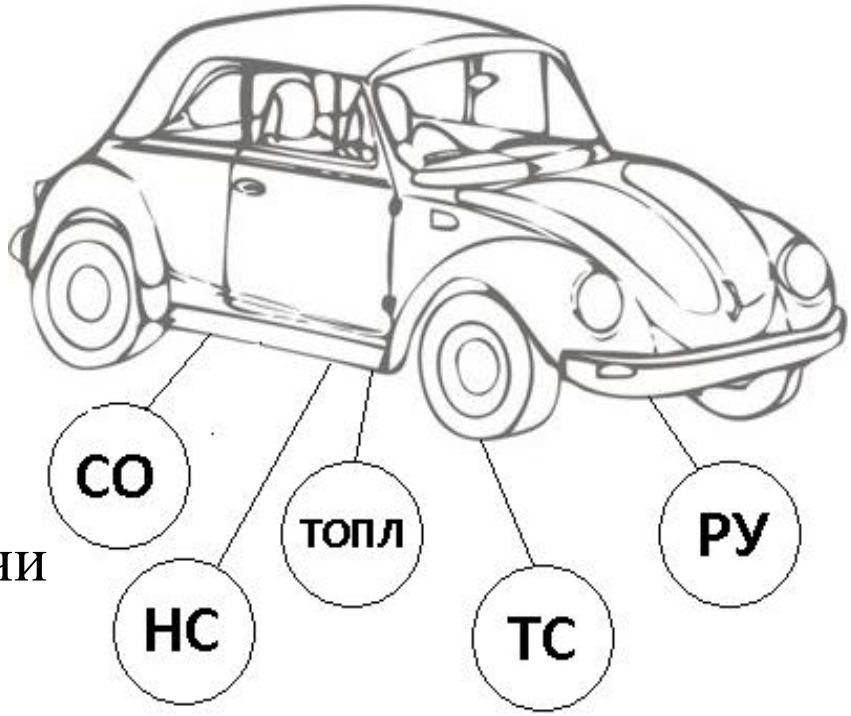
# Объектно-ориентированное программирование (ООП)

Все программы состоят из кода и данных. ООП организует программу вокруг её данных (объектов) и наборов интерфейсов (методов) к этим данным.

**Абстракция. Автомобиль.**
Автомобиль - это программа (конечная цель) .




Состоит из нескольких подсистем: рулевое управление, тормозная система, система подачи топлива и т.д. (каждая система состоит из своих подсистем).
В целом же мы видим автомобиль.
Объекты подсистем взаимодействуют между собой и получается автомобиль.

# Объектно-ориентированное программирование

## Абстракция.Иерархия.

# Три принципа ООП

1. **Инкапсуляция** – механизм, который связывает код и данные, которыми он манипулирует, защищая оба этих компонента от внешнего вмешательства и злоупотреблений.

Доступ к коду и данным контролируется тщательно определенным интерфейсом. Каждый знает, как получить доступ к коду и может использовать код независимо от деталей реализации. Основой **инкапсуляции** является **класс**.

Класс определяет структуру и поведение (данные и код), которые будут совместно использоваться набором объектов. Каждый объект данного класса содержит структуру и поведение, которые определены классом (копии переменных и методы).

При создании класса определяют данные и код (члены класса) - переменные и методы (функции) объекта.

# Три принципа ООП

Каждая переменная или метод могут быть *общедоступными* (**public**) или *приватными* (**private**).

Приватные (**private**)– доступны только для методов, которые являются членами класса.

Общедоступные (**public**) – доступны для всех методов вне класса (для внешних пользователей класса).

