Java

Введение в классы

Лекция #3

Пустовалова О.Г.
доцент. каф. мат.мод.
ИММиКН ЮФУ

Содержание

Введение в классы

Класс определяет новый тип данных.

После того, как класс определен можно создавать объекты данного класса.

Класс – шаблон объекта

Объект – экземпляр класса

Класс – логическая конструкция

Объект – физическая сущность

Введение в классы. Пример 1

```
/* A program that uses the Box class.    Call this file BoxDemo.java */
class Box {
    double width;
    double height;
    double depth;
}
// This class declares an object of type Box.
class BoxDemo {
    public static void main(String args[]) {
        Box mybox = new Box();
        double vol;

        // assign values to mybox's instance variables
        mybox.width = 10;
        mybox.height = 20;
        mybox.depth = 15;

        // compute volume of box
        vol = mybox.width * mybox.height * mybox.depth;
        System.out.println("Volume is " + vol);
    }}
```

Класс Box с тремя полями

Создание экземпляра класса

Присвоение полям значений

Введение в классы

Компилятор Java автоматически помещает каждый класс в отдельный класс с расширением .class.

Каждый класс можно было бы поместить в отдельный файл – Box.java и BoxDemo.java.

В Java все объекты классов должны создаваться динамически:

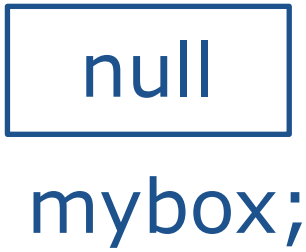
```
Box mybox = new Box();
```

```
Box mybox;// объявление ссылки на объект
mybox = new Box(); // резервирование памяти для объекта
```

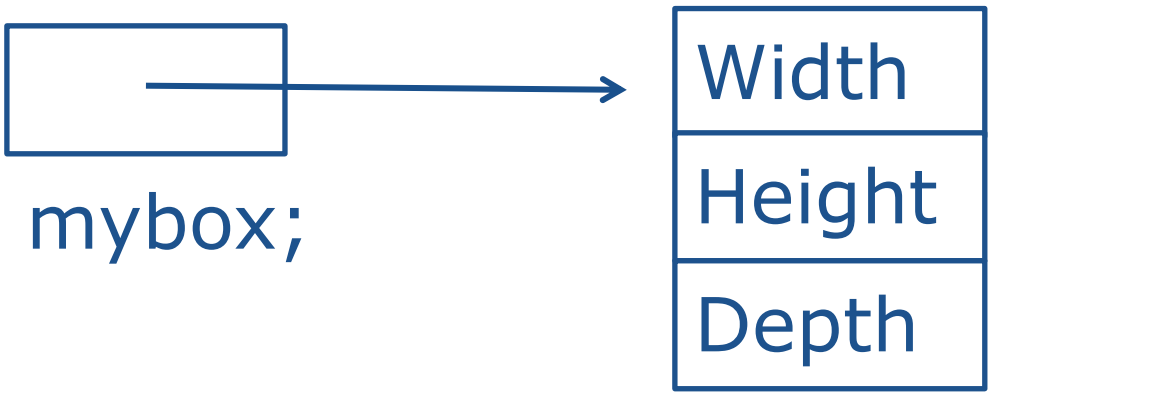
Введение в классы

Оператор **Эффект**

Box mybox;



mybox = **new** Box();



Объект класса Box

Введение в классы

```
Box mybox = new Box();
```

Конструктор класса

Конструктор определяет действия, выполняемые при создании объектов класса.

Оператор **new** резервирует память для объекта во время выполнения. В случае нехватки памяти передается исключение времени выполнения.

Введение в классы. Ссылочные переменные

```
Box mybox = new Box();

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

Box mybox2 = mybox;

System.out.println("mybox2.depth = " + mybox2.depth);
```

mybox2.depth = 15.0

Объект класса Box

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

Введение в классы. Ссылочные переменные

```
mybox=null;

// System.out.println("mybox.depth = " + mybox.depth);

System.out.println("mybox2.depth = " + mybox2.depth);
```

mybox2.depth = 15.0

```
24          mybox=null;
25
26          System.out.println("mybox.depth = " + mybox.depth);
```

```
Exception in thread "main" java.lang.NullPointerException
    at BoxDemo.main(BoxDemo.java:26)

Process finished with exit code 1
```

