Объектно-ориентированное программирование (ООП)

Языки и системы программирования

Основные парадигмы программирования:

- процедурное программирование (Fortran, Basic, Cobol, Algol, Pascal, Ada, C, Logo, FoxPro);
- объектно-ориентированное программирование (Simula, Smalltalk, Object Pascal, C++, Java, C#);
- визуально-событийное программирование (Visual Basic, Delphi, Visual C++, Visual Java, Visual FoxPro);
- функциональное программирование (Lisp);
- логическое программирование (Prolog).

Принципы процедурного программирования

Главный принцип *процедурного* программирования — основная программа выполняется строчка за строчкой, вызывая процедуры и функции, все ветвления выполняются с помощью условных операторов.

Этот подход хорошо подходит для простых задач, где последовательность операций известна заранее.

В некоторых задачах порядок действий определяется пользователем, другими программами или поступлением новых данных из другого источника (например, из сети), поэтому классическая схема не работает.

В таких задачах программа должна «включаться в работу» только тогда, когда получит условный сигнал, то есть произойдет некоторое **событие** (изменение состояния). При этом программа должна «знать», как реагировать на нужное ей событие, то есть должна иметь **обработчик этого события** — процедуру специальног

Методология объектно-ориентированного программирования

- Методология объектно-ориентированного программирования пришла на смену процедурной организации структуры программного кода, когда стало очевидно, что традиционные методы процедурного программирования не способны справиться ни с растущей сложностью программ и их разработки, ни с повышением их надежности.
- Во второй половине 80-х годов возникла настоятельная потребность в новой методологии программирования, которая бы позволила решить весь этот комплекс проблем. Такой методологией стало объектно-ориентированное программирование (ООП).

Определение

- **Объектно-ориентированное программирование** (ООП, Object-Oriented Programming) - совокупность принципов, технологий , а также инструментальных средств для создания программных систем на основе архитектуры взаимодействия **объектов**.

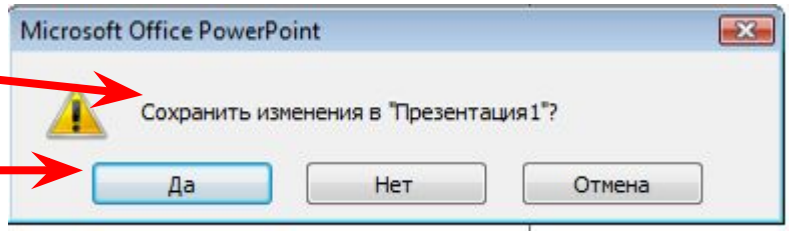
- Программа, построенная по принципам ООП, - это не последовательность операторов, не некий жесткий алгоритм, а совокупность объектов и способов их взаимодействия. Обмен информацией между объектами происходит посредством *сообщений*.

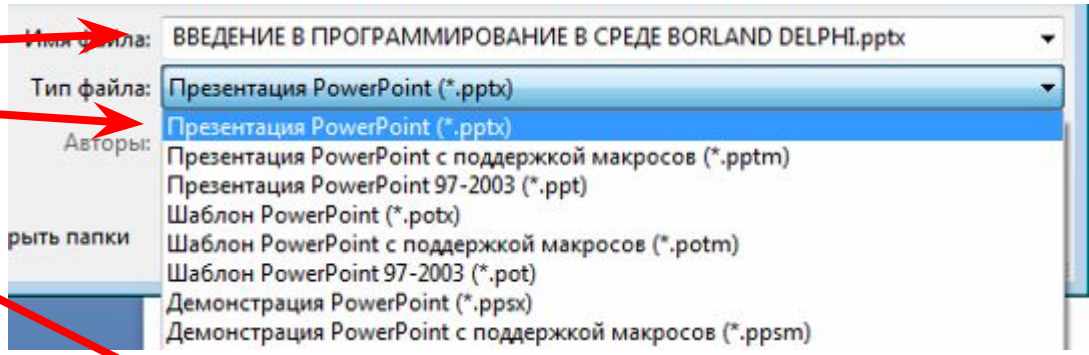
Объект

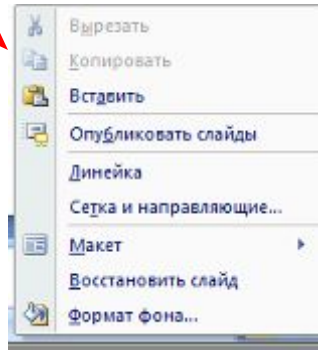
- **Объект** — понятие, абстракция или любой предмет с четко очерченными границами, имеющий смысл в контексте рассматриваемой прикладной проблемы.

