GetInfo:string; override;
end;
```

Дочерний класс (Рептилии)

```
TReptiles=class(TAnimal)
Length:real;
constructor Create(FNameAnim:string; FLength:real);
function GetInfo:string; override;
end;
```

Реализация методов класса TAnimal:

```
constructor TAnimal.Create (FNameAnim:string);
begin
NameAnimal:=FNameAnim;
end;
function TAnimal.GetInfo:string;
begin
Result:=NameAnimal;
end;
```

Реализация методов класса TMammal:

```
constructor TMammal.Create(FNameAnim:string; FAge:real);
begin
inherited create(FNameAnim);
Age:=FAge;
end;
function TMammal.GetInfo:string;
begin
result:=NameAnimal+' возраст ' +FloatToStr(Age);
end;
```

Реализация методов класса TReptiles:

```
constructor TReptiles.create(FNameAnim:string; FLength:real);
```

```
begin
inherited create(FNameAnim);
Length:=FLength;
end;
function TReptiles.GetInfo:string;
begin
result:=NameAnimal+' длина животного '+FloatToStr(Length);
end;
```

Раздел констант в среде Delphi располагается между секцией описания типов и глобальным разделом описания переменных

```
const
Size=10;  // размер списка
var
  Form1: TForm1;
  Massiv: array[1..Size] of TAnimal; // список
  n:integer;     // счетчик количества животных в списке
```

Обработчик события OnClick для объекта Button1. Осуществляет создание объектов описанных классов, помещает данные о животных в список.

```
procedure TForm1.Button1Click(Sender: TObject);
begin
IF  n<=Size THEN
begin
if RadioGroup1.ItemIndex=0
then                                    // создадим объект TMammal
                Massiv[n]:=TMammal.Create(Edit1.Text,StrToFloat(Edit2.Text))
Else                                    // создадим объект TReptiles
Massiv[n]:=TReptiles.Create(Edit1.Text,StrToFloat(Edit2.Text));

n:=n+1;
End
```

ELSE ShowMessage('Список заполнен!');

end;

Обработчик события OnClick компонента Button2, осуществляющий вывод списка животных в окне сообщения.

```
procedure TForm1.Button2Click(Sender: TObject);
var
i:integer ;
st:string;
begin
for i:=1 to Size do
                    if list[i] <> NIL then st:=st+Massiv[i].Getinfo+#13;
ShowMessage('Животные зоопарка: '+#13+st);
end;
```

Дополнительные методы проекта

Задание начального значения элементу списка:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
n:=1;
end;
```

Формирование текста надписи в зависимости от выбранного животного:

```
procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
if RadioGroup1.ItemIndex=0 then Label2.Caption:='Возраст';
if RadioGroup1.ItemIndex=1 then Label2.Caption:='Длина животного';
end;
```

2.6 Варианты индивидуальных заданий

1 Сформировать, применяя метод полиморфизма список обитателей озера Байкал, объявив родительский класс как TFauna;

2 Сформировать, применяя метод полиморфизма список обитателей с/х фермы, родительский класс TFerma;

3 Сформировать, применяя метод полиморфизма список лекарственных растений, родительский класс TFlora;

4 Сформировать, применяя метод полиморфизма список животных опасных для жизни человека, родительский класс TFauna;

5 Сформировать, применяя метод полиморфизма список ядовитых растений, родительский класс TFlora;

6 Сформировать, применяя метод полиморфизма список музыкальных инструментов для оркестра, родительский класс TInstument;

7 Сформировать, применяя метод полиморфизма список обитателей реки Лимпопо, родительский класс TFauna;

8 Сформировать, применяя метод полиморфизма список инструментов, применяемых в строительстве, родительский класс TInstrument;

9 Сформировать, применяя метод полиморфизма список цветов для создания композиции, родительский класс TFlora;

10 Сформировать, применяя метод полиморфизма список орудий Второй Мировой Войны, родительский класс TOrygie;

11 Сформировать, применяя метод полиморфизма список транспорта аэрофлота города, родительский класс TTransport;

12 Сформировать, применяя метод полиморфизма список восточных единоборств, родительский класс TMartialArts;

13 Сформировать, применяя метод полиморфизма список жанров кино, родительский класс TCinema;

14 Сформировать, применяя метод полиморфизма список уровней компьютерной игры, родительский класс TGame;

15 Сформировать, применяя метод полиморфизма список канцелярских принадлежностей школьника, родительский класс TStationery.

Список использованных источников

1 Хомоненко А.Д. Delphi 7 / А.Д. Хомоненко [и др.]. – СПб.: БХВ-Петербург, 2008. – 1216 с. : ил.

2 Культин Н.Б. Delphi в задачах и примерах [Комплект] / Н.Б. Культин. – 3 изд. – СПб.: БХВ-Петербург, 2012. – 228 с. : ил. – 1 электрон. опт. диск (CD-ROM).

3 Катаев М.Ю. Объектно-ориентированное программирование : лабораторный практикум / М.Ю. Катаев. – Томск: Томский межвузовский центр дистанционного образования, 2007. – 68 с.

4 Марченко А.И. Программирование в среде TurboPascal 7.0 : учебное пособие / А.И. Марченко, Л.А. Марченко. – 9 изд. – СПб.: Корона-Век, 2007. – 458 с.

5 Бескоровайный И.В. Азбука Delphi : программирование с нуля / И.В. Бескоровайный. – Новосибирск: Сиб. унив. изд-во, 2008. – 112 с.

6 Хомоненко А. Delphi 7. [Электронный ресурс] : Электронно-библиотечная система ibooks.ru. / А. Хомоненко, В. Гофман, Е. Мещеряков. – СПб.: Айбукс. – 2010. – Режим доступа : http://ibooks.ru/product.php?productid=18411

7 Профессиональные программы для разработчиков [Электронный ресурс] : Delphi World / под ред. Н. Акулова. – Алматы: WDS, 2002. – Режим доступа : http://delphiworld.narod.ru/.

8 Королевство Delphi. Виртуальный клуб программистов [Электронный ресурс] / под ред. Е. Филипповой. – М.: DOTNETPARK, 1998. – Режим доступа : http://delphikingdom.ru/index.asp.

9 Высокоуровневые методы математики и информатики [Электронный ресурс] : Киберфак / под ред. Ni-Cd. – М.: МИФИ, 2007-2014. – Режим доступа : http://cyberfac.ru/publ/informatika/vysokourovnevye_metody_informatiki_i_programmirovanija/konstruktor_i_destruktor/29-1-0-946.