Введение в классы. Методы класса

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

    // display volume of a box
    void volume() {
        System.out.print("Volume is ");
        System.out.println(width * height * depth);
    }
}
class BoxDemo {
    public static void main(String args[]) {

        Box mybox = new Box();
        mybox.width = 10;
        mybox.height = 20;
        mybox.depth = 15;

        mybox.volume();
    }}
```

Метод ничего не возвращает, а выводит результат на экран.
Обычно такие методы не используют, за исключением методов, которые предназначены для печати.

Volume is 3000.0

Введение в классы. Методы класса. Возвращение значения

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

    // compute and return volume
    double volume() {
        return width * height * depth;
    }
}
class BoxDemo {
    public static void main(String args[]) {

        Box mybox = new Box();
        mybox.width = 10;
        mybox.height = 20;
        mybox.depth = 15;

        double volume=mybox.volume();
        System.out.println("volume = "+volume);
    }}
```

Метод возвращает число (объем) типа double.

Volume is 3000.0

Введение в классы. Конструктор с параметрами

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

    // This is the constructor for Box
    Box(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }
    double volume() {
        return width * height * depth;
    }
}
class BoxDemo {
    public static void main(String args[]) {

        Box mybox = new Box(10,20,15);
        System.out.println("volume = "+mybox.volume());
    }}
```

Конструктор с параметрами

Инициализация объекта

Введение в классы. this

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

This всегда служит ссылкой на объект, для которого был вызван метод.

Введение в классы. Сбор «мусора»

Освобождение памяти в Java выполняется автоматически.

В Java существует процесс, называемый финализацией, который нужен для выполнения каких-либо действий перед уничтожением объектов.

```
protected void finalize(){
    // код финализации
}
```

В Java деструкторы отсутствуют

Метод finalize() вызывается только перед сбором «мусора». Нормальная работа программы не должна зависеть от метода finalize().

ПЕРЕГРУЗКА МЕТОДОВ

Введение в классы. Перегрузка методов

Внутри класса Java можно определить несколько методов с одним именем, только если объявления их параметров (сигнатура) различны.

Перегруженные методы должны отличаться по типу и/или количеству параметров.

```
void test() {
    System.out.println("No parameters");
}

// Overload test for one integer parameter.
void test(int a) {
    System.out.println("a: " + a);
}
```

Когда среда исполнения встречает вызов перегруженного метода, она выполняет ту его версию, параметры которой соответствуют аргументам, использованным в вызове.

Введение в классы. Перегрузка методов. Пример

```
// Demonstrate method overloading.
class OverloadDemo {
    void test() {
        System.out.println("No parameters");
    }

    // Overload test for one integer parameter.
    void test(int a) {
        System.out.println("a: " + a);
    }

    // Overload test for two integer parameters.
    void test(int a, int b) {
        System.out.println("a and b: " + a + " " + b);
    }
}
class Overload {
    public static void main(String args[]) {
        OverloadDemo ob = new OverloadDemo();
        double result;

        // call all versions of test()
        ob.test();
        ob.test(10);
        ob.test(10, 20);}}
```

Перегрузка методов ценна тем, что позволяет обращаться к схожим методам по одному имени.

Перегрузка методов обеспечивает полиморфизм. Это один из способов реализации «один интерфейс – несколько методов»

```
No parameters
a: 10
a and b: 10 20
```

Перегрузка конструкторов. Пример класс Box. Часть 1

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

    // constructor used when all dimensions specified
    Box(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }
    // constructor used when no dimensions specified
    Box() {
        width = -1;  // use -1 to indicate
        height = -1; // an uninitialized
        depth = -1;  // box
    }
    // constructor used when cube is created
    Box(double len) {
        width = height = depth = len;
    }
    // compute and return volume
    double volume() {
        return width * height * depth;
    }}
```

Конструктор для параллелепипеда

Конструктор, если параметры неопределены

Конструктор для куба

Перегрузка конструкторов. Пример класс Box. Часть 2

```
class OverloadCons {

   public static void main(String args[]) {

      // create boxes using the various constructors
      Box mybox1 = new Box(10, 20, 15);
      Box mybox2 = new Box();
      Box mycube = new Box(7);

      double vol;

      // get volume of first box
      vol = mybox1.volume();
      System.out.println("Volume of mybox1 is " + vol);

      // get volume of second box
      vol = mybox2.volume();
      System.out.println("Volume of mybox2 is " + vol);

      // get volume of cube
      vol = mycube.volume();
      System.out.println("Volume of mycube is " + vol);
   }
}
```

