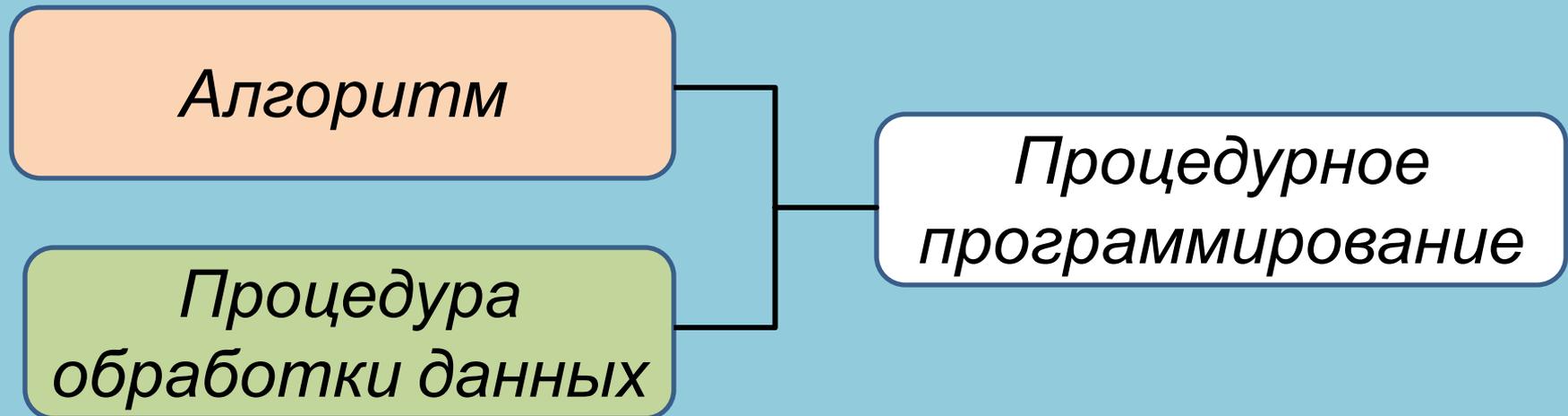


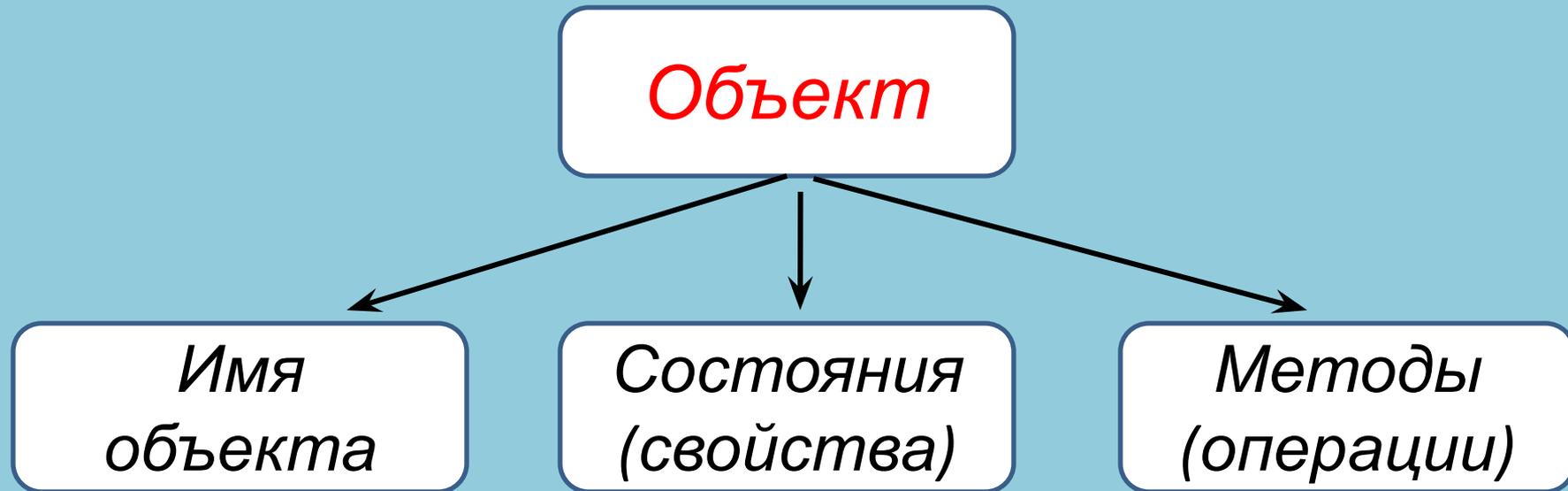
Введение
*в объектно-
ориентированное
программирование*