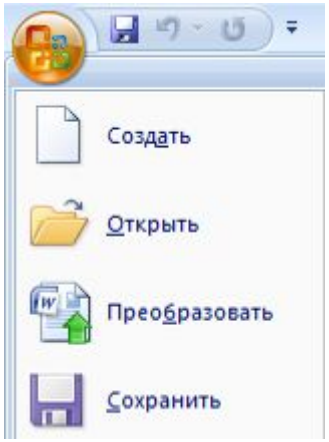
Объекты – это что?

- Это сообщения и кнопки
- Окна ввода и раскрывающиеся списки
- Главные меню и контекстные









- В каждый момент времени объект характеризуется присущим именно ему набором **свойств** (properties) и **методов** (methods) – операций, совершаемых над другими объектами или данным объектом, а также реагирует на **события** (events).

Свойства

- **Свойства** – перечень параметров объекта, которые определяют внешний вид и поведение объекта, выделяют уникальные особенности каждого экземпляра.
- К свойствам относятся: имя, тип, значение, цвет, размер и др.
- **Состояние** – совокупность всех свойств данного объекта.

Методы

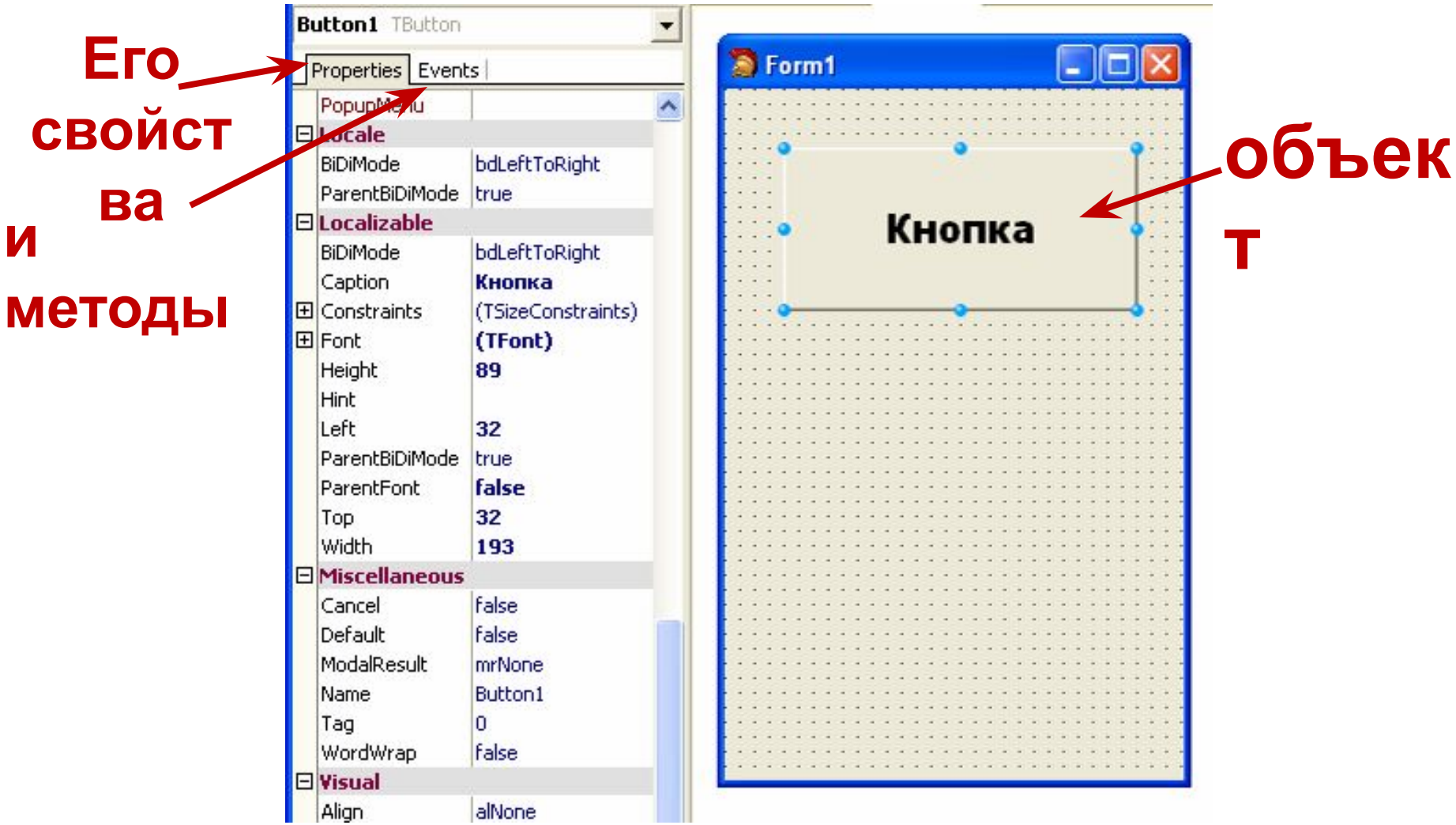
- **Метод** - это некоторое действие (операция), которое можно выполнять над данным объектом. В результате этого действия в объекте что-нибудь меняется (например, местоположение, цвет и др.).
- Другими словами можно еще сказать, методом называется команда, которую может выполнять объект.

