

Объекты в ООП

Внутренний класс

```
// Применение внутреннего класса
```

```
class Outer {  
    int nums[];  
    Outer(int n[]) {  
        nums = n;}  
    void Analyze() {  
        Inner inOb = new Inner();  
  
        System.out.println("Максимум: " +  
            inOb.max());}
```

Результат: Максимум: 9

```
// Внутренний класс
```

```
    class Inner {  
        int max() {  
            int m = nums[0];  
            for(int i = 1; i <  
nums.length; i++) {  
                if(nums[i] > m) m =  
nums[i]; }  
            return m;  
        }  
    }  
    class NestedClassDemo {  
        public static void main(String  
args[]) {  
            int x[] = {1, 2, 3, 5, 6, 7, 8,  
9};  
  
            Outer outOb = new Outer(x);  
            outOb.Analyze(); }}
```

Нестатические вложенные классы

```
public class Airplane
{ private String name, id, flight;
  public Airplane(String name, String id,
String flight)
{ this.name = name; this.id = id;
this.flight = flight; }
// getters/setters }
```

```
public class Airplane
{ private String name, id, flight;
  private Wing leftWing = new Wing("Red", "X3"),
rightWing = new Wing("Blue", "X3");
  public Airplane(String name, String id, String
flight)
  { this.name = name; this.id = id; this.flight
= flight; }
  private class Wing
  { private String color, model;
    private Wing(String color, String model)
    { this.color = color; this.model = model;
}
}
// getters/setters } // getters/setters }
```

Вложенный статический класс

```
public class Outer2 {  
    public int pubOutVar; //  
переменная не статическая и мы  
не имеем к ней доступа  
    // из внутреннего  
статического класса  
    private int prOutVar;  
    public Outer2(){  
        static class Nested2{  
            public static int  
pub_innVar; // тут все в порядке  
        }  
    }  
}
```

```
public Nested2() {}  
  
int getOuterPublicVariable()  
{  
    return Outer2.this.pubOutVar; //  
ошибка  
    return Outer2.pubOutVar; // ошибка  
}  
  
int getOuterPrivateVariable()  
{  
    return Outer2.this.prOutVar; // ошибка  
    return Outer2.prOutVar; // ошибка  
}  
  
}
```

```
public abstract class Building {
    private String name, address,
type;
    Building(String name, String
address) {
        this.name = name;
        this.address = address;
    }
    public static class Platform
extends Building {
        public Platform(String
name, String address) {
            super(name, address);
            setType("Platform");
        }
        // some additional logic
    }
}
```

```
public static class House extends Building {
    public House(String name, String address) {
        super(name, address);
        setType("House");
    }
    // some additional logic
}
public static class Shop extends Building {
    public Shop(String name, String address) {
        super(name, address);
        setType("Shop");
    }
    // some additional logic
}
// getters/setters
}
```

Пример создания экземпляра вложенного статического класса:
`Building.Shop myShop = new Building.Shop("Food & Fun!", "Kalyaeva 8/53");`

Локальный класс

```
class LocalClassDemo {
    public static void main(String
args[]) {
        // Внутренняя версия класса
        ShowBits
        // Локальный класс, вложенный
        в главном методе main
        class ShowBits {
            int numbits;
            ShowBits(int n) {
                numbits = n;
            }
            void show(long val) {
                long mask = 1;
```

```
// Сдвиг влево для установки единицы в нужно позиции
        mask <<= numbits - 1;
            int spacer = 0;
            for(; mask != 0; mask >>=1) {
                if((val & mask) != 0)
System.out.print("1");
                else System.out.print("0");
                spacer++;
                if((spacer % 8) == 0) {
                    System.out.print(" ");
                    spacer = 0;
                }
            }
            System.out.println();
        } }
        for(byte b = 0; b < 10; b++) {
            ShowBits byteval = new ShowBits(8);
            System.out.print(b + " в двоичном
представлении: ");
            byteval.show(b); } }
```

Локальный класс

Результат:

0 в двоичном представлении: 00000000

1 в двоичном представлении: 00000001

2 в двоичном представлении: 00000010

3 в двоичном представлении: 00000011

4 в двоичном представлении: 00000100

5 в двоичном представлении: 00000101

6 в двоичном представлении: 00000110

7 в двоичном представлении: 00000111

8 в двоичном представлении: 00001000

9 в двоичном представлении: 00001001

АНОНИМНЫЙ КЛАСС

```
class OuterClass
{ public OuterClass() {}
void methodWithLocalClass (final int interval)
{ // При определении анонимного класса применен
  полиморфизм - переменная listener содержит экземпляр
  // анонимного класса, реализующего существующий
  интерфейс ActionListener
  ActionListener listener = new ActionListener()
  { @Override public void actionPerformed(ActionEvent event)
    { System.out.println("Эта строка выводится на экран каждые " +
    + interval + " секунд"); } };
  Timer t = new Timer(interval, listener);
  // Объект анонимного класса использован внутри метода
  t.start(); } }
```

Абстракция

```
public class Animal {  
    int weight;//default  
    public void voice()  
    {System.out.println("Some  
Voice");  
}}
```

```
public class Main {  
  
    public static void  
    main(String[] args) {  
        Animal animal=new Animal();  
        animal.weight=2;  
        System.out.println(animal.weigh  
t);  
    } }
```

Результат: 2

С модификатором доступа private

```
public class Animal {  
    private int weight;  
    public void voice()  
    {System.out.println("Some  
Voice");  
}  
}
```

```
public class Main {  
  
    public static void  
    main(String[] args) {  
        Animal animal=new Animal();  
        animal.weight=2;  
        System.out.println(animal.weigh  
t);  
    }  
}
```

Результат: The field Animal.weight is not visible

Инкапсуляция

```
public class Animal {  
    private int weight;  
    public int getWeight() {  
        return weight;  
    }  
    public void setWeight(int weight) {  
        if (weight>0)  
            this.weight = weight;  
        else System.out.println("Weight must be  
> 0");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Animal animal=new Animal();  
        animal.setWeight(-1);  
        System.out.println(animal.getWeight());  
    }  
}
```

Результат: Weight must be > 0

```
public class Animal {
    private int weight;
    public void voice()
    {System.out.println("Some
Voice");
}
    public int getWeight() {
    return weight;
}
    public void setWeight(int
weight) {
    this.weight = weight;
}}
```

```
public class Main {

    public static void
main(String[] args) {
    Animal animal=new Animal();
    animal.weight=2;
    System.out.println(animal.weigh
t);
} }
```

```
public class Cat extends
Animal
{
public void voice()
{
System.out.println("Meow")
;}
}
```

```
public class Main {
public static void
main(String[] args) {
Cat cat=new Cat();
cat.setWeight(6);
System.out.println(cat.getWeigh
t());
} }
```

Результат: 6

```
public class Animal {  
    public void voice()  
    {System.out.println("Some Voice");  
    }  
}
```

```
public class Cat extends Animal  
{public void voice()  
{System.out.println("Meow");}}
```

```
public class Dog extends Animal  
{public void voice(){  
System.out.println("Bark Bark");  
}}
```

```
public class Main {  
    public static void main(String[] args) {  
        Animal animal=new Animal();  
        animal.voice();  
        animal=new Dog();  
        animal.voice();  
        animal=new Cat();  
        animal.voice();  
    }  
}
```

Результат: Some Voice
Bark Bark
Meow

Хотя наследование очень интересный и эффективный механизм, но в некоторых ситуациях его применение может быть нежелательным. И в этом случае можно запретить наследование с помощью ключевого слова