Работают 3 конструктора

```
3000.0
-1.0
343.0
```

ИСПОЛЬЗОВАНИЕ ОБЪЕКТОВ В КАЧЕСТВЕ ПАРАМЕТРОВ

Использование объектов в качестве параметров

Методам можно передавать объекты

```
class Test {
   int a, b;

   Test(int i, int j) {
      a = i;
      b = j;
   }
   // return true if o is equal to the invoking object
   boolean equals(Test o) {
      if(o.a == a && o.b == b) return true;
      else return false;
   }
}
class PassOb {
   public static void main(String args[]) {
      Test ob1 = new Test(100, 22);
      Test ob2 = new Test(100, 22);
      Test ob3 = new Test(-1, -1);

      System.out.println("ob1 == ob2: " + ob1.equals(ob2));
      System.out.println("ob1 == ob3: " + ob1.equals(ob3));
   }}
```

Возвращает истину, если объекты эквивалентны

```
ob1 == ob2: true
ob1 == ob3: false
```

Использование объектов в качестве параметров. Пример

```
// Here, Box allows one object to initialize another.

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

    // construct clone of an object
    Box(Box ob) { // pass object to constructor
        width = ob.width;
        height = ob.height;
        depth = ob.depth;
    }

    // constructor used when all dimensions specified
    Box(double w, double h, double d) {
        width = w;
        height = h;
        depth = d;
    }

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

Конструктор. Инициализация объекта другим объектом.

Использование объектов в качестве параметров. Пример

```
class OverloadCons2 {
   public static void main(String args[]) {

      // create boxes using the various constructors
      Box mybox1 = new Box(10, 20, 15);

      Box myclone = new Box(mybox1);

      double vol;

      // get volume of first box
      vol = mybox1.volume();
      System.out.println("Volume of mybox1 is " + vol);

      // get volume of clone
      vol = myclone.volume();
      System.out.println("Volume of clone is " + vol);
   }
}
```

Создание клона на основе mybox1

```
3000.0

3000.0
```

Если поля объекта **myclone** изменить, то это не отразиться на объекте **mybox1**.

ПРИМЕР. КЛАСС EmployeeTest

Пример. Класс EmployeeTest. Часть 1. Employee

```
class Employee
{
    private String name; // к данным полям имеют доступ только методы самого класса
    private double salary;

// конструктор
    public Employee(String n, double s)     {
        name = n;
        salary = s;
    }
// метод
    public String getNameO()     {
        return name;
    }
// метод
    public double getSalary()     {
        return salary;
    }
// метод
    public void raiseSalary(double byPercent)     {
        double raise = salary * byPercent / 100;
        salary += raise;
    }
}
```

В Java все методы определяются в пределах класса но это не делает их встраиваемыми.

Пример. Класс EmployeeTest. Часть 1. Employee

```
class Employee
{
    private final String name;

    private double salary;
```

поле name из класса Employee
можно объявить неизменяемым, поскольку после создания объекта оно уже не изменяется

Пример. Класс EmployeeTest. Часть 2. Employee

```
public class EmployeeTest
{
   public static void main(String[] args)
   {

// заполнить массив staff тремя объектами типа Employee
      Employee[] staff = new Employee[3];

      staff[0] = new Employee("Carl Cracker", 75000);
      staff[1] = new Employee("Harry Hacker", 50000);
      staff[2] = new Employee("Tony Tester", 40000);

// поднять всем работникам зарплату на 5%
      for (Employee e : staff)
         e.raiseSalary(5);

// вывести данные обо всех объектах типа Employee
      for (Employee e : staff)
         System.out.println("name=" + e.getNameO() + ",salary="
               + e.getSalary() );
   }
}
```

Введение в классы. Конструктор

- ✓ Имя конструктора совпадает с именем класса.
- ✓ Класс может иметь несколько конструкторов.
- ✓ Конструктор может иметь один или несколько параметров или же вообще их не иметь.
- ✓ Конструктор не возвращает никакого значения.
- ✓ Конструктор всегда вызывается совместно с операцией **new**.

Все объекты в Java размещаются в динамической памяти и конструкторы вызываются только вместе с операцией **new**.