События

- **События** – сигналы, формируемые внешней средой, на которые объект должен отреагировать соответствующим образом.
- Средой взаимодействия объектов являются *сообщения*, генерируемые в результате наступления различных *событий*.
- События наступают в результате действий пользователя – перемещение курсора пользователя, нажатия кнопок мыши или клавиш на клавиатуре, а также в результате работы самих объектов. Для каждого объекта определено множество событий, на которые он может реагировать.

- объекты – это «существительные», свойства объекта – «прилагательные», а методы объекта – это «глаголы».

Объекты – это что?



Объект является экземпляром того или иного класса.

- Ученик – экземпляр класса «Школьники»
- Ноутбук – экземпляр класса «Компьютеры»
- Земля – экземпляр класса «Планеты»

Классы объектов

- **Классом** называют особую структуру, которая может иметь в своем составе поля, методы и свойства.
- **Класс** представляет собой множество **объектов**
  - имеющих общую структуру
  - обладающих одинаковым поведением.
- Класс выступает в качестве объектного типа данных, а объект – это конкретный *экземпляр* класса.

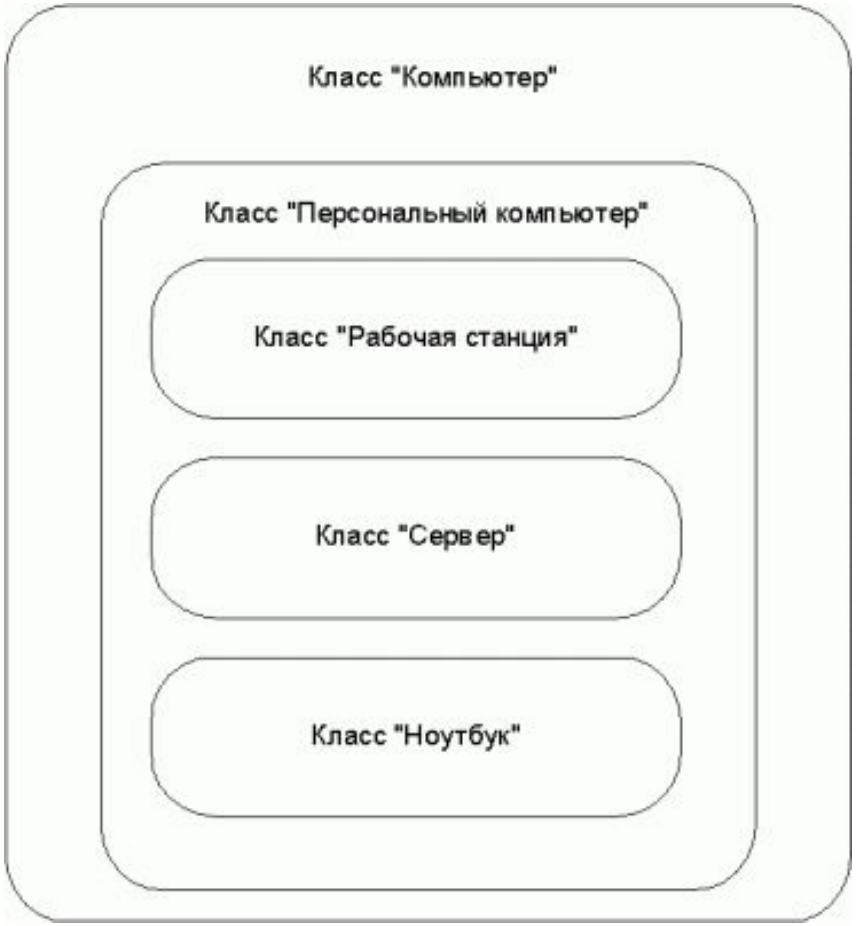
Классы объектов

- Каждый конкретный класс имеет свои особенности поведения и характеристики, определяющие этот класс. Например,

Наивысший уровень – самый общий и самый простой. Каждый последующий уровень более специфический и менее общий. На самом последнем уровне можно определить цвет, стиль заполнения, величину радиуса окружности и т.п.