Процедурное программирование

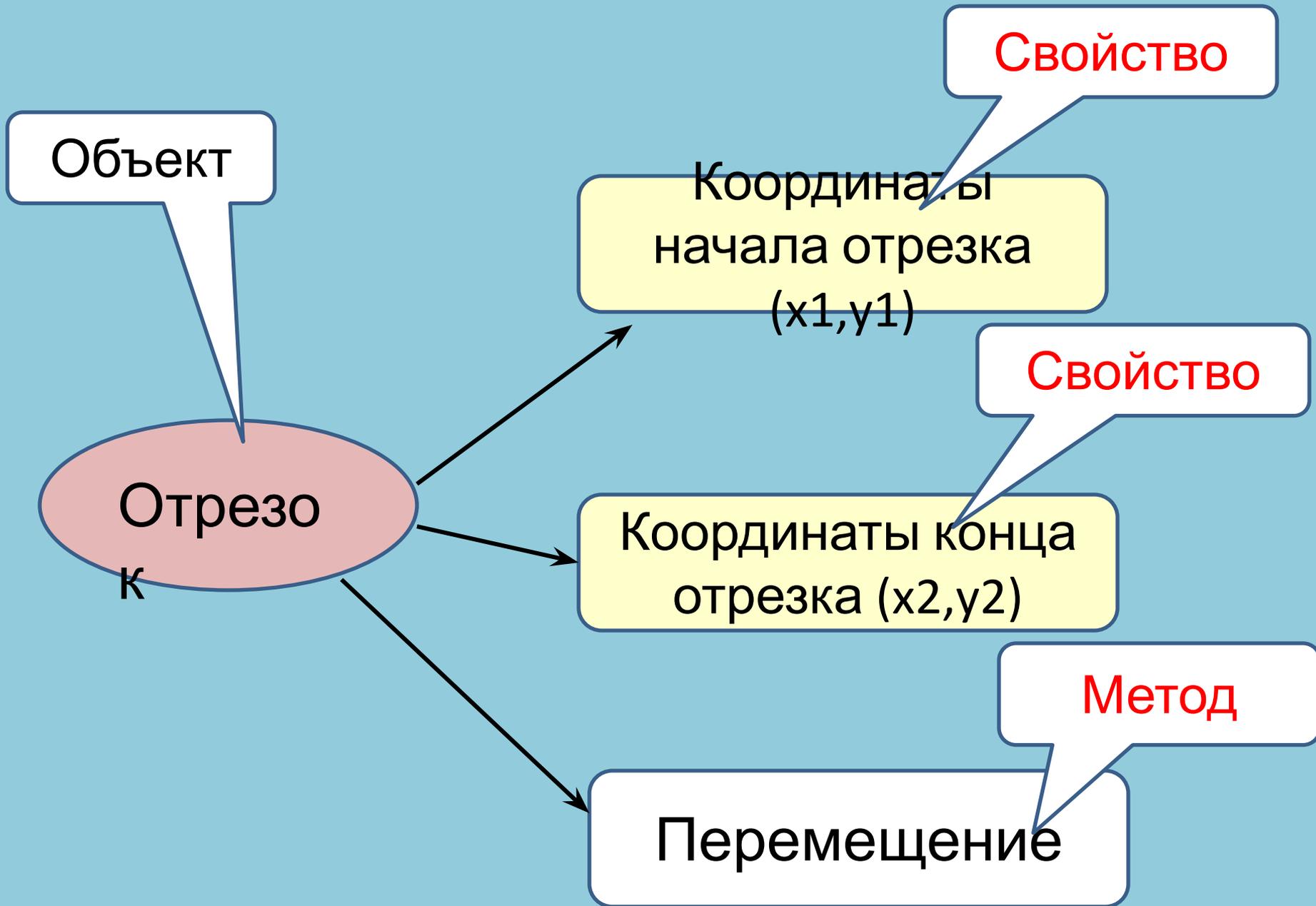
предполагает, что основой программы является *алгоритм и процедура обработки данных*.



