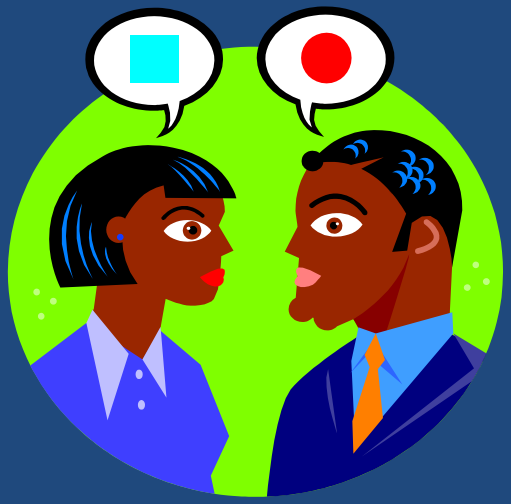


ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Выбор точки зрения есть первичный акт культуры.

Х. Ортега-и-Гассет

Содержание

- Парадигма в языке программирования
- Основные концепции ООП
- Реализация концепций ООП в языке Pascal

Парадигма

- *Парадигма* в науке
  - совокупность ценностей, методов, подходов, технических навыков и средств, принятых в научном сообществе в рамках устоявшейся научной традиции в определенный период времени.
  - способ постановки проблем и их решения.
- *Парадигма программирования* – совокупность идей и понятий, определяющая стиль написания программ.

Процедурное *vs* объектно-ориентированное программирование

- *Процедура* – синтаксически обособленная часть программы для решения подзадачи.
- *Программа*
  - переменные различных типов
  - вызовы процедур для обработки переменных
    - формальные/фактические параметры
    - передача по ссылке/значению.

- *Объект* – синтаксически обособленная часть программы для имитации сущности реального мира.
- *Программа*
  - объекты различных классов
  - вызовы методов объектов для обработки атрибутов объектов
    - сообщения
    - передача сообщений.

Основные концепции ООП

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

Абстракция: объект

- *Объект* – упрощенное, идеализированное описание реальной сущности предметной области.
- Неотъемлемые характеристики объекта
  - *состояние*: заданные значения *атрибутов (свойств)*
  - *поведение*: набор *операций* над атрибутами *(методы)*
  - *идентифицируемость*: атрибут(ы), однозначно отличающий данный объект от других объектов.

Человек
ФИО
Пол
…

Абстракция: класс

- *Класс* – тип данных, определяющий схожие объекты.
- Объект является *экземпляром* некоторого класса.

Человек
ФИО
Пол
…

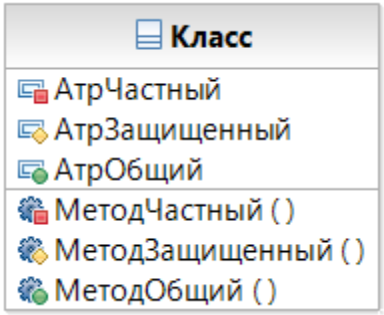
Человек
+ФИО +Пол +Дата рождения -Способности
+Работать() +Отдыхать() -Думать() +...()

Инкапсуляция

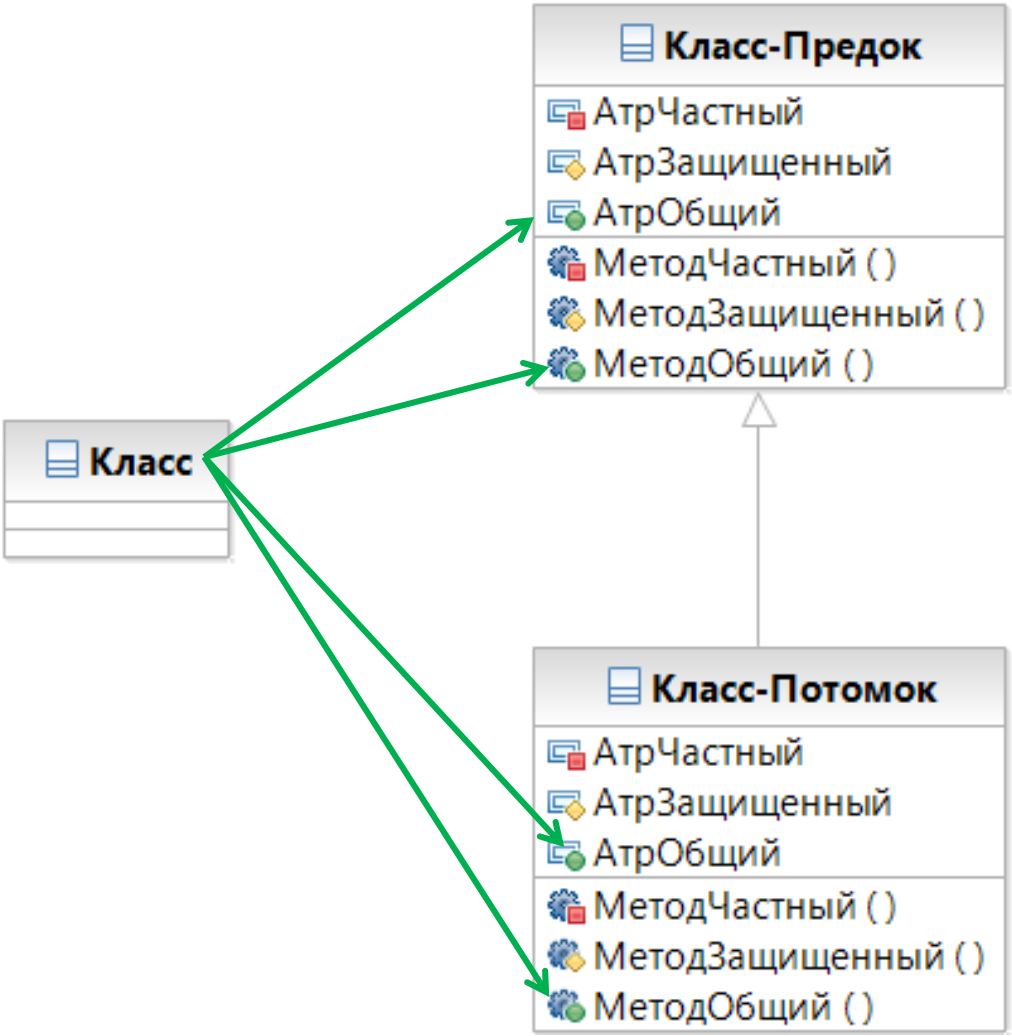
- *Инкапсуляция* – принцип, в соответствии с которым любой класс должен рассматриваться как черный ящик.
- Пользователь класса должен видеть и использовать только *интерфейс класса* (список свойств и методов класса) и не вникать в его внутреннюю реализацию.
  - Доступ к свойствам класса может быть организован не напрямую (квалификация ".", присваивание ":="), а с помощью методов (GetAttr, SetAttr).
  - Задание *области видимости* свойств и методов.

