# Лекция 14

1. ***Объектно-ориентированное программирование***
    1. ***Инкапсуляция***
    2. ***Наследование***
    3. ***Полиморфизм***
2. ***Классы и объекты***
3. ***Описание объектов в Delphi***
5. ***Поля, свойства, методы***
6. ***Реализация методов в Delphi***
7. ***События***
8. ***Инкапсуляция. Интерфейс объекта***
8. ***Конструкторы и деструкторы***
9. ***Наследование***
10. ***Полиморфизм***
11. ***Визуальная библиотека компонент***

# Объектно-ориентированное программирование: инкапсуляция, наследование, полиморфизм

Объектно-ориентированное программирование (ООП) – это методика разработки программ, в основе которое лежит понятие объекта, как некоторой структуры, описывающей объект и его поведение.

Основными принципами ООП являются:
**инкапсуляция** – объединение данных и обрабатывающих их методов внутри класса. Это означает, что объединяются и помещаются внутри класса (инкапсулируются) поля, свойства и методы. При этом класс приобретает определенную функциональность;

**наследование** заключается в порождении новых объектов-потомков от существующих объектов-родителей. При этом потомок берет от родителя все его поля, свойства и методы, которые можно использовать в неизменном виде или переопределять. Удалить какие-то элементы родителя в потомке нельзя;

**полиморфизм** заключается в том, что методы различных объектов могут иметь одинаковые имена, но различное содержание. Это достигается переопределением родительского метода в классе-потомке. В результате родитель и потомок ведут себя по-разному.

# Классы и объекты

Объекты – ключевое понятие объектно-ориентированных технологий. Оглядитесь: мы живём в мире взаимодействующих объектов!

Каждый объект характеризуется состоянием и поведением.

Объекты в программных системах – аналоги объектов в реальном мире. Состояние хранится в полях . Поведение доступно через методы.

Методы изменяют состояние объекта. Сокрытие внутреннего состояния объекта от посторонних глаз называется инкапсуляцией. Единственный способ изменить это состояние – вызвать метод объекта.

Класс – множество объектов, имеющих схожее поведение и свойства.

# Класс - автомобили

Свойства

- Марка
- Цвет
- Скорость движения
- Объем двигателя

Методы

- Переключение скоростей
- Торможение
- Ускорение

Объекты класса – автомобиль Audi, мой автомобиль, автомобиль соседа, автомобиль Петрова

# Классы и объекты в Delphi

**Классы** – это специальные типы данных, используемые для описания объектов. Соответственно объект, имеющий тип какого-либо класса, является экземпляром этого класса или переменной этого типа.

Класс – это сложная структура, включающая, помимо описания данных, описание процедур и функций, которые могут быть выполнены над представителем класса – **объектом** (экземпляром класса).

Класс имеет в своем составе поля, свойства и методы.

**Поля** класса аналогичны полям записи и служат для хранения информации об объекте.

**Методами** называются процедуры и функции, предназначенные для обработки полей.

**Свойства** реализуют механизм доступа к полям и занимают промежуточное положение между полями и методами. С одной стороны, свойства можно использовать как поля, присваивая им значения с помощью оператора присваивания; с другой стороны, внутри класса доступ к значениям свойств выполняется методами класса.

# Описание объекта в Delphi

**Type <имя класса> = class (<имя класса-родителя>)**
**private**
<содержит частные описания членов класса, которые доступны только в том модуле, где данный класс описан>
**protected**
<содержит защищенные описания членов класса, которые доступны также внутри методов класса являющихся наследниками данного класса и описанных в других модулях>
**public**
<содержит общедоступные описания членов класса, которые доступны в любом месте программы, где доступен сам класс >
**published**
<содержит опубликованные описания членов класса, которые доступны для редактирования и изменения значений во время проектирования приложения. Это то, что мы видим в Инспекторе объектов>
**end;**

# Примеры

Tobject - базовый класс для всех классов
Обычно имя класса начинается с большой буквы T

```
unit Unit1;
Interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit3: TEdit;
    Button1: TButton;
    RadioGroup1: TRadioGroup;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;

implementation
```

