Объектно-ориентированное программирование

Выбор точки зрения – это первичный акт культуры.

Х. Ортега-и-Гассет

Компьютерные науки © М.Л. Цымблер

Содержание

- Основные концепции ООП
  - Объект
  - Инкапсуляция
  - Наследование
  - Полиморфизм
- Реализация концепций ООП в языке Pascal

ООП © М.Л. Цымблер 2

Основные концепции ООП

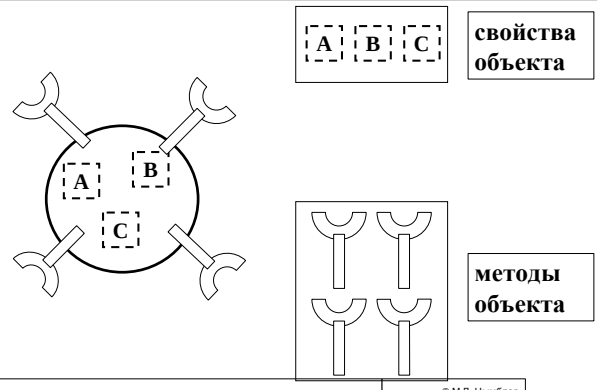
- *Объектно-ориентированное программирование (ООП)* – это методология программирования, основанная на следующих концепциях:
  - *объект*
  - *инкапсуляция*
  - *наследование*
  - *полиморфизм*.

ООП © М.Л. Цымблер 3

Объект

- *Объект* – объединение структуры данных и операций над этой структурой данных.
- Структура данных представляет *свойства* (атрибуты, поля) объекта.
- Операции представляют *методы* для анализа и изменения свойств объекта.

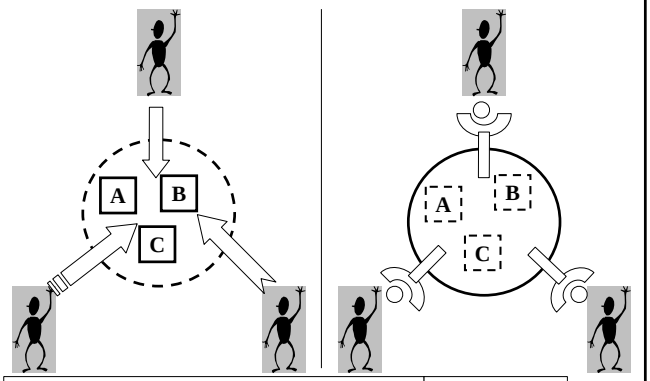
Объект



Инкапсуляция

- *Инкапсуляция* – объединение структуры данных и операций для работы с нею в форме единого объекта.

Инкапсуляция (до ООП и ООП)



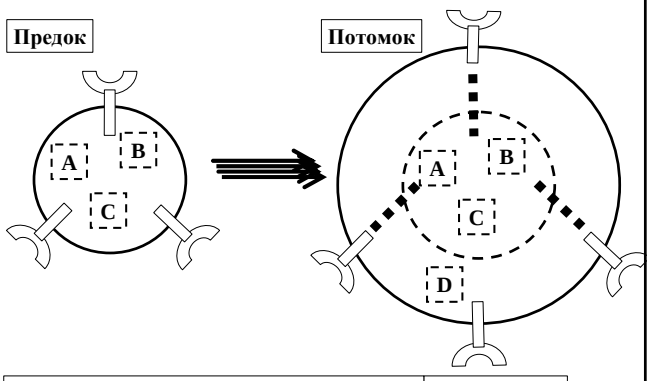
ООП © М.Л. Цымблер 7

Наследование

- *Наследование* – определение некоторого объекта-*предка* и использование его для построения иерархии объектов.
  При этом каждый объект-потомок *наследует* свойства и методы объекта-предка.