Область видимости атрибутов и методов

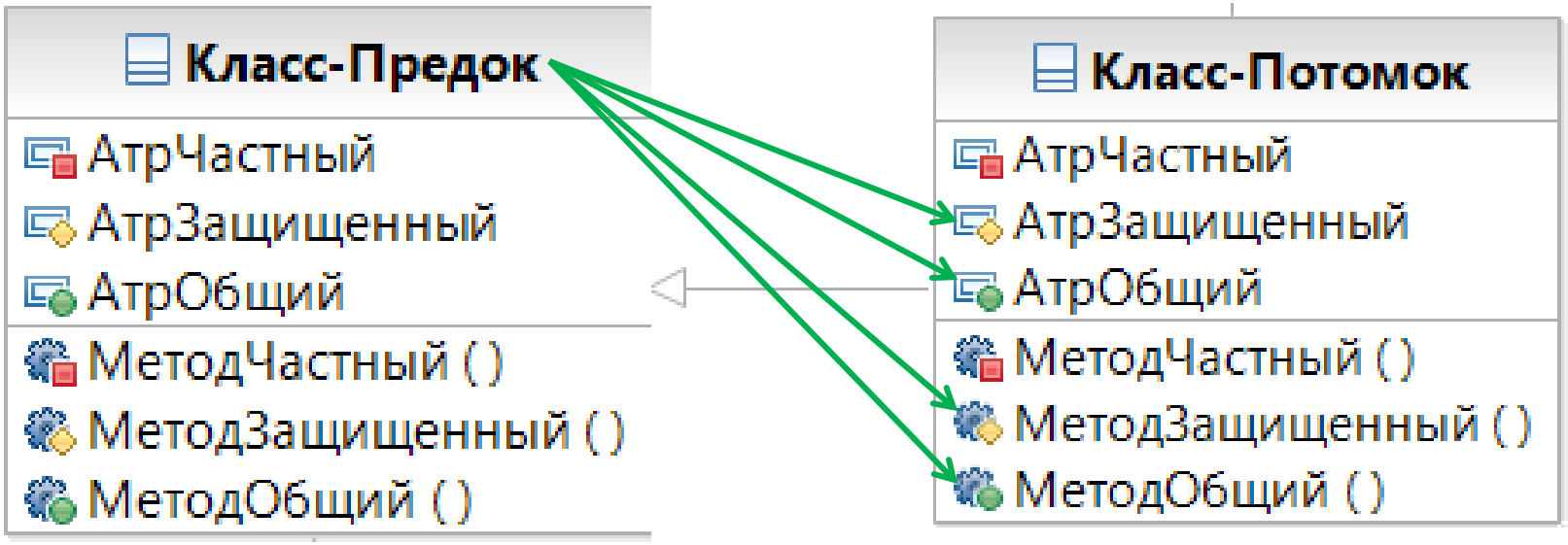
- Для атрибутов и методов класса может быть определена *область видимости* с помощью спецификаторов:
  - *private (частный)* – видны экземплярам данного класса
  - *protected (защищенный)* – видны экземплярам данного класса и классов-потомков этого класса
  - *public (общедоступный)* – видны экземплярам любых классов.