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b: integer;
   c:extended;
begin
  if (Edit1.Text <>'') and (Edit2.Text <>'') then
    begin
    a:=StrToInt(Edit1.Text);
    b:=StrToInt(Edit2.Text);
    Edit3.Text:='';
    Case RadioGroup1.ItemIndex of
    0: c:=a+b;
    1: c:=a-b;
    2: c:=b*a;
    3: if b=0 then begin showmessage('На ноль делить нельзя!');
                exit; end
                else c:=b/a
    end;

    Edit3.Text:=FloatToStrF(c,ffGeneral,10,4);
    end
  else Showmessage('Не заданы значения');
end;

end.
```

# Методы

- Метод представляет собой подпрограмму (процедуру или функцию), являющуюся элементом класса.
- Описание метода - обычная подпрограмма модуля.
- Заголовок метода располагается в описании класса

# Реализация метода

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b: integer;
    c:extended;
begin
  if (Edit1.Text <>'') and (Edit2.Text <>'') then
    begin
    a:=StrToInt(Edit1.Text);
    b:=StrToInt(Edit2.Text);
    Edit3.Text:='';
    Case RadioGroup1.ItemIndex of
    0: c:=a+b;
    1: c:=a-b;
    2: c:=b*a;
    3: if b=0 then begin showmessage (‘На ноль делить нельзя!');
                exit; end
                else c:=b/a
    end;

    Edit3.Text:=FloatToStrF(c,ffGeneral,10,4);
    end
  else Showmessage(‘Не заданы значения’);
end;

end.
```

```
Interface
uses
  Windows, Messages, SysUtils, Variants,
       Classes, Graphics, Controls, Forms;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit3: TEdit;
    Button1: TButton;
    RadioGroup1: TRadioGroup;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Большинству методов при вызове передаются параметр Sender, имеющий тип TObject.

```
procedure TForm1.Button1Click(Sender: TObject);
var a,b: integer;
   c:extended;
begin
  if (Edit1.Text <>'') and (Edit2.Text <>'') then
    begin
    a:=StrToInt(Edit1.Text);
    b:=StrToInt(Edit2.Text);
    Edit3.Text:='';
    Case RadioGroup1.ItemIndex of
    0: c:=a+b;
    1: c:=a-b;
    2: c:=b*a;
    3: if b=0 then begin showmessage('На ноль
делить нельзя!');
                exit; end
                else c:=b/a
    end;

    Edit3.Text:=FloatToStrF(c,ffGeneral,10,4);
    end
  else Showmessage('Не заданы значения');
end;

end.
```

# Инкапсуляция

- Инкапсуляция (encapsulation) – «заключение в капсулу»- ограничение доступа к содержимому «капсулы» извне и отсутствие такого ограничения внутри «капсулы»
  - Один из основных принципов ООП
  - Свойство языка программирования, позволяющее пользователю не задумываться о сложности реализации используемого программного компонента (о том, что у него внутри), а взаимодействовать с ним посредством предоставляемого интерфейса (публичных методов и членов), а также объединить и защитить жизненно важные для компонента данные.

- **Инкапсуляция** – объединение данных и обрабатывающих их методов внутри класса. Это означает, что объединяются и помещаются внутри класса (инкапсулируются) поля, свойства и методы.

# Интерфейс объекта

- При соблюдении принципа инкапсуляции пользователю предоставляется только спецификация (интерфейс) объекта. Пользователь может взаимодействовать с объектом только через этот интерфейс. Реализуется с помощью ключевого слова: public.
- Пользователь не может использовать закрытые данные и методы. Реализуется с помощью ключевых слов: private, protected

```
Interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Edit1: TEdit;
    Edit3: TEdit;
    Button1: TButton;
    RadioGroup1: TRadioGroup;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Метод, объявленный в классе, может вызываться различными способами. Вид метода определяется модификатором, который указывается в описании класса после заголовка метода и отделяется от заголовка точкой с запятой. По умолчанию все методы считаются статическими и вызываются как обычные подпрограммы.