**Инкапсуляция** – это сокрытие реализации класса и отделение его внутреннего представления от внешнего.

# Три принципа ООП

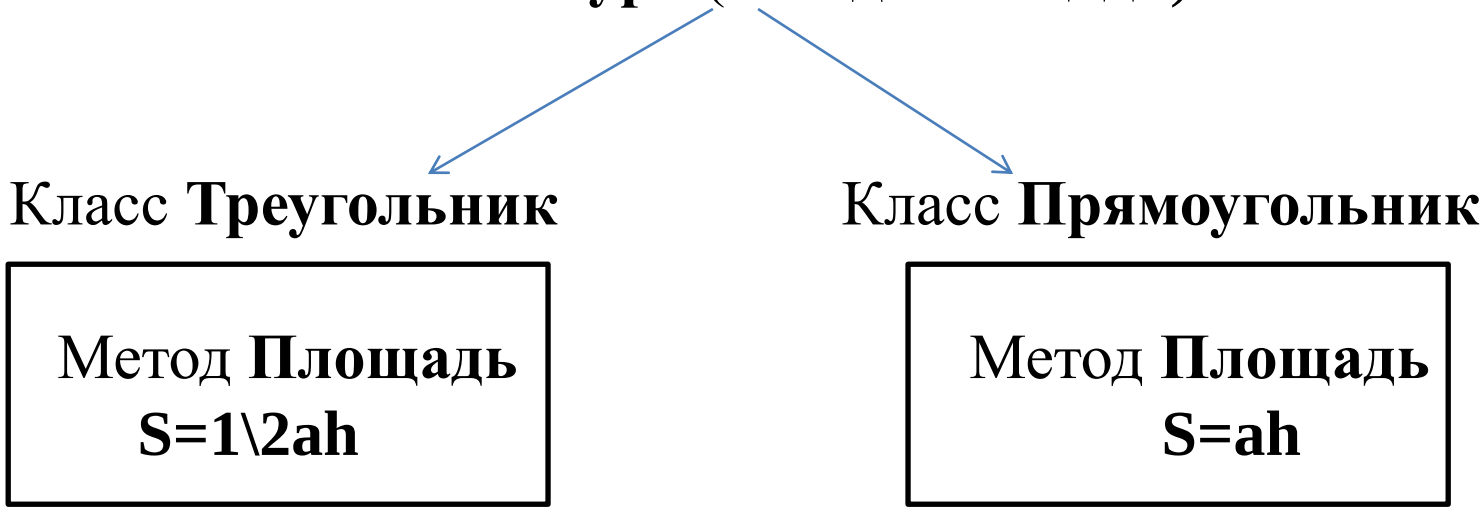
2. **Наследование** – процесс, посредством которого один объект получает свойства другого объекта.

   (отношения между классами, при которых класс использует структуру или поведение другого или других классов).

   Подклассы обычно переопределяют или дополняют унаследованную структуру и поведение.

3. **Полиморфизм** - («имеющий много форм») свойство, которое позволяет использовать один и тот же интерфейс для общего класса действий (один интерфейс – несколько методов (переопределение и перегрузка методов)).

# Три принципа ООП




Один и тот же метод (имя) работает по-разному в разных подклассах одного суперкласса – это полиморфизм.

# Три принципа ООП

Объектно-ориентированный подход способствует:

1. уменьшению сложности программного обеспечения;
2. повышению надежности программного обеспечения;
3. обеспечению возможности модификации отдельных компонентов без изменения остальных;
4. обеспечению возможности повторного использования отдельных компонентов ПО.

# Классы

**Класс** определяет новый тип данных. Этот тип можно использовать для создания объектов.

**Класс** – это шаблон для создания объекта.
**Объект** – экземпляр класса.

**Класс** определяет структуру объекта (переменные) и его методы, образующие функциональный интерфейс (функции для работы с переменными класса).

**Класс** – это логическая конструкция,
**объект** – физическое воплощение, участок памяти.

Общая форма определения класса:

**class** *имя_класса* **extends** *имя_суперкласса* **{**
**type** *перем1***;**
**….**
**type** *перемN***;**

**type** *метод1* **(параметры) { ***тело метода;* **}**
**…**
**type** *методM* **(парметры) { ***тело метода;* **}**
**}**

Методы реализуются внутри класса и определяют, как могут использоваться переменные класса.
По умолчанию в JAVA все переменные public в пределах своего пакета.

# Классы

Каждый экземпляр класса (объект) содержит собственные копии переменных класса.

Данные одного объекта отделены от данных другого объекта.

Общая форма класса не содержит метод **main()**.

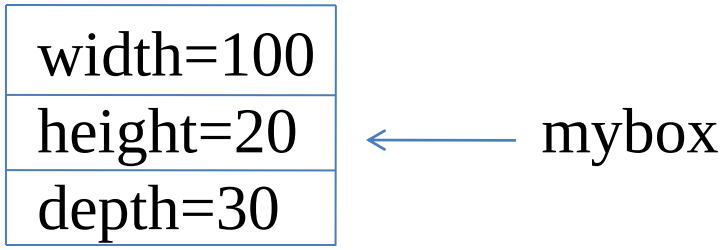
Коробка

```
class Box {
    double  width;
    double  height;
    double  depth;
    }
```

Класс определяет новый тип данных.