Область видимости



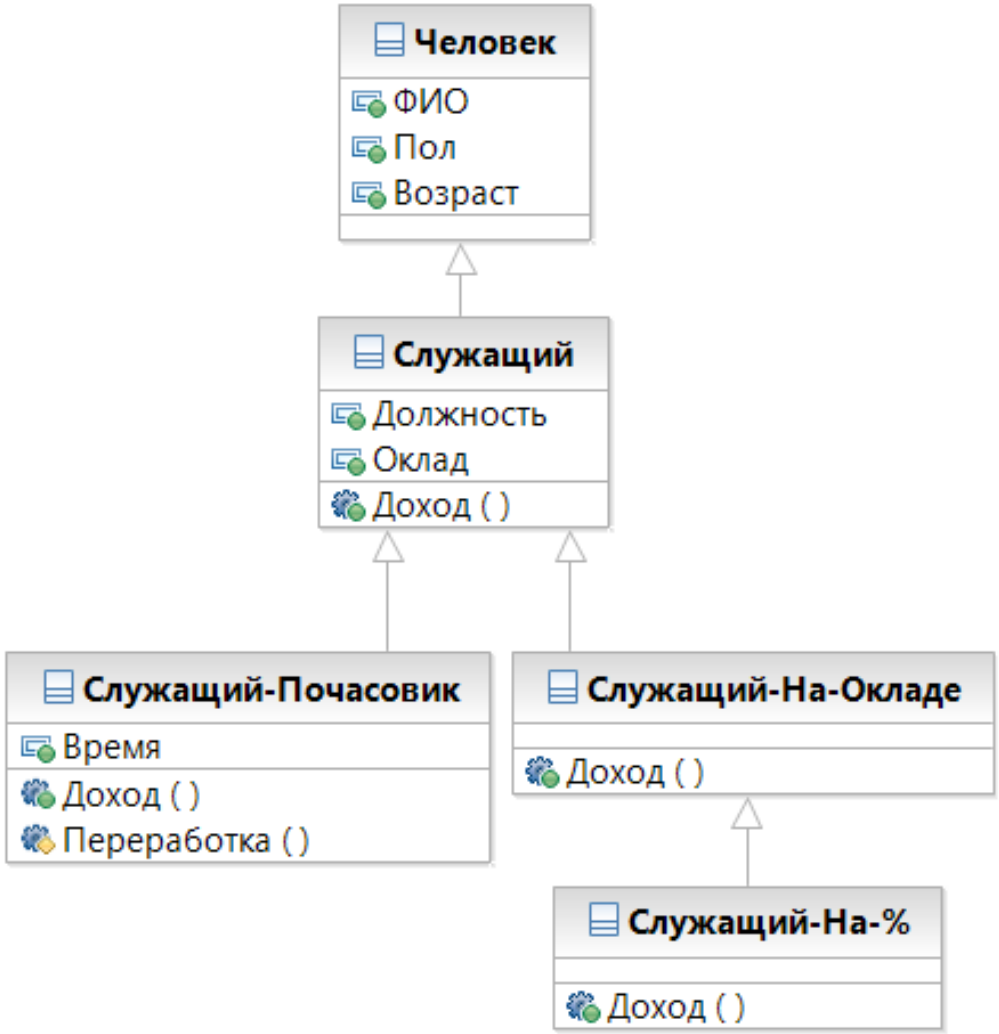
Область видимости



Наследование

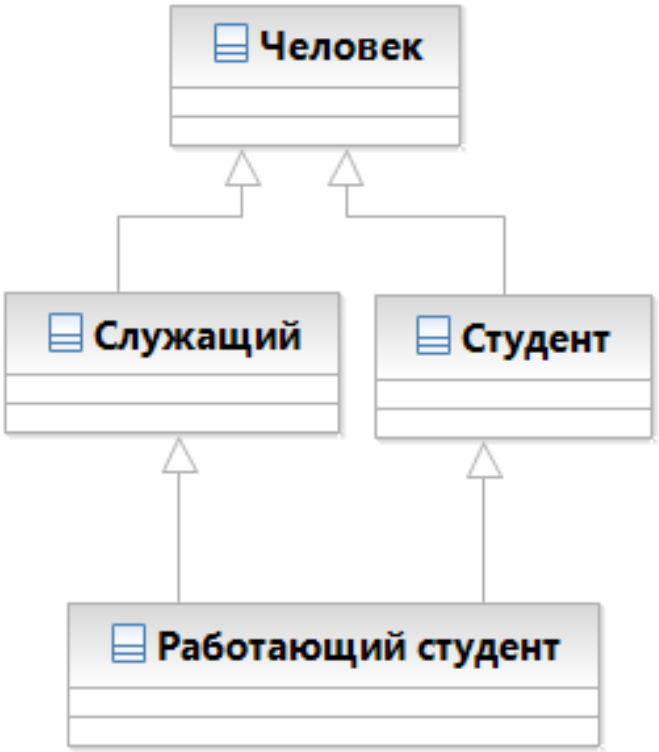
- *Наследование* – возможность порождать один класс от другого.
  - *Класс-потомок (подкласс)* сохраняет атрибуты и методы *класса-предка (суперкласса)*.
  - В подкласс могут быть добавлены *новые атрибуты и методы*.
  - В подклассе может быть *перекрыта* (изменена) реализация унаследованных методов.
- *Иерархия классов* – набор классов, связанных отношением наследования.