Примеры некоторых модификаторов:
virtual – виртуальный метод
dynamic – динамический метод
override – перекрывающий метод
message –метод обработки сообщения
abstract – абстрактный метод

# Конструкторы и деструкторы

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>)];

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

**End**;

**Примечания:**

• Объявляются конструкторы и деструкторы, как правило, в разделе Public класса.

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

• В одном классе может быть объявлено несколько деструкторов, но чаще бывает один деструктор без параметров с именем Destroy.

***Свойства*** – это переменные, которые влияют на состояние объекта. Например, ширина, высота, положение кнопки на форме или надпись на ней.

**Методы** – это те же процедуры и функции, то есть это то, что объект умеет делать (вычислять). Например, объект может иметь процедуру для вывода какого-то текста на экран. Кнопка при нажатии меняет форму – это метод кнопки, процедура прорисовки вида нажатой и не нажатой кнопки.

**События** – это те же процедуры и функции, которые вызываются при наступлении определенного события. Например, пользователь нажал на кнопку, вызывается процедура обработки этого нажатия. Или мышка оказалась над кнопкой – вызывается процедура обработки этого события, если программист ее создал.

**Компоненты** – это объекты, с которыми можно работать визуально.
Существуют и не визуальные компоненты, например, диалоги.
Это то , с чем мы работаем через панели инструментов.

# События

- Операционная система **Windows** - многозадачная, т.е. несколько программ в ней могут функционировать одновременно. Когда, например, мы щёлкаем по кнопке в окне нашей программы, система **Windows** определяет, что произошло событие именно в нашей программе, и посылает ей сообщение об этом. Наша программа должна соответствующим образом отреагировать на него. Для этого программисты должны написать код-обработчик этого события. Таким образом, структура программы для **Windows** представляет собой набор подпрограмм, каждая из которых ответственна за обработку конкретного события и вызывается только при его возникновении. **Удобство Delphi** состоит в том, что мы избавлены от необходимости получать сообщения от **Windows** сами, **Delphi** это делает за нас. Каждый компонент имеет впечатляющий набор событий, на которые он может реагировать. Программист сам определяет, какие события в программе требуется обрабатывать.

- Для управления событиями используется Инспектор объектов.

# Наследование

**Наследование** — механизм ООП, позволяющий описать новый класс на основе уже существующего (родительский класс, родитель, прародитель, предок, ancestor ), при этом свойства и функциональность родительского класса заимствуются новым классом.

Другими словами, класс-наследник (потомок, дочерний, descendant) реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса.

Для использования механизма наследования необходимо в объявлении класса справа от слова class указать класс предок:

Предок:

```
TAncestor = class(TObject)
```

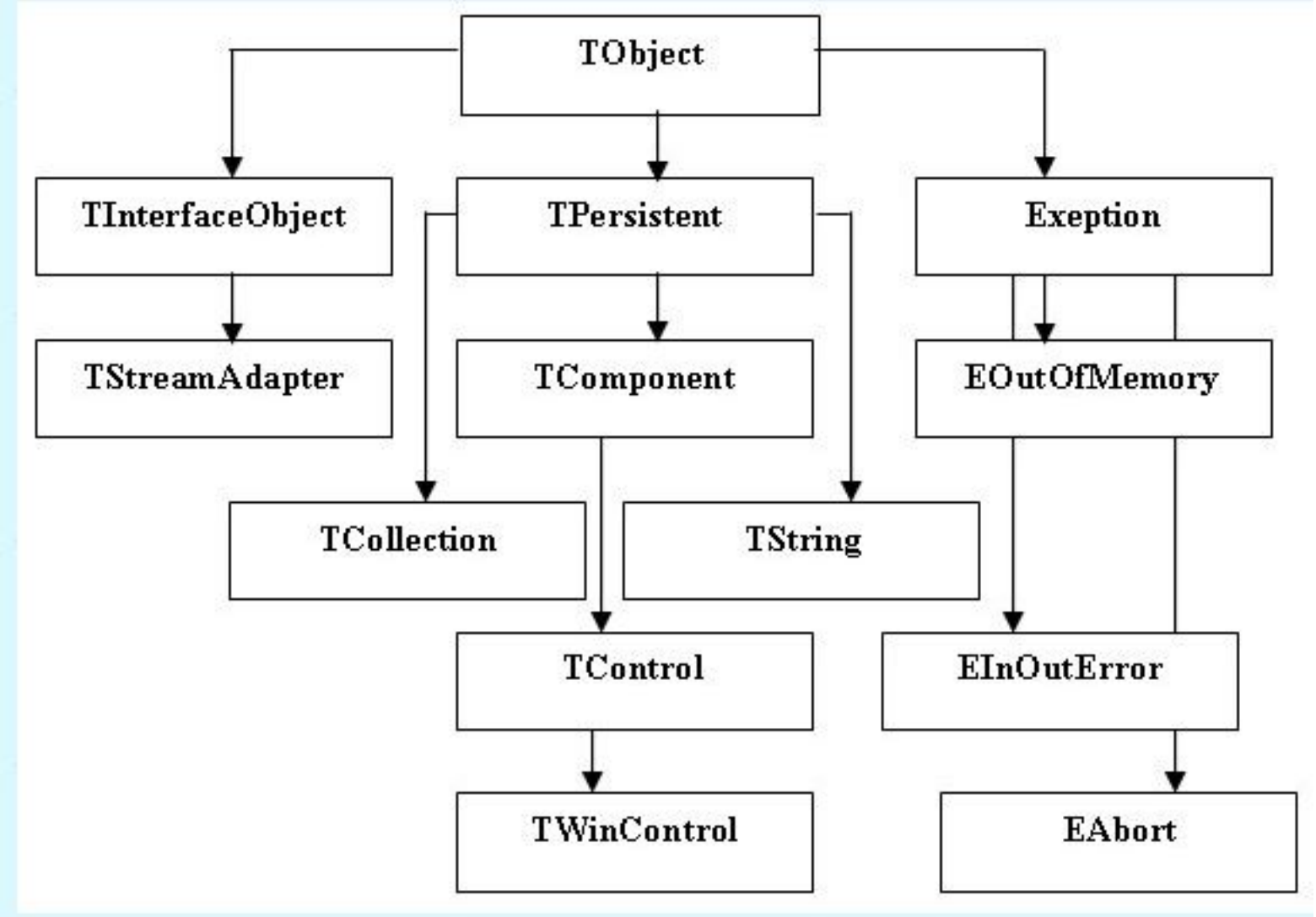
```
TAncestor = class
```

```
TDescendant = class(Tancestor)
```

Абсолютно все классы в Delphi являются потомками базового класса Tobject.

Если класс-предок не указан, то подразумевается, что новый класс является прямым потомком класса TObject.

# Иерархия базовых классов в VCL




Фрагмент иерархии базовых классов

# Полиморфизм

- **Полиморфизм** — это возможность использовать одинаковые имена для методов, входящих в различные классы.
  **Концепция полиморфизма** обеспечивает при применении метода к объекту использование именно того метода, который соответствует классу объекта.

Переопределение метода – создание метода с тем же именем, что и класс у родителя, но реализованного другим образом.

Принцип полиморфизма заключается в том, что в каждом классе наследнике могут быть переопределены методы класса родителя.

# Пример использования объекта

Для примера возьмем объект — кнопку. Такой объект должен обладать следующим минимальным набором.

1. **Свойства:** левая позиция **Left**;
   верхняя позиция **Top**;
   ширина **Width**;
   высота **Height**;
   заголовок **Caption**;
2. **Методы:** создать кнопку;
   нарисовать кнопку;
   уничтожить кнопку.
3. **События:** кнопка нажата;
   кнопка отпущена;
   заголовок кнопки изменен.

Мы изменили заголовок кнопки. Объект генерирует событие "заголовок кнопки изменен". По этому событию вызывается метод "нарисовать кнопку". Этот метод рисует кнопку в позиции, указанной в свойствах объекта, и выводит на кнопке новый текст, указанный в свойстве "заголовок".