ООП © М.Л. Цымблер 8

Наследование



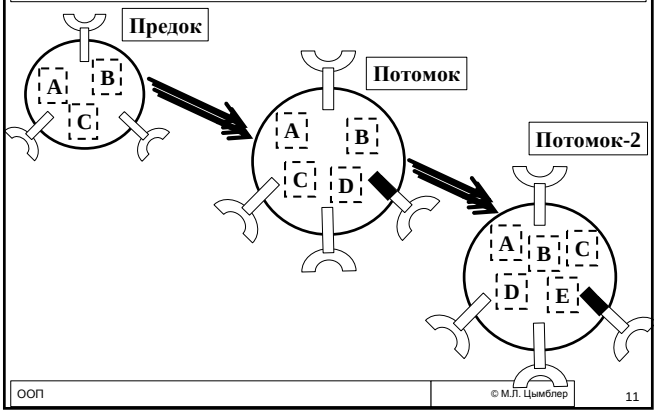
ООП © М.Л. Цымблер 9

Полиморфизм

- *Полиморфизм* – наделение некоторой операции именем, разделяемым вверх и вниз по иерархии объектов.
  При этом каждый объект в иерархии имеет свою реализацию операции с таким именем.

ООП © М.Л. Цымблер 10

Полиморфизм



ООП © М.Л. Цымблер 11

Описание объектного типа

```
type
    <имя> = object [(<имя объекта-предка>)]
        <описание полей объекта>
        <описание методов объекта>
    end;
<Реализация методов объекта>
```

ООП © М.Л. Цымблер 12

Пример: объектный тип

```
{ Описание полей и методов }
type
   TMan = object
      Name: String;
      Sex: Char;
      Age: Integer;
      procedure Init(aName: String; aSex: Char;
                     aAge: Integer);
   end;
```


Пример: объектный тип (2)

```
{ Реализация методов }
procedure TMan.Init(aName: String;
                    aSex: Char; aAge: Integer);
begin
   Name := aName;
   Sex := aSex;
   Age := aAge;
end;
```


Пример: объектный тип (3)

```
{ Создание и использование объектов }
var
   Ivanov, Petrova: TMan;
begin
   Ivanov.Init('Иванов И.И.', 'М', 30);
   Petrova.Init('Петрова И.П.', 'Ж', 20);
   WriteLn('Имя Пол Возраст');
   WriteLn(Ivanov.Name, Ivanov.Sex, Ivanov.Age);
   WriteLn(Petrova.Name, Petrova.Sex, Petrova.Age);
end.
```

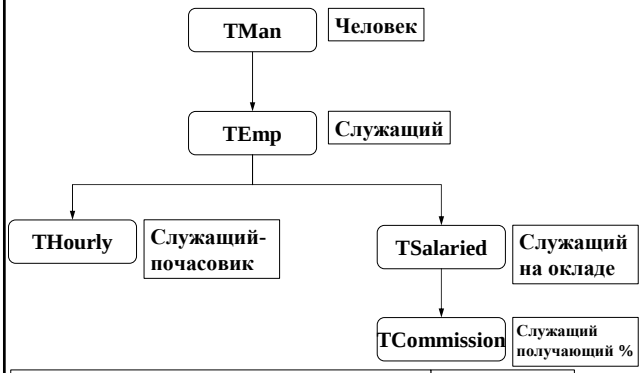
Инкапсуляция

```
type
    <имя> = object
    public
        <список общедоступных полей>
        <список общедоступных методов>
    private
        <список скрытых полей>
        <список скрытых методов>
    end;
```


Наследование

- *Наследование* позволяет на базе уже определенного объектного типа-*предка* создать объектный тип-*потомок*, который *наследует* все свойства и методы типа-предка.
  При этом в объектный тип-потомок могут быть добавлены новые свойства и методы.
- Потомок может осуществить *перекрытие* метода – переопределить один или более методов, унаследованных от предка.
  При этом перекрытый метод предка становится недоступным для потомка.