Наследование



Виды наследования

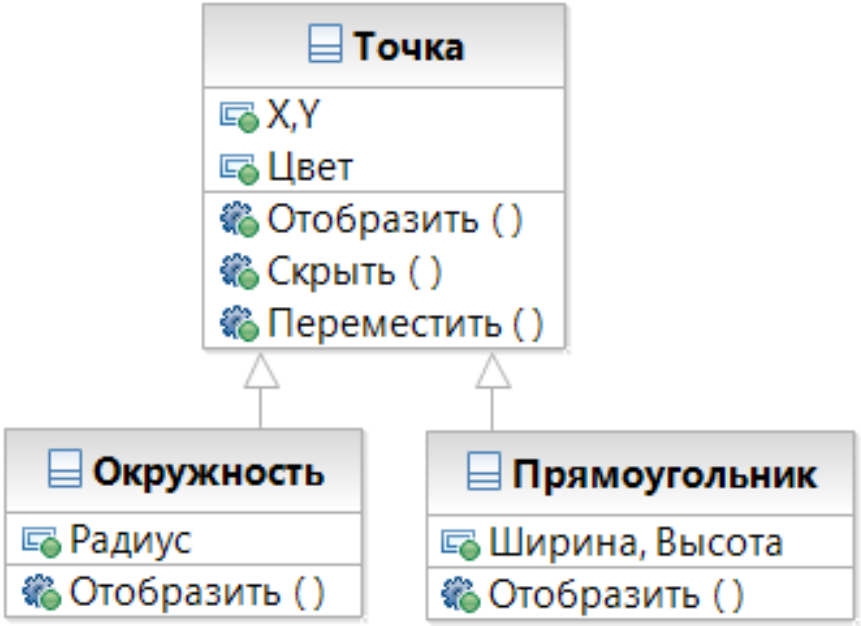
- *Простое наследование*: у класса не более одного класса-предка.
- *Множественное наследование*: у класса произвольное количество классов-предков.

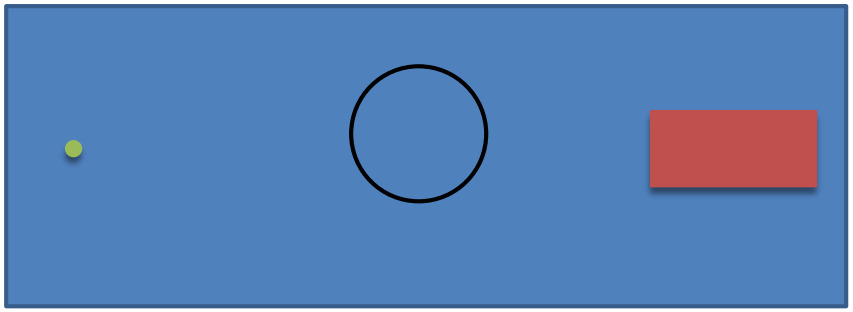


Полиморфизм

- *Полиморфизм* – наличие в иерархии классов метода с одинаковым именем, но разной у каждого класса реализацией: класс-потомок может изменить (*перекрыть*) реализацию метода, унаследованного от класса-предка.

Полиморфизм





- Метод *Скрыть*
  - Цвет:=ПолучитьЦветФона();
  - Отобразить();
- Метод *Переместить*
  - tmp:=Цвет;
  - Скрыть();
  - X:=НовX; Y:=НовY;
  - Цвет:=tmp;
  - Отобразить();
- Методы *Скрыть* и *Переместить* реализованы в классе Точка и наследуются классами-потомками.

Концепции ООП

- Абстракция
  - Упрощение
- Инкапсуляция
  - Скрытие
- Наследование
  - Повторное использование
- Полиморфизм
  - Гибкость

Описание класса


```
type
  <имя> = class [(<имя класса-предка>)]
    <описание атрибутов класса>
    <описание методов класса>
  end;
<Реализация методов класса>
```

Пример: абстракция (класс)


```
{ Определение }
type
  TMan=class
    Name: String;
    Sex: Char;
    Age: Integer;
    procedure Init(aName: String;
     aSex: Char;anAge: Integer);
  end;
```