Объектно-ориентированное программирование (ООП) – это методика разработки программ, в основе которой лежит понятие **объект**.



Объект характеризуется совокупностью всех своих свойств и их текущих значений, а также связанных с ними допустимыми для данного объекта действиями.



Объект

Свойство

Координаты
начала отрезка
(x1,y1)

Свойство

Отрезок

к

Координаты конца
отрезка (x2,y2)

Метод

Перемещение

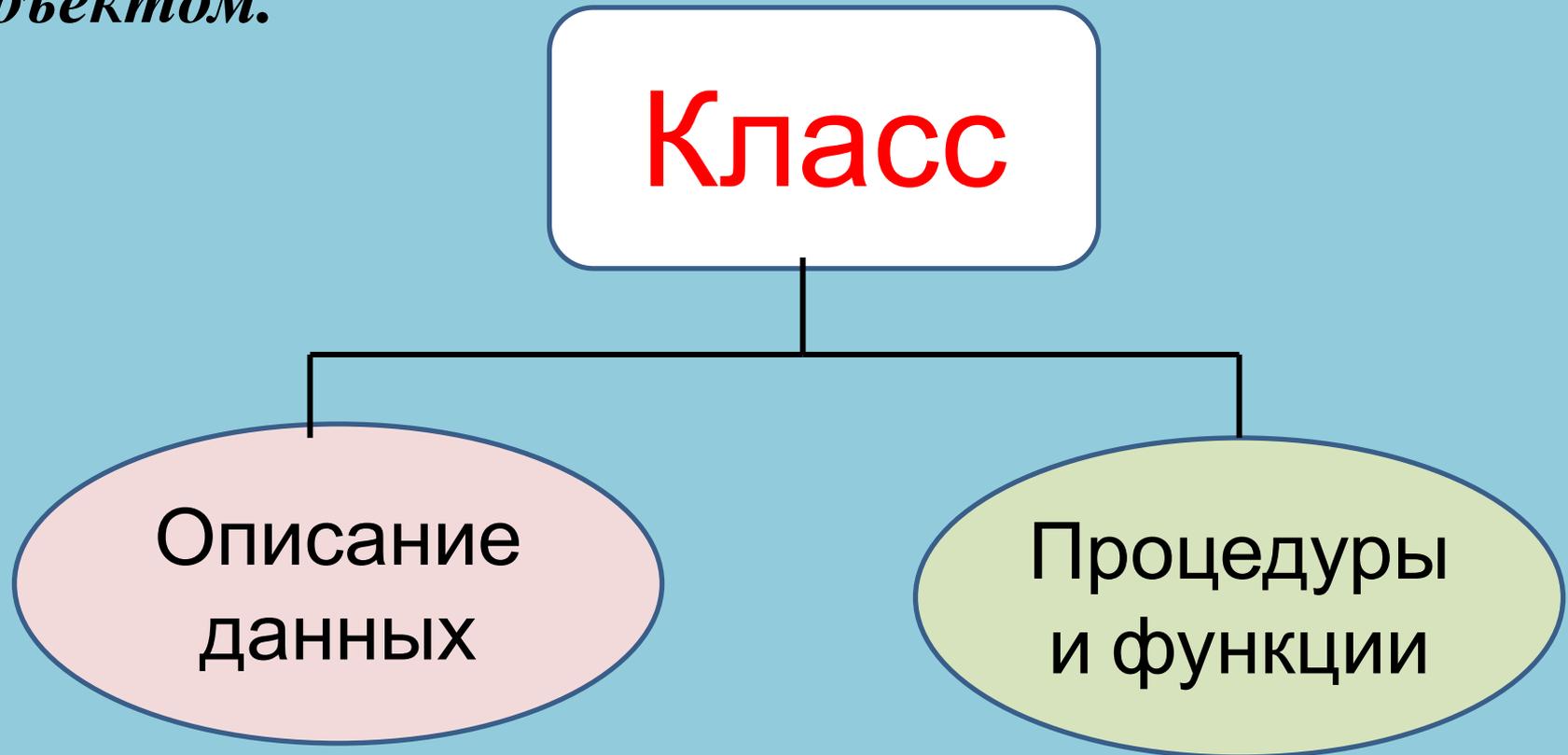
Свойство - определяющая характеристика объекта, которая влияет на то, как будет выглядеть компонент, а также на его поведение.