Пример: наследование



Пример: наследование (2)

```
type
  { Объектный тип «Служащий» }
  TEmp = object (TMan)
    public
      Title: String; { Должность }
      Rate: Real; { Ставка (оклад) }
      procedure Init(aName : String; aSex: Char;
            aAge: Integer; aTitle: String; aRate: Real);
    private
  end;
```


Пример: наследование (3)

```
{ Реализация методов объектного типа
«Служащий» }
procedure TEmp.Init(aName : String; aSex:
Char; aAge: Integer; aTitle: String; aRate: Real);
begin
   inherited Init(aName, aSex, aAge);
   Title := aTitle;
   Rate := aRate;
end;
```


Пример: наследование (4)

```
type
  { Объектный тип «Служащий-почасовик» }
  THourly = object (TEmp)
    public
      Time: Integer;
      procedure Init(aName : String; aSex: Char;
            aAge: Integer; aTitle: String; aRate: Real;
            aTime: Integer);
      function Salary: Real;
    private
      function OverTime: Integer;
  end;
```

Пример: наследование (5)

```
{ Реализация методов объектного типа
  «Служащий-почасовик» }
procedure THourly.Init(aName : String; aSex: Char;
                       aAge: Integer; aTitle: String;
                       aRate: Real; aTime: Integer);
begin
  inherited Init(aName, aSex, aAge, aTitle, aRate);
  Time := aTime;
end;
```


Пример: наследование (6)

```
{ Реализация методов объектного
  типа «Служащий-почасовик» }
function THourly.Salary: Integer;
begin
  if OverTime>0 then
    Salary := Rate*(MaxTime+
                    OverTime*Factor)
  else
    Salary := Rate*Time;
end;
```

```
const
  { Рабочих часов
    в месяце }
  MaxTime = 120;
  { Коэффициент
    для
    сверхурочных }
  Factor = 1.5;
```


Пример: наследование (7)

```
{ Реализация методов объектного типа
  «Служащий-почасовик» }
function THourly.OverTime: Integer;
begin
  if Time <= MaxTime then
    OverTime := 0
  else
    OverTime := Time - MaxTime;
end;
```

Пример: наследование (8)

```
type
  { Объектный тип «Служащий на окладе» }
  TSalaried = object (TEmp)
    public
       function Salary: Real;
    private
  end;
  function TSalaried.Salary: Real;
  begin
     Salary := Rate / MonthsInYear;
  end;
```


Пример: наследование (9)

```
type
  { Объектный тип «Служащий, получающий %» }
  TCommission = object (TSalaried)
    public
       Percent: Real;
       Sales: Real;
       procedure Init(aName : String; aSex: Char;
                aAge: Integer; aTitle: String;
                aRate, aPercent, aSales: Real);
       function Salary: Real;
    private
  end;
```

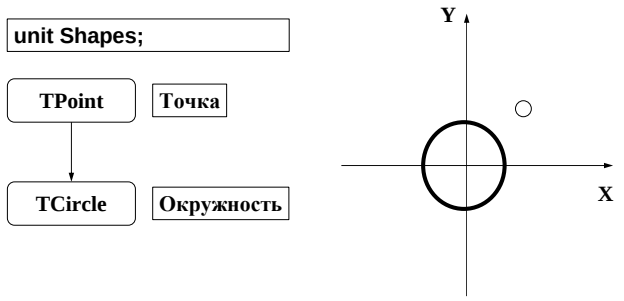

Пример: наследование (10)

```
{ Реализация методов объектного типаv «Служащий,
   получающий %» }
procedure TCommission.Init(aName : String; aSex: Char; aAge:
   Integer; aTitle: String; aRate, aPercent, aSales: Real);
begin
   inherited Init(aName, aSex, aAge, aTitle, aRate);
   Percent := aPercent;
   Sales := aSales;
end;
function TCommission.Salary: Real;
begin
   Salary := inherited Salary + Percent*Sales;
end;
```

Полиморфизм

- *Полиморфизм* – наделение некоторой операции именем, разделяемым вверх и вниз по иерархии объектных типов.
  При этом каждый объектный тип в иерархии имеет свою реализацию операции с таким именем.