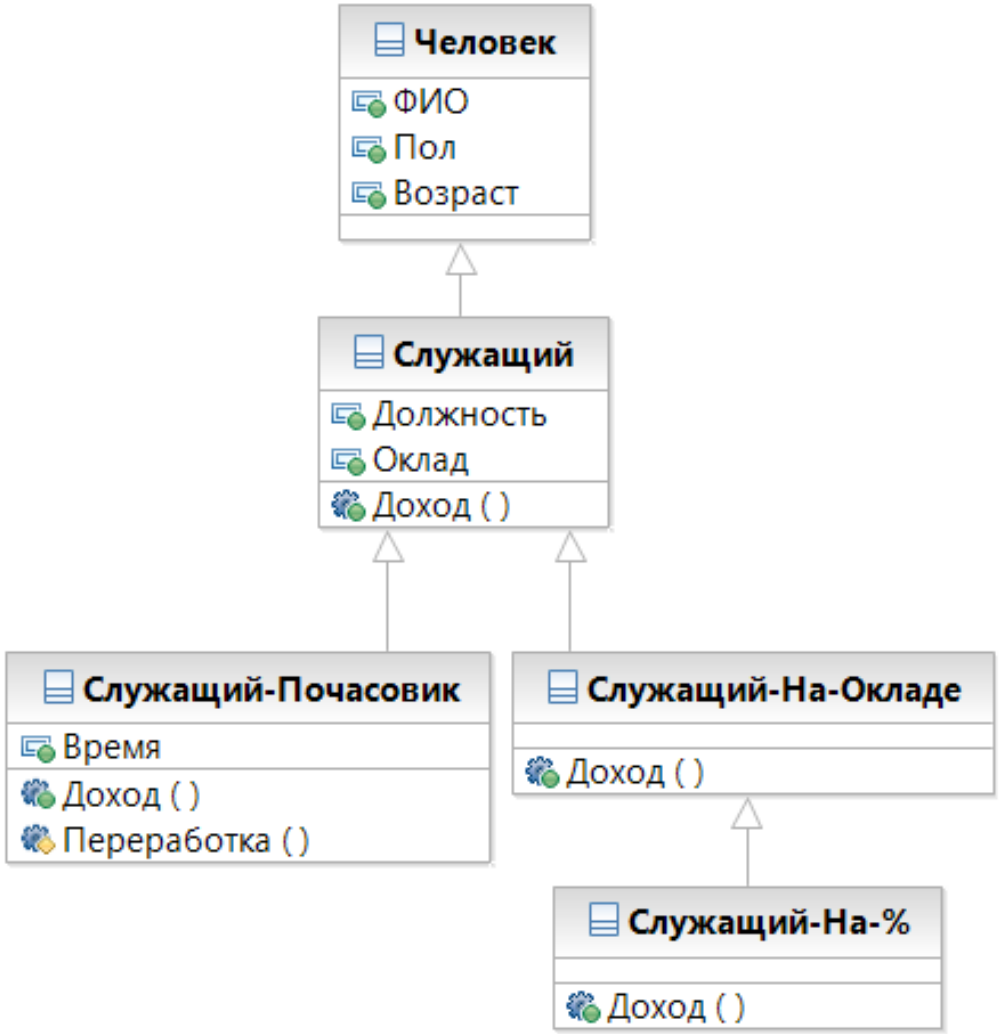
Пример: класс


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

Пример: класс


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

Пример: наследование



Пример: наследование


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

Пример: наследование


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

Пример: наследование (4)


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

Пример: наследование


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

Пример: наследование


```
{ Реализация методов класса
«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


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

Пример: наследование


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

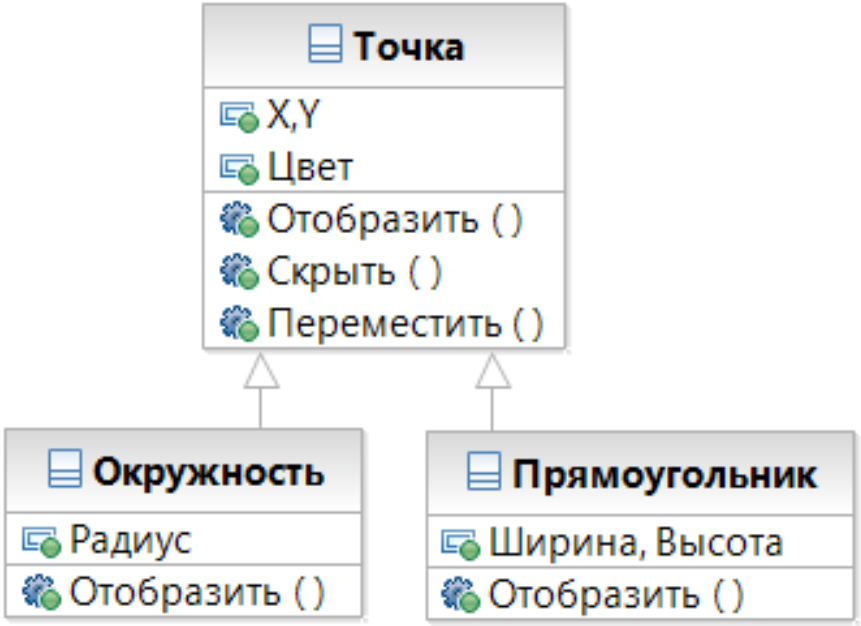
Пример: наследование

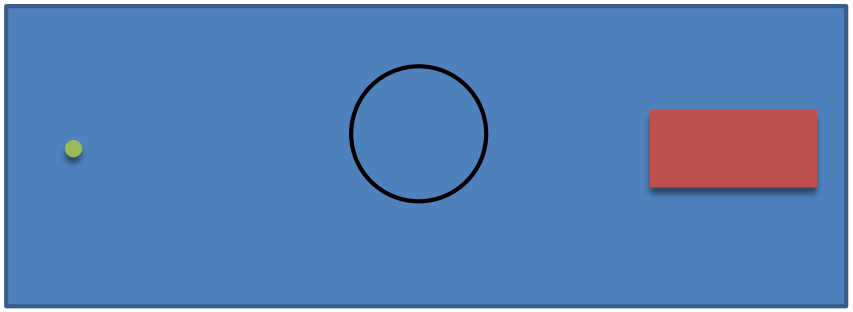

```
type
 { Класс «Служащий, получающий %» }
 TCommission = class(TSalaried)
  public
  Percent: Real;
  Sales: Real;
  procedure Init(aName : String; aSex: Char;
  anAge: Integer; aTitle: String;
  aRate, aPercent, aSales: Real);
  function Salary: Real;
  private
 end;
```

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;
```

Пример: полиморфизм





- Метод *Скрыть*
  - Цвет:=ПолучитьЦветФона();
  - Отобразить();
- Метод *Переместить*
  - tmp:=Цвет;
  - Скрыть();
  - X:=НовX; Y:=НовY;
  - Цвет:=tmp;
  - Отобразить();
- Методы *Скрыть* и *Переместить* реализованы в классе Точка и наследуются классами-потомками.