# Классы

Чтобы создать объект типа **Box:**

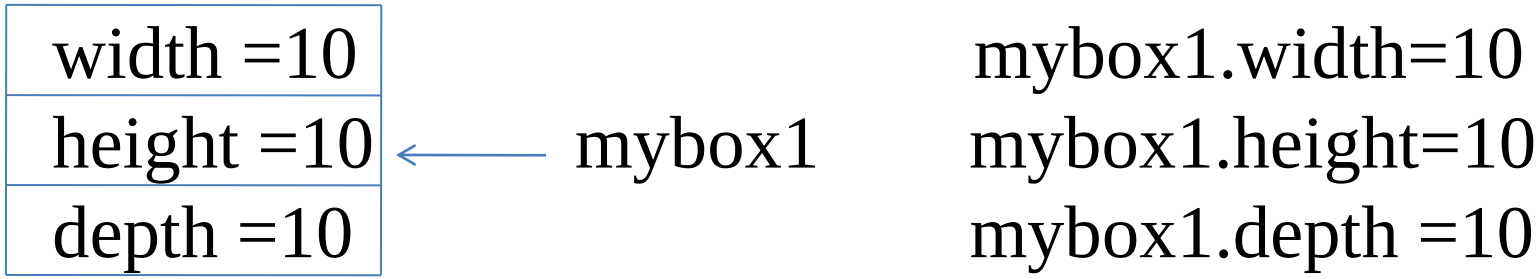

mybox.width = 100 mybox. height = 20 mybox. depth=30

Операция **new** динамически выделяет память и возвращает ссылку на неё. Данная ссылка является адресом объекта в памяти.

# Классы

Можно создать несколько объектов класса Box, каждый из них будет содержать свою копию переменных класса: width, height, depth.

**Box mybox1 = new Box();**




**mybox** и **mybox1** – это ссылки на разные объекты (на разные участки памяти).

**Box mybox2;** Куда указывает ссылка **mybox2**?

mybox2=null;

```
class Box {
      double  width;
      double  height;
      double  depth;       }

class BoxDemo {
   public static void main (String args[]) {
            Box mybox = new Box();
            double vol;
            mybox.width=10;
            mybox.height=20;
            mybox.depth=15;
            vol=mybox.width*mybox.height*mybox.depth;
            System.out.println("Объем равен"+ vol);          }   }
```

Файл с именем BoxDemo.java ( или 2 файла Box.java и BoxDemo.java). После компиляции появляется 2 файла Box.class и BoxDemo.class. На запуск класс, в котором есть main() , т.е. BoxDemo.class.

# Классы

**Box mybox = new Box();**

**Box()** – вызов конструктора класса Box.

**mybox** содержит адрес ячейки памяти фактического объекта.

Конструктор определяет действия, выполняемые при создании объекта класса. Если явный конструктор не определён, то JAVA обеспечит конструктор по умолчанию.

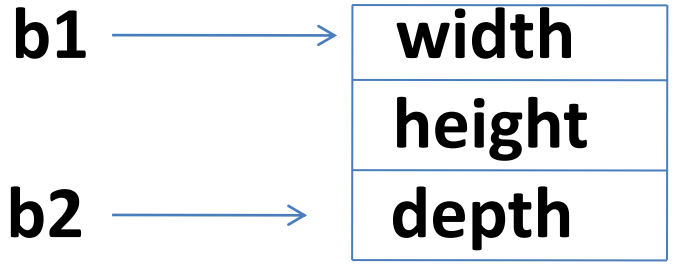
Простым типам не требуется операция **new**.

**new** распределяет память во время выполнения – т.е программа может содержать столько объектов, сколько потребуется.

# Классы

Что происходит в этом коде?

```
Box b1 = new Box();
Box b2 = b1;
```




**b1** и **b2** не связаны между собой, только ссылаются на один участок памяти (объект).

Если b1=null, то на объект будет ссылаться только b2.

Присваивание ссылочной переменной одного объекта ссылочной переменной другого объекта не ведет к созданию копии объекта, а лишь создает копию ссылки.

# Методы

Методы определяют интерфейсы классов.

```
class Box {
    double width;
    double height;
    double depth;

   double volume( ){
       System.out.println(“Объём равен”+ width*height*depth);
       return    width*height*depth;
                        }
  }
```

# Методы

```
class BoxDemo{
    public static void main (String args[]) {
            Box mybox1 = new Box();
            Box mybox2 = new Box();
               mybox1.width=10;
               mybox1.height=10;
               mybox1.depth=10;
               mybox2.width=5;
               mybox2.height=5;
               mybox2.depth=5;
                        mybox1.volume();
                        mybox2.volume();        }   }
```

Метод volume() вызывается к объекту mybox1 со значениями 10 10 10, к объекту mybox2 со значениями 5 5 5. Внутри метода не нужно указывать имя объекта. Метод всегда знает, для какого объекта вызван.

# Методы, возвращающие и принимающие параметры

```
class Box {
    double width;
    double height;
    double depth;

    double volume() {    return    width*height*depth;    }

    void setDim (double w, double h, double d) {
                        width=w;   height=h;    depth=d;   }    }

 class BoxDemo{
    public static void main (String args[]) {
          Box mybox = new Box();
          double vol;
          mybox.setDim(10,10,10);
          vol=mybox.volume();
          System.out.println("Объем равен"+vol);  }   }
```

# Конструкторы

**Конструктор** инициализирует объект после его создания.

Имя конструктора совпадает с именем класса, не имеет возвращающего типа, т.к. неявно возвращается ссылка на создаваемый объект.