У каждого объекта обязательно присутствуют два метода: "создать объект" и "уничтожить объект". Во время создания объекта происходит выделение памяти для хранения необходимых свойств, и заполняются значения по умолчанию. Во время уничтожения объекта происходит освобождение выделенной памяти. Метод для создания объекта называется конструктором (constructor). Метод для уничтожения объекта называется деструктором (destructor). Сам процесс создания объекта называется инициализацией.

Теперь рассмотрим использования нашей кнопки. Он выглядит следующим образом:

- Создание кнопки с помощью вызова конструктора;
- Изменение необходимых свойств;
- Все. Наша кнопка готова к работе.

Объект — сложный тип данных. Это значит, что вы можете объявлять переменные типа "объект" (точно так же, как объявлялись переменные типа "число" или "строка") и обращаться к объекту через эту переменную. На языке программирования это будет выглядеть немного сложнее:

- Объявить переменную типа "кнопка".
- В эту переменную нужно проинициализировать объект.
- Изменить нужные свойства.
- Можно использовать объект.

Доступ к свойствам и методам объектов осуществляется при помощи записи

ИмяОбъекта.Свойство

или

ИмяОбъекта.Метод

т.е. в записи имя объекта и его свойства или методы объекта разделяются точкой.

Давайте объявим переменную-объект типа **Кнопка**. Теперь можно создать кнопку, для чего есть конструктор (метод для создания объекта), который выделяет свободную память под этот объект. Процесс инициализации объекта-кнопки выглядит так: переменной **Объект** нужно присвоить результат работы конструктора объекта **Кнопка**. Конструктор выделит необходимую объекту память и присвоит свойствам значения по умолчанию. Результат этого действия будет присвоен переменной **Объект**. Эта переменная будет указывать на область памяти в которой находится созданная кнопка, и ее свойства. После всех этих действий мы можем получить доступ к созданному объекту через переменную **Объект**.

**Начало программы**

```
Переменные:
  Объект1: Кнопка;
Начало кода
  Объект1:= Кнопка.Создать объект;
  Объект1.Заголовок:='Привет';
  Объект1.Уничтожить объект.
Конец кода;
```

# Визуальная библиотека компонент Visual Component Library (VCL)

**Библиотека визуальных компонентов** содержит большое количество классов, предназначенных для быстрой разработки приложений. Библиотека написана на Object Pascal и непосредственно связана с интегрированной средой разработки приложений Delphi. Несмотря на название, в VCL содержатся главным образом невизуальные компоненты, однако имеются и визуальные, а также другие классы, начиная с абстрактного класса **TObject**. При этом все компоненты являются классами, но не все классы являются компонентами.

Все классы **VCL** расположены на определенном уровне иерархии и образуют дерево (иерархию) классов. Знание происхождения объекта оказывает значительную помощь при его изучении, т. к. потомок наследует все элементы объекта-родителя. Так, если свойство **caption** принадлежит классу **TControl**, то это свойство будет и у его потомков, например, у классов **TButton** и **TCheckBox**, и у компонентов— кнопки **Button** и флажка **CheckBox** соответственно.

# Компоненты, которые мы уже знаем

| | | | | |
|---|---|---|---|---|
| Класс | Объекты (компоненты) | Описание | Object inspector | Использование |
| TLabel; | Label1, Label2, Label2, | Label2: TLabel; | Label1.Caption = 'Значение z =' | Выводит текст в заданном месте формы |
| TEdit; | Edit1, Edit2, Edit3 | Edit1: TEdit; | Edit3.Text = ‘ ' | z:=StrToFloat(Edit1.Text) |
| TButton | Button1 | Button1:TButton | Button1.Caption = 'Счет' | Обработчик события процедура TForm1.Button1Click |
| TBitBtn | BitBtn1 | BitBtn1:TBitBtn | BitBtn1.Caption: =’&Закрыть’ | |
| TRadioGroup | RadioGroup1 | RadioGroup1:TRadioGroup | RadioGroup1.ItemIndex | Case RadioGroup1.ItemIndex of |

http://ge.tt/5mGYDhr

https://www.dropbox.com/sh/ex26edcc026h23c/AADkuFomJzyAYKceibd4o-5Aa?dl=0