Пример: полиморфизм


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

Пример: полиморфизм

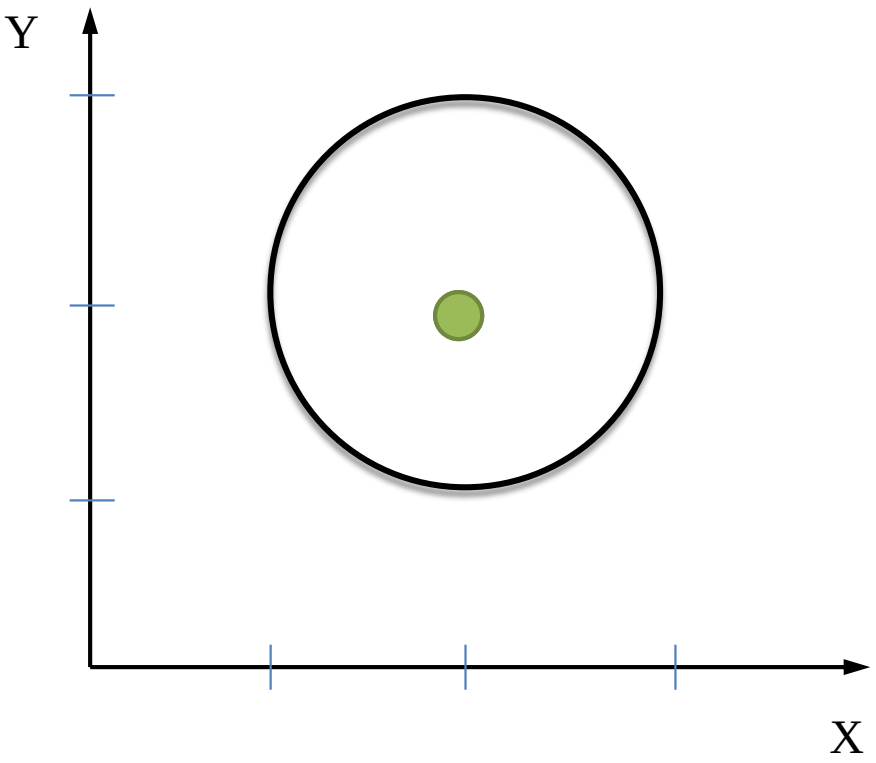

```
{ Класс «Окружность» }
TCircle = class (TPoint)
 private
  Radius: Integer;
 public
  procedure Init(X, Y, Radius: Integer);
  procedure Show; { Отобразить окружность }
end;
```

Пример: полиморфизм


```
{ Реализация методов класса «Точка» }
procedure TPoint.MoveTo(NewX, NewY: Integer);
var tmp: Integer;
begin
 tmp:=Color;
 Hide;
 X:= NewX;
 Y:= NewY;
 Color:=tmp;
 Show;
end;
{ Класс «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(2,2,1); C.Show;
 P.MoveTo(2,0);
 C.MoveTo(0,2);
end.
```



Пример: полиморфизм (4)

TPoint

```
procedure MoveTo(NewX, NewY: Integer);
begin
  Hide;
  X:= NewX;
  Y:= NewY;
  Show;
end;
```

```
procedure Hide;
begin
  { Скрыть точку }
end;
```

```
procedure Show;
begin
  { Нарисовать точку }
end;
```

TCircle

```
procedure MoveTo(NewX, NewY: Integer);
begin
  Hide;
  X:= NewX;
  Y:= NewY;
  Show;
end;
```

```
procedure Hide;
begin
  { Скрыть окружность }
end;
```

```
procedure Show;
begin
  { Нарисовать окружность }
end;
```