Классы объектов

- Иерархия вложенности классов для примера общего класса "Компьютер"



Пример

Каждый из классов обладает специфическим набором свойств, методов и событий.

Например, в приложении Word существует класс объектов «Документ»(Document), который обладает определенными наборами:

- **Свойств**: имя (Name), полное имя (FullName) и так далее;
- **Методов**: открыть документ (Open), напечатать документ (PrintOut), сохранить документ (Save);
- **Событий**: создание документа (Document_New), закрытие документа (Document_Close) и т.д.

Объявление класса

Type
<имя класса> = Class(<имя класса - родителя>)
public   // т.е. доступно всем
<поля, методы, свойства, события>
published // т.е. видны в Инспекторе Объекта и изменяемы
<поля, свойства>
protected // доступно только потомкам
<поля, методы, свойства, события>
private// доступно только в этом модуле
<поля, методы, свойства, события>
end;

Создание класса

MyClass = class – создаем класс Myclass.

Name: string;

Color: string; – описываем параметры объекта, его имя и цвет. Практически задаем переменные.

function MyNewFunction(o: MyClass): string ;

– создаем метод используя один параметр передавая в него объект.

constructor Create (NewName: string; NewColor: string); – формируем конструктор и задаем начальные значения параметров объекта.

destructor Destroy; – разрушаем объект.

```
MyClass = class – создаем класс Myclass.
Name: string;
Color: string; – описываем параметры объекта, его имя и цвет. Практически задаем переменные.
function MyNewFunction(o: MyClass): string ;– создаем метод используя один параметр передавая
 в него объект.
constructor Create (NewName: string; NewColor: string); – формируем конструктор и задаем нача
льные значения параметров объекта.
destructor Destroy; – разрушаем объект.
```

Параметр Sender

- Параметр **Sender** в Delphi-программе присутствует в каждом обработчике событий любого компонента.

```
procedure TForm1.EditExit(Sender: TEdit);
Var
   S:string;
    i:integer;
begin
     S:=Sender.text;
     For i:=length(S) downto 1 do
       if not (S[i] in ['0'..'9']) then  //Удаляем НеЦифры
          Delete(S,i,1);
     Sender.text:=S;
end;
```

Теперь эту процедуру можно подставлять в обработчик OnExit для любого edit'a на форме.

Sender имеет тип TObject, и имеет значение объекта - источника события, в обработчике которого он используется. То есть, если на Форме находится несколько одинаковых компонентов, к тому же выполняющих одинаковые функции, то нет необходимости для каждого из них создавать свои процедуры-обработчики событий.

Принципы ООП:

- ✔ инкапсуляция
- ✔ наследование
- ✔ полиморфизм

- Мы ничего не знаем о том, как устроен объект внутри. Для нас это, как говорят кибернетики, «черный ящик».
- Для того, чтобы работать с объектом, не нужно знать его внутреннее устройство! Достаточно, что можно узнать и изменить его свойства, а также применять доступные методы управления.

Свойства и методы представляют собой **интерфейс** объекта, то есть способ его общения с внешним миром.

Инкапсуляция

- Пример: объект - телевизор.
- Внутри этого объекта находятся множество других объектов: микросхемы, провода, электронно-лучевая трубка и так далее. Но при взаимодействии с телевизором мы об этом даже и не задумываемся. В этом заключается первый принцип ООП - *инкапсуляция*.

Инкапсуляция

- **Инкапсуляция** (encapsulation) — это сокрытие реализации класса и отделение его внутреннего представления от внешнего (интерфейса).

Наследование

- Пример: цветной телевизор произошел от черно-белого, а телевизор с плазменным экраном - от обыкновенного.
- При этом каждый потомок наследовал свойства и функции предшественника, дополняя их своими, качественно новыми.
- Наследование позволяет расширить возможности объекта, не создавая при этом новый объект с нуля.

Наследование

- **Наследование** (inheritance) — это отношение между классами, при котором класс использует структуру или поведение другого класса (одиночное наследование), или других (множественное наследование) классов.
- Наследование вводит иерархию «общее/частное», в которой подкласс наследует от одного или нескольких более общих суперклассов.
- Подклассы обычно дополняют или переопределяют унаследованную структуру и поведение.