Пример: полиморфизм



Пример: полиморфизм (2)

```
{ Объектный тип «Точка» }
TPoint = object
  private
    X: Real;
    Y: Real;
  public
    procedure Init(X, Y: Real);
    procedure Show; { Отобразить точку }
    procedure Hide;  { Спрятать точку }
    procedure MoveTo(NewX, NewY: Real);
                          { Переместить точку }
  end;
```

Пример: полиморфизм (3)

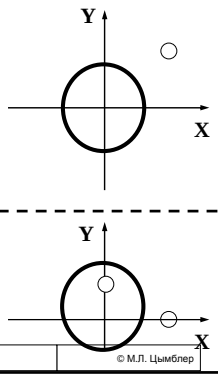
```
{ Объектный тип «Окружность» }
TCircle = object (TPoint)
   private
      Radius: Real;
   public
      procedure Init(X, Y, Radius: Real);
      procedure Show; { Отобразить окружность }
      procedure Hide;  { Спрятать окружность }
end;
```


Пример: полиморфизм (2)

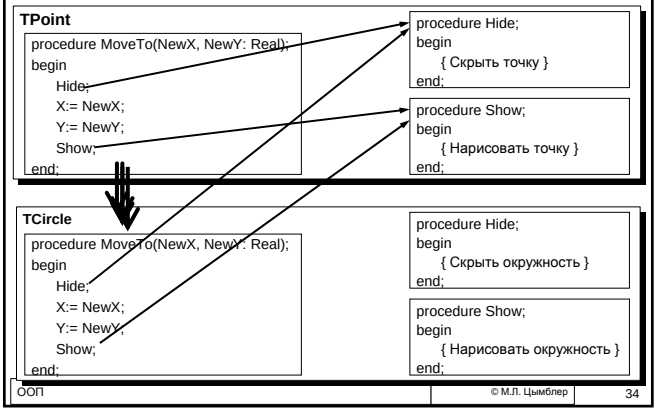
```
{ Реализация методов объектного типа «Точка» }
procedure TPoint.MoveTo(NewX, NewY: Real);
begin
   Hide;
   X:= NewX;
   Y:= NewY;
   Show;
end;
{ Объектный тип «Окружность» наследует метод
   MoveTo, не перекрывая его. }
```


Пример: полиморфизм (3)

```
uses Shapes;
var
   P: TPoint;
   C: TCircle;
begin
   P.Init(2,2); P.Show;
   C.Init(0,0,1); C.Show;
   P.MoveTo(2,0);
   C.MoveTo(0,1);
end.
```



Пример: полиморфизм (4)



ООП © М.Л. Цымблер 34

Раннее и позднее связывание

- Связывание экземпляров объектного типа с кодом методов *на этапе компиляции* программы называется *ранним*, а соответствующие методы объектного типа – *статическими*.
- Связывание экземпляров объектного типа с кодом методов *на этапе выполнения* программы называется *поздним*, а соответствующие методы объектного типа – *виртуальными*.

ООП © М.Л. Цымблер 35

Пример: виртуальные методы

```
{ Объектный тип «Точка» }
TPoint = object
  private
    X: Real;
    Y: Real;
  public
    constructor Init(X, Y: Real);
    procedure Show; virtual;
    procedure Hide; virtual;
    procedure MoveTo(NewX, NewY: Real);
  end;
```

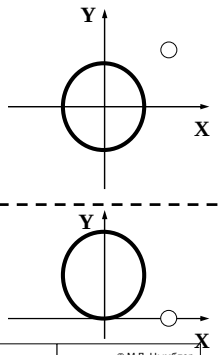
ООП © М.Л. Цымблер 36

Пример: виртуальные методы (2)

```
{ Объектный тип «Окружность» }
TCircle = object (TPoint)
  private
    Radius: Real;
  public
    constructor Init(X, Y, Radius: Real);
    procedure Show; virtual;
    procedure Hide; virtual;
end;
```


Пример: виртуальные методы (3)