Раннее и позднее связывание

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

Пример: виртуальные методы

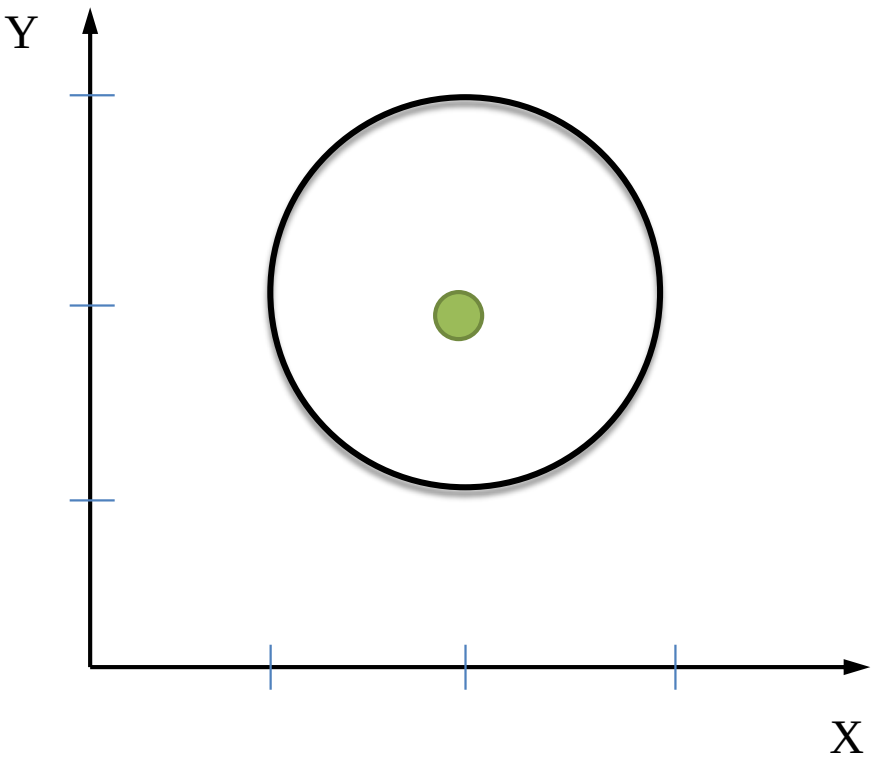

```
{ Класс «Точка» }
TPoint = class
 private
  X: Integer;
  Y: Integer;
 public
  constructor Init(X, Y: Integer);
  procedure Show; virtual;
  procedure Hide;
  procedure MoveTo(NewX, NewY: Integer);
 end;
```

Пример: виртуальные методы


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

Пример: полиморфизм

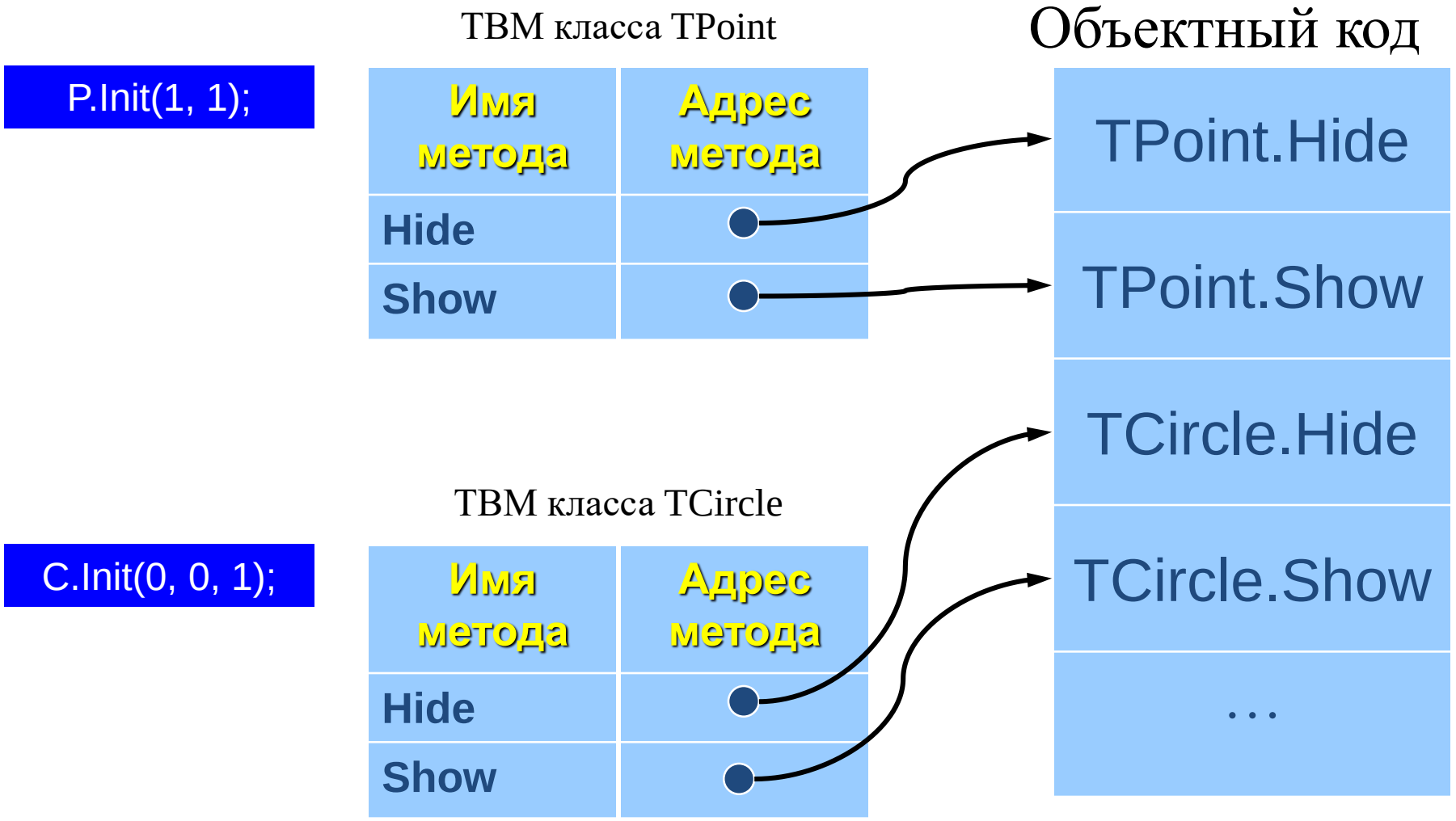

```
uses Shapes;
var
 P: TPoint;
 C: TCircle;
begin
 P.Init(2,2); P.Show;
 C.Init(2,2,1); C.Show;
 P.MoveTo(2,0);
 C.MoveTo(0,2);
end.
```