```
Employee number007("James Bond", 10000)
// допустимо в C++, но не в Java!
```

СТАТИЧЕСКИЕ ПОЛЯ И МЕТОДЫ

Статические поля и методы

Статические поля

Поле с модификатором доступа **static** существует в одном экземпляре для всего класса.

```
class Employee
{
…
   private int id;

   private static int nextId = 1;
}
```

Если существует тысяча объектов типа Employee, то в них есть тысяча полей id: по одному на каждый объект.

В то же время существует только один экземпляр статического поля nextId.

Даже если не создано ни одного объекта типа Employee, статическое поле nextId все равно существует. Оно принадлежит классу, а не конкретному объекту.

Статические поля и методы

Статические константы

Статические переменные используются довольно редко. В то же время статические константы применяются намного чаще.

```
public class Math
{
…
    public static final double PI = 3.14159265358979323846;
…
}
```

Если бы ключевое слово static было пропущено, константа PI была бы обычным полем экземпляра класса Math.

Это означает, что для доступа к такой константе нужно было бы создать объект типа Math, причем каждый такой объект имел бы свою копию константы PI.

Статические поля и методы

Статические константы

Еще одной часто употребляемой является статическая константа **System, out.**

Она объявляется в классе **System** следующим образом:

```
public class System
{
…
    public static final PrintStream out =…;
…
}
```

открытыми константами (т.е. полями, объявленными с ключевым словом **final)** можно пользоваться смело.
Так, если поле **out** объявлено как **final,** ему нельзя присвоить другой поток вывода:

```
System.out = new PrintStream(...); //Ошибка: поле out изменить нельзя!
```

Статические поля и методы

Статические методы

Статическими называют методы, которые не оперируют объектами.

Например, метод pow () из класса Math является статическим. При вызове метода Math. pow (x, a) вычисляется степень числа. При выполнении этого метода не используется ни один из экземпляров класса Math.

Статические методы имеют доступ к статическим полям класса:

```
public static int getNextId {

    return nextId; // возвратить статическое поле
}
```

ЗАДАНИЯ

Задания

1. Создать класс прямоугольного треугольника (см. слайд 18). Поля – основание и высота. Методы – площадь, гипотенуза, периметр.

2. Создать класс окружность (см слайд 18). Поля – координаты центра и радиус. Предусмотреть конструктор для вырожденной окружности, когда радиус равен нулю. Методы – площадь, длина окружности.

3. Создать класс студент (см слайд 25). Поля – фамилия, имя, курс, группа, средний балл. Методы – перевести на следующий курс. Изменить средний балл. Стипендия в зависимости от среднего балла.

4. Создать класс книга (см слайд 25). Поля – название, автор, год, цена, количество. Методы – изменить количество. Изменить цену в зависимости от количества. Стоимость всего количества книг.

ПЕРЕДАЧА АРГУМЕНТОВ

Передача аргументов. Передача по значению

Элементарные типы передаются **по значению**

```
class Test {
    void meth(int i, int j) {
        i *= 2;
        j /= 2;
    }
}

class CallByValue {

    public static void main(String args[]) {

        Test ob = new Test();
        int a = 15, b = 20;

        System.out.println("a and b before call: " + a + " " + b);

        ob.meth(a, b);

        System.out.println("a and b after call: " +  a + " " + b);
    }}
```

→ **15 20**

→ **15 20**

Передача аргументов. Передача по ссылке

Объекты передаются **по ссылке**

```
class Test2 {
    int a, b;

    Test2(int i, int j) {
        a = i;
        b = j;
    }
    // pass an object
    void meth(Test2 o) {
        o.a *=  2;
        o.b /= 2;
    }
}
class PassObjRef {

    public static void main(String args[]) {
        Test2 ob = new Test2(15, 20);

        System.out.println("ob.a and ob.b before call: " +  ob.a + " " + ob.b);
        ob.meth(ob);
        System.out.println("ob.a and ob.b after call: " +   ob.a + " " + ob.b);
    }}
```

15 20

30 10

ГОРЯЧИЕ КЛАВИШИ IntelliJ IDEA

Горячие клавиши IntelliJ IDEA

Ctr + Alt + L

- приведение кода в соответствие code style

Спасибо за внимание!