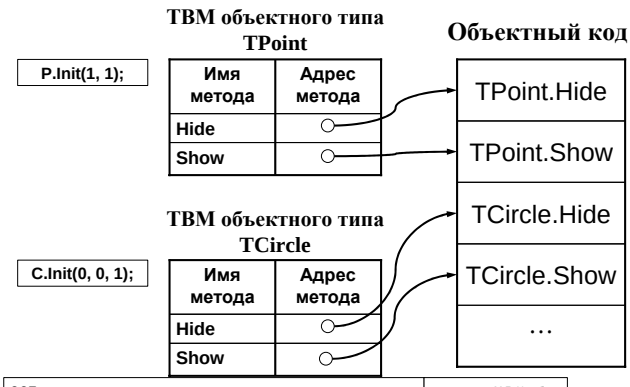
```
uses Shapes;
var
  P: TPoint;
  C: TCircle;
begin
  P.Init(2,2); P.Show;
  C.Init(0,0,1); C.Show;
  P.MoveTo(2,0);
  C.MoveTo(0,1);
end.
```



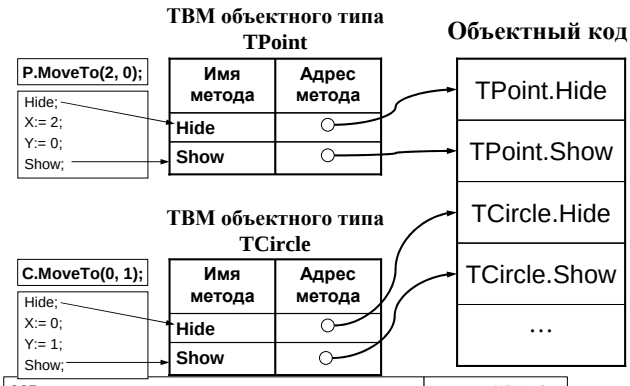
Механизм позднего связывания

- Объектный тип с виртуальными методами должен иметь хотя бы один специальный метод, называемый *конструктором*.
- Для каждого объектного типа в сегменте данных создается своя *таблица виртуальных методов (ТВМ)*.
  ТВМ содержит имена методов и адреса реализующих их подпрограмм. При вызове конструктора устанавливается связь между экземпляром объектного типа и ТВМ данного типа.

Пример: позднее связывание



Пример: позднее связывание (2)



Использование виртуальных методов

- Объектный тип с виртуальными методами должен иметь хотя бы один метод-конструктор.
- Вызов конструктора должен осуществляться до вызова любого виртуального метода.
- Виртуальный метод не может быть перекрыт статическим методом (и наоборот).
- Список формальных параметров виртуального метода должен совпадать со списком перекрываемого метода (статического – может не совпадать).

Динамические объекты

- Экземпляры объектного типа могут создаваться как динамические переменные. При этом используется стандартная процедура New с расширенным синтаксисом, где в качестве второго параметра указывается вызов соответствующего конструктора.
- Для уничтожения объектов и освобождения памяти используется стандартная процедура Dispose с расширенным синтаксисом, где в качестве второго параметра указывается вызов специального метода, называемого *деструктором*.
- Деструктор выполняет освобождение памяти. Деструктор может иметь пустое тело. Деструктор может быть виртуальным.

Пример: динамические объекты

```
unit Shapes;
interface
type
    TPoint=object
      private
          …
      public
          …
    destructor Done; virtual;
end;
implementation
destructor TPoint.Done; virtual;
begin
end;
…
end.
```

```
Program Animation;
uses Shapes;
var
   P: ^TPoint;
begin
   New(P, Init(2,2));
   P^.MoveTo(2,0);
   Dispose(P, Done);
end.
```


Вопросы для обсуждения

- Сколько конструкторов (деструкторов) может иметь объект?
- Может ли конструктор (деструктор) быть виртуальным?
- Почему есть ключевое слово virtual и нет ключевого слова static?
- Возможно ли ОО-программирование на языке без синтаксической поддержки ООП?