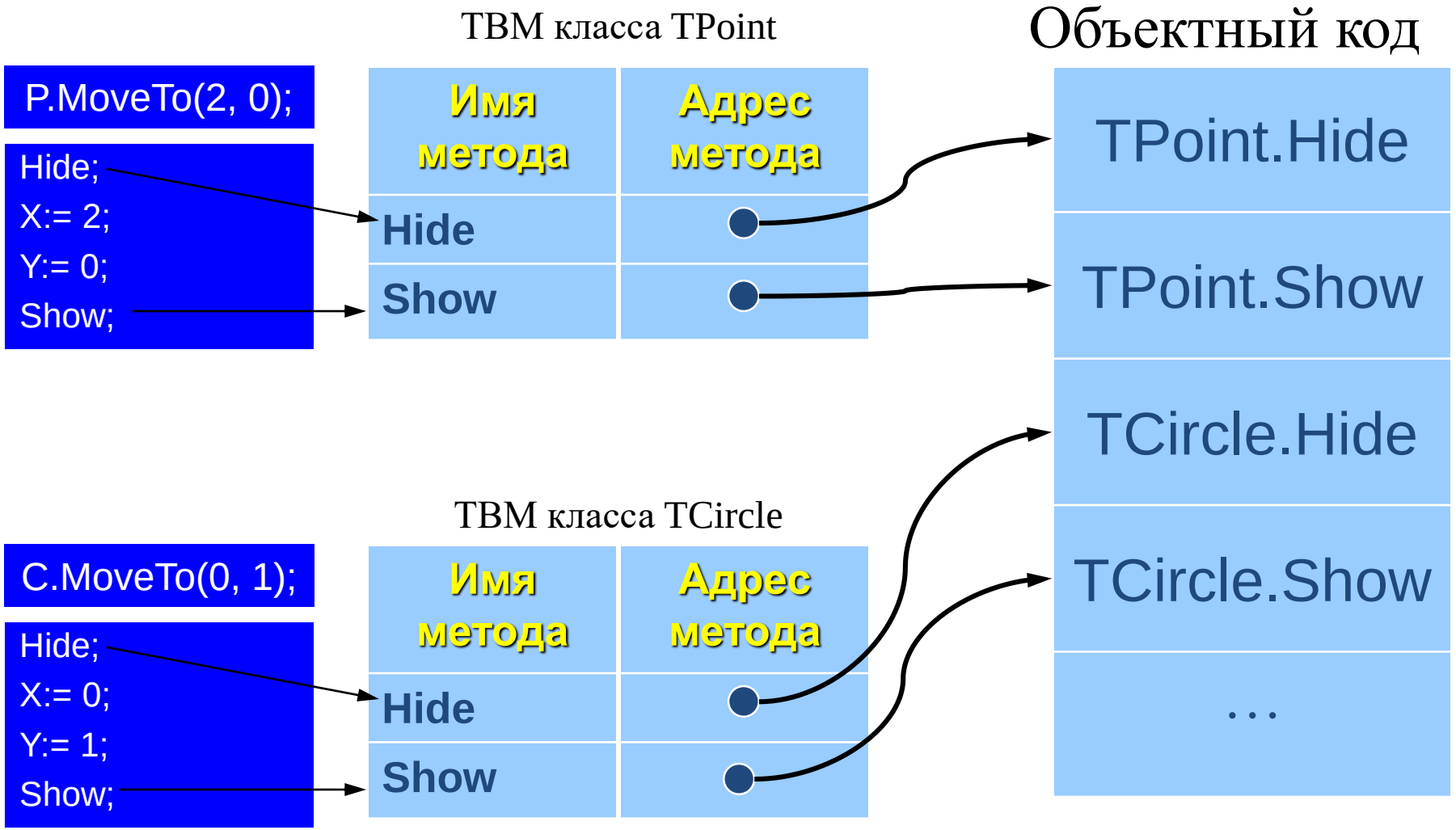
Механизм позднего связывания

- Класс с виртуальными методами должен иметь хотя бы один специальный метод, называемый *конструктором*.
- Для каждого класса 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 класса.

Пример: позднее связывание



Пример: позднее связывание (2)



Использование виртуальных методов

- Класс 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.
- Список формальных параметров виртуального метода должен совпадать со списком перекрываемого метода (статического – может не совпадать).

Заключение

- ООП – это методология программирования, основанная на концепции объекта.
- Основные свойства ООП – инкапсуляция, наследование, полиморфизм.
- ООП поддерживает два вида связывания объектов с кодом методов – раннее и позднее связывание; соответствующие методы называются статическими и виртуальными.