Наследование

Класс Точка (родитель)		Класс Окружность (наследник)	
Свойства	Методы	Свойства	Методы
Координаты (x,y)	Перемещение	Координаты центра (x, y)	Перемещение
Цвет	Изменение цвета	Цвет	Изменение цвета
		Радиус	Изменение радиуса

Классы-наследники могут наследовать характеристики классов-родителей. Т.е. один объект приобретает свойства другого объекта, добавляя к ним свойства, характерные только для него.

Наследование определяет отношение между классами: объекты класса-наследник обладают всеми свойствами и методами объектов класса-родитель и не должны их повторно реализовывать.

Полиморфизм

- Пример из реального мира:
- родственные объекты «птица», «рыба» и «животное» (базовый класс «живое существо») по-разному реализуют операцию «перемещение», летая, плавая и бегая соответственно.
- Метод «перемещение» потребует полиморфного объявления. Это позволит избежать ситуаций, когда вызов метода «перемещение» для объекта типа «рыба» приведет к тому, что объект реализует операцию «бежать».

Полиморфизм

- К объектам разных классов можно применять один и тот же метод, вот только действовать этот метод будет по-разному.
- Например, к большинству объектов в Windows&Office можно применять одни и те же методы: копирование, перемещение, переименование, удаление и т.п. Однако, механизмы реализации этих методов для разных классов неодинаковы.

Полиморфизм

- Слово *полиморфизм* происходит от греческих слов *poly* (много) и *morphos* (форма) и означает множественность форм.
- *Полиморфизм* – возможность использования одних и тех методов для объектов разных классов, только реализация этих методов будет *индивидуальной* для каждого класса.

Полиморфизм. Пример

Пусть у нас имеются некое обобщенное поле для хранения данных — класс TField и три его потомка — для хранения строк, целых и вещественных чисел:

```
type  TField = class
function GetData:string; virtual;
abstract;
end;
TStringField = class(TField)
FData : string;
function GetData: string; override;
end;
TIntegerField = class(TField)
FData : Integer;
function GetData: string; override;
  end;
TExtendedField = class(TField)
FData : Extended;
function GetData: string; override;
end;
```

```
function TStringField.GetData;
 begin
Result := FData
  end;
function TIntegerField.GetData;
begin
Result := IntToStr(FData);
end;
function TExtendedField.GetData;
begin
 Result:= FloatToStrF(FData, ffFixed, 7, 2);
end ;
procedure ShowData(AField : TField);
 begin
Forml.Label1.Caption := AField.GetData;
end;
```

- В базовом классе при помощи директивы *virtual* метод GetData объявлен виртуальным.
- Объявление метода виртуальным дает возможность дочернему классу произвести замену виртуального метода своим собственным.
- В каждом дочернем классе определен свой метод GetData, который замещает соответствующий метод родительского класса (метод порожденного класса, замещающий виртуальный метод родительского класса, помечается директивой override).

- классы содержат разнотипные поля данных FData и только-то и "умеют", что сообщить о значении этих данных текстовой строкой (при помощи метода GetData). Внешняя по отношению к ним процедура ShowData получает объект в виде параметра и показывает эту строку.
- В процедуре showData параметр описан как TFieid — это значит, что в нее можно передавать объекты классов и TStringField,и TIntegerField, и TExtendedField,и любого другого потомка класса TFieid.
- Но какой (точнее, чей) метод GetData при этом будет вызван? Тот, который соответствует классу фактически переданного объекта. Это иллюстрация принципа «полиморфизм».

Преимущества ООП:

(при создании больших программ):

- использование при программировании понятий, более близких к предметной области;
- локализация свойств и поведения объекта о одном месте, позволяющая лучше структурировать и, следовательно, отлаживать программу;
- возможность создания библиотеки объектов и создания программы из готовых частей;
- исключение избыточного кода за счет того, что можно многократно не описывать повторяющиеся действия;
- сравнительно простая возможность внесения изменений в программу без изменения уже написанных частей, а в ряде случаев и без их перекомпиляции.

Недостатки ООП:

- некоторое снижение быстродействия программы, связанное с использованием виртуальных методов (объявление метода виртуальным дает возможность дочернему классу произвести замену виртуального метода своим собственным);
- идеи ООП не просты для понимания и в особенности для практического использования;
- для эффективного использования существующих ОО систем требуется большой объем первоначальных знаний.