Конструктор вызывается перед завершением выполнения new.

```
class Box {
        double width;
        double height;
        double depth;

        Box() {
          System.out.println (“Создание Box-объекта”);
           width=10;   height=10;      depth=10;                }

      double volume() {    return   width*height*depth;   }
 }
```

# Конструкторы

**Box mybox = new Box();**

mybox.width=10; mybox.height=10; mybox.depth=10;

Конструктор с параметрами:

```
Box(double w, double h, double d) {
        width  = w;
        height = h;
        depth = d;       }
```

**Box mybox = new Box (10, 15, 20);**

mybox.width=10; mybox.height=15; mybox.depth=20;

или

**double a=2, b=3, c=4;**

**Box mybox = new Box (a, b, c);**

mybox.width=2; mybox.height=3; mybox.depth=4;

## Ключевое слово this

**this** – ссылка на текущий объект, может использоваться внутри метода, где допускается ссылка на объект текущего класса.

Используется при совпадении имен параметров метода и имен переменных класса.

```
class Box{
        double   width;
        double   height;
        double   depth;

     Box (double width,   double height,   double depth){
          this.width   = width;
          this.height  = height;
          this.depth   = depth;   }
}
```

# Перегрузка методов

В пределах одного класса можно определить два или более методов с одним именем, если только их параметры различны. Такие методы называются *перегруженными* (один из способов поддержки полиморфизма).

При вызове перегруженного метода JAVA  использует тип и/или количество аргументов метода (перегруженные методы должны различаться по типу и/или количеству их параметров).

Java использует ту версию метода, параметры которого соответствуют аргументам, использованным в вызове.

*Перегрузку методов следует использовать для тесно связанных операций.*

# Перегрузка конструктора

```
class Box {
     double width;
     double height;
     double depth;

    Box (double w, double h, double depth){
                          width=w;    height=h;     depth=d;   }

    Box( ) {   width=-1;   height=-1;    depth=-1;   }

    Box (double len) {   width = height = depth = len;  }

    Box( Box ob) {
         width  = ob.width;
         height = ob.height;
         depth  = ob.depth;    }
  }
```

# Перегрузка конструктора

```
class BoxDemo{
        public static void main (String args[]){
            Box m1 = new Box( );
            Box m2 = new Box(10,15,20);
            Box m3 = new Box (7);
            Box m4 = new Box (m1);    } }
```

Передача объекта методу происходит по ссылке, поэтому можно будет изменять значения переменных переданного объекта.

# Перегрузка методов

```
class OverLoad{
   void test( ) {  System.out.println (“Параметры отсутствуют”);   }
   void test( double a) {
         System.out.println(“Внутреннее преобразование test
                                                  с int к double”);  }
    int test( int a,  int b) {   System.out.println (“a+b” + (a+b));
                                  return a+b; }
}
class ODemo {
  public static void main (String args[]){
         OverLoad Ob=new OverLoad();
         int i=88, res;
         Ob.test();
         Ob.test(0.5);
         Ob.test(i);      //такого метода нет, будет вызван test( double a)
         res=Ob.test(10,15);      }      }
```

Любой метод может возвращать любой тип данных, в том числе и созданные типы классов.

```
class Test {
    int a;
    Test ( int i ){ a = i; }

    Test Inc( ) {  Test  temp = new   Test( a+10 );
                   return temp;   }

class Return {
  public static void main (String args[]){
            Test ob1 = new  Test(2);      // ob1.a = 2
            Test ob2;
            ob2=ob1.Inc();                //  ob2.a=12;
            ob2=ob2.Inc();                //ob2.a=22
                                              }   }
```

При вызове Inc() каждый раз создается новый объект и возвращается ссылка на него.

# Управление доступом

**Инкапсуляция** – связывает данные и код.

Посредством инкапсуляции можно управлять тем, какие части программы могут получать доступ к членам класса.

Способ доступа к членам класса определяется спецификатором доступа:

1) **public** – общедоступный (для любого другого кода);
2) **private** – приватный (только для других членов этого класса);
3) **protected** – защищенный (используется при наследовании).

При отсутствии спецификатора член класса считается **public** внутри своего пакета, но не доступен для кода вне этого пакета.

```
class test {
        int a;
        public int b;
        private int c;
         void setc ( int i ){ c = i;}
         int getc( ) {  return c;  }
}
class testDemo {
      public static void main(String args[]){
         test ob = new test( );
            ob.a = 10;
            ob.b = 20;
          //ob.c=30; Ошибка!!!
            ob.setc( 100 );
            System.out.println("a="+a+"b="+ob.b+"c="+ob.getc( )); } }
```

К переменной **c** можно обращаться только с помощью методов, описанных в классе.