Событие – это то, что происходит в реальном времени и может вызывать те или иные ответные действия.

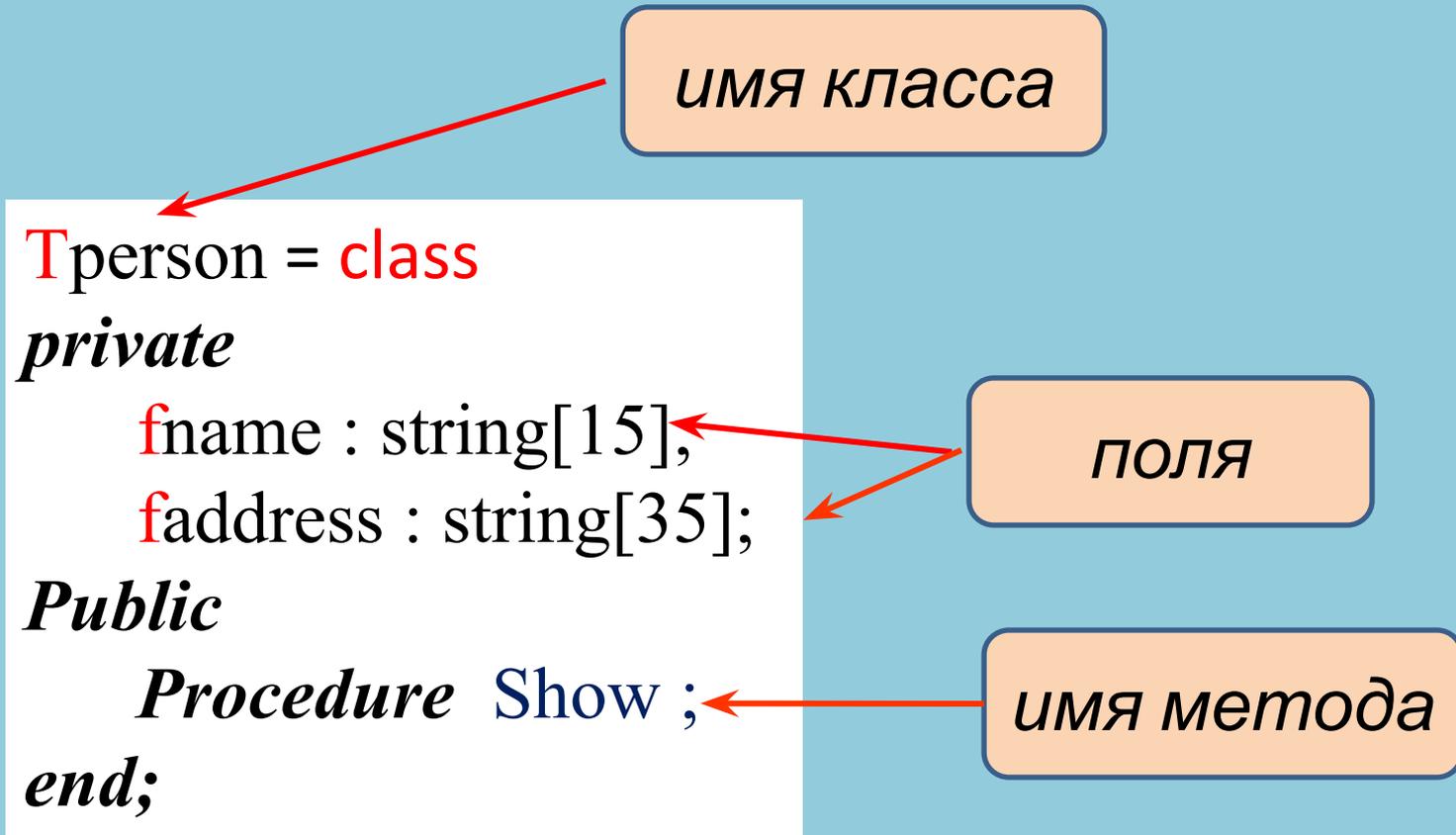
Методы – это способы, которыми объект может реагировать на те или иные действия.

Класс - это категория объектов или методов, обладающих одинаковыми свойствами и поведением.

Описание класса - включает, помимо описания данных, описания процедур и функций, которые могут выполняться над представителем класса – *объектом*.



Пример объявления простого класса



Имена полей должны начинаться с буквы **f** (field - поле)

Имя класса должно начинаться с

буквы **T**

ОБЪЕКТ

ОБЪЕКТЫ, как представители класса, объявляются в разделе `var`.

Например:

```
Var  
    student : Tperson;  
    professor : Tperson;
```

Конструктор

В Delphi объект – это динамическая структура. Переменная **объект** содержит не данные, а **ссылку** на данные объекта. Поэтому программист должен позаботиться о выделении памяти для этих данных. Выделение данных осуществляется с помощью специального метода класса – **конструктора**, которому присваивают имя **Create** (*создать*).

Описание класса с конструктором

```
Tperson = class
  private
    fname : string[15];
    faddress : string[35];
    constructor Create;
  Public
    Procedure Show ;
end;
```



КОНСТРУКТОР

Реализация конструктора

Помимо выделения памяти, конструктор решает задачу *присваивания полям объекта начальных значений*.

Пример реализации конструктора для объекта Tperson:

```
constructor Tperson.Create;  
begin  
    fname := ' '  
    faddress := ' '  
end;
```

МЕТОДЫ

Как ОБЪЕКТЫ, так и МЕТОДЫ подразделяются на классы.

МЕТОД - процедура, которая определена как часть класса и содержится в нем.

ООП характеризуется тремя основными свойствами

Наследование

Инкапсуляция

Полиморфизм

Инкапсуляция

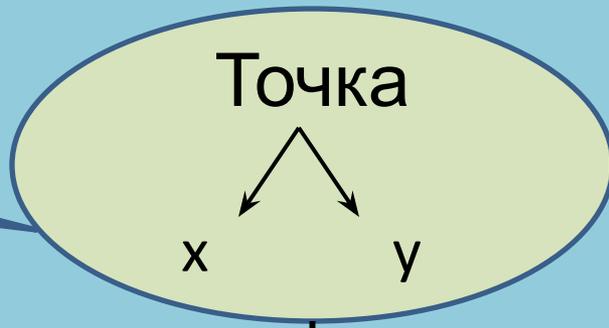
– это механизм, который *объединяет данные и методы в одном объекте* и защищает их от внешнего вмешательства и неправильного использования.

Применение данного принципа помогает локализовать возможные ошибки в коде программы

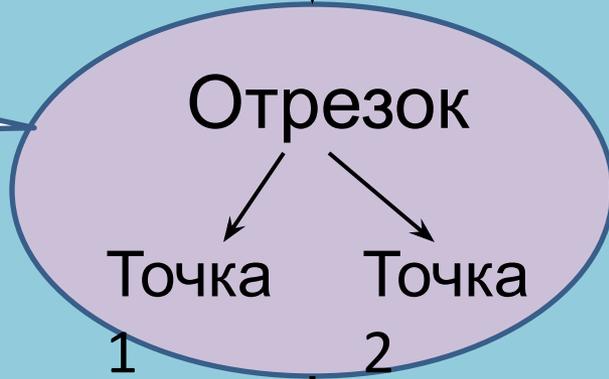
Наследование

– процесс, посредством которого один объект может **наследовать свойства другого объекта** и добавлять к ним черты, характерные только для него.

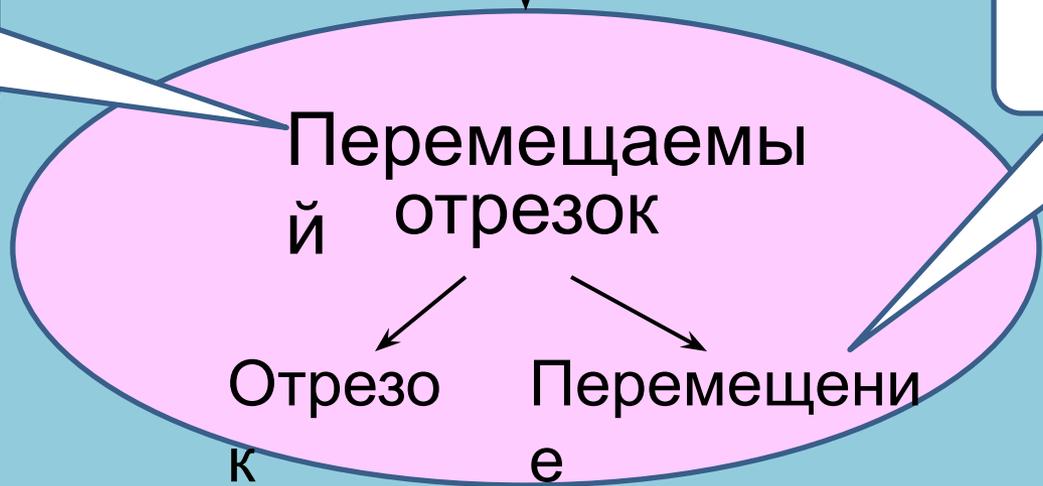
*Предо
к*



*Потомо
к точки*



*Потомо
к
отрезка*



Метод

Смысл и универсальность наследования

заключается в том, что *не надо каждый раз заново описывать новый объект*, а можно указать «родителя» (базовый класс) и описать ***отличительные особенности*** нового класса. В результате *новый объект будет обладать всеми свойствами родительского класса плюс своими собственными отличительными особенностями.*

Наследование относится только к *типам*, но не экземплярам объекта.

Описание типа потомка:

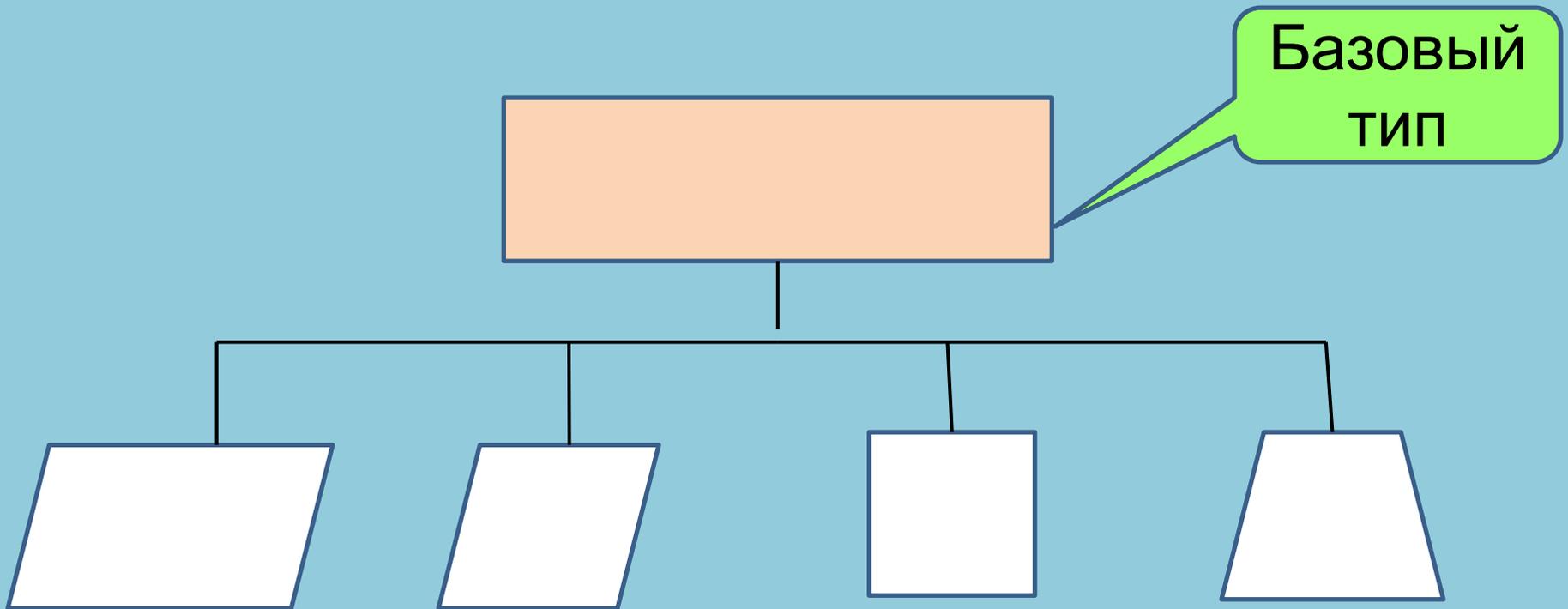
```
<имя типа-потомка> = Object (<имя типа-предка>);
```

Например, родительский объектный тип - «выпуклый четырехугольник» (поля типа «координаты вершин») и типы, им порожденные: параллелограмм, ромб, квадрат.

```
FourAngle=Object x1,x2,x3,x4 : BaseType;  
Parall = Object(FourAngle);
```

Пример

Пусть имеется родительский объектный тип «выпуклый четырехугольник» (заданный координатами вершин: $x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4$) и типы, им порожденные: параллелограмм, ромб, квадрат, трапеция.



Описание объекта в Турбо

Паскале

Для описания объекта служит слово *Object*.

Тип *Object* - это структура, которая содержит поля и методы.

```
Туре <идентификатор типа объекта> = Object;
```

```
  <поле>;
```

```
  ....
```

```
  <поле>;
```

```
  <метод>;
```

```
  ....
```

```
  <метод>;
```

```
End;
```

Под *методами объекта* понимают *процедуры* и *функции*, объявление которых включены в описание объекта и которые выполняют действия.

Procedure 1

Procedure 2

Function 1

Procedure 3

Пример описания

```
Type Natur = 1 .. 32767;  
Frac = Record  
    P : Integer;    // числитель  
    Q : Natur       // знаменатель  
End;  
Drob = Object A : Frac;  
    Procedure NOD (Var C:Natur);  
    Procedure Sokr;  
    Procedure Stepen (N : Natur; Var C : Frac);  
End;
```

Описание объекта «обыкновенная дробь» с методами «НОД числителя и знаменателя», «Сокращение» и

«Построение десятичной дроби».

Если к описанию объекта «Дробь» добавить методы ввода и вывода дроби:

```
Procedure Vvod;  
Procedure Print;
```

то основная программа будет выглядеть

```
Var Z: Drob; F: Frac;  
Begin  
  Z . Vvod;           // ввод дроби  
  Z . Print;         // печать введенной дроби  
  Z . Sokr;          //сокращение введенной дроби  
  Z . Print;         //печать дроби после сокращения  
  Z . Stepen (4, F); //возведение дроби в 4-ю степень  
  WriteLn(F . P, '/' F . Q)  
End.
```

Комментарии к примеру:

- Реализация методов осуществляется в разделе описаний, после объявления объекта, причем при реализации метода достаточно указать его заголовки без списка параметров, но с указанием объектного типа, методом которого он является.

Например:

Procedure Drob . Stepen;

- Все действия над объектом выполняются только с помощью его методов.
- Для работы с отдельным экземпляром переменных должна быть объявлена переменная соответствующего типа.

В нашем примере это переменная `Z: Drob;`

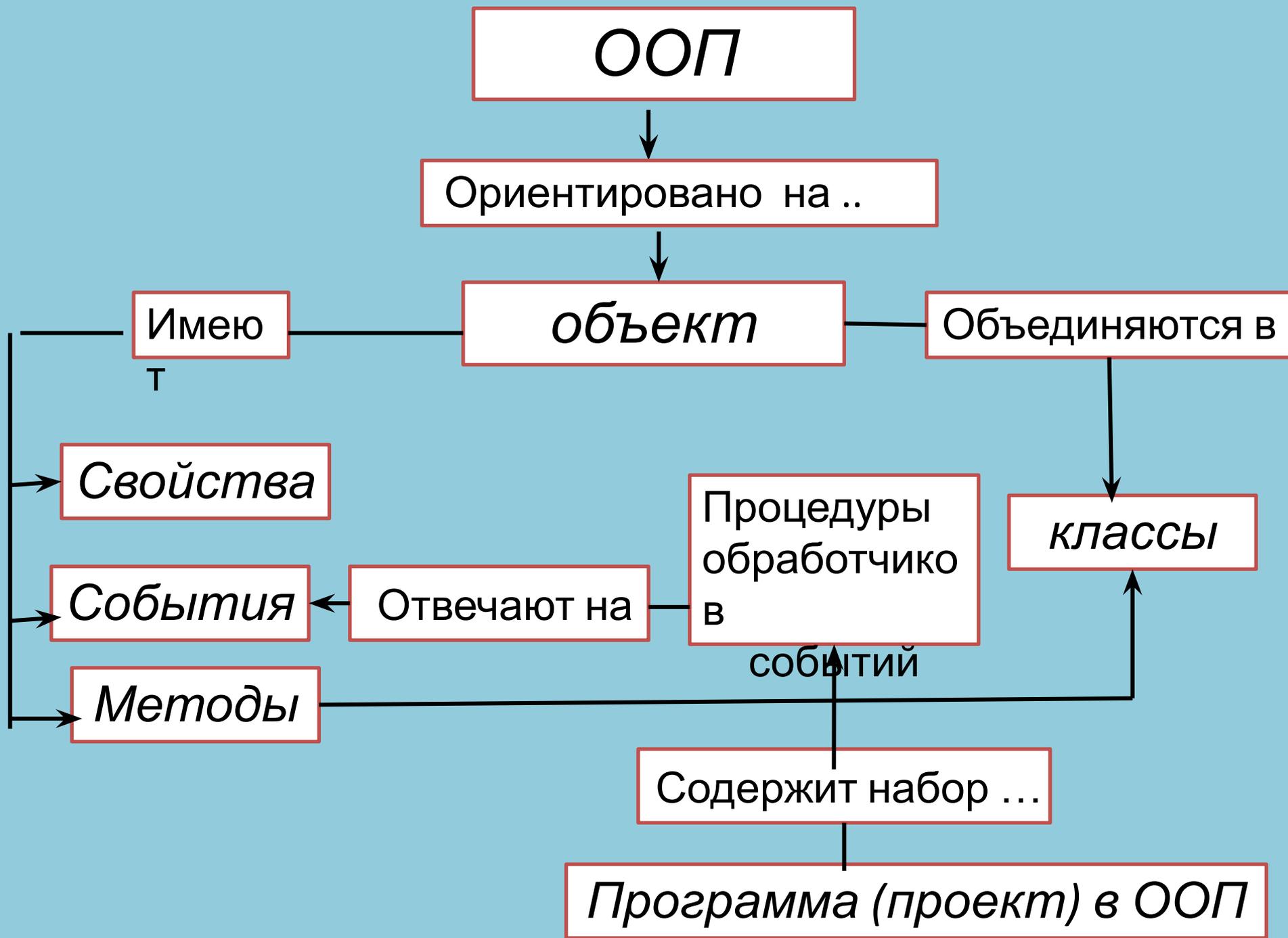
Полиморфизм –

свойство, которое позволяет метод с одним именем применять к различным родственным объектам.

Идея полиморфизма – «**один интерфейс – множество методов**»

Примером *Полиморфизма* может служить перемещение по экрану геометрической фигуры. Но для каждого конкретного объекта составляется подпрограмма, выполняющая это действие непосредственно для данного объекта, причем **все эти подпрограммы могут иметь одно и то же имя.** Когда потребуются перемещать конкретную фигуру, будет выбрана из всего класса соответствующая

подпрограмма



Выводы

1. Использование ООП к небольшим программам неэффективно.

2. Но при создании больших программ имеет ряд преимуществ:

- Использование более естественных понятий, простота введения новых понятий.

- Некоторое сокращение размеров программ за счет того, что *наследуемые свойства* можно многократно не описывать.

- Возможность создания библиотеки объектов.

- Возможность внесение изменений в программу без изменения уже написанных частей

□ Возможность написания подпрограмм с различными наборами формальных параметров, но имеющих одно и то же имя, используя свойство *полиморфизма*.

□ Более четкая локализация свойств и поведения объектов в одном и том же месте (свойство *инкапсуляции*), позволяющей проще разобраться со структурой программы и

□ Возможность разделения доступа к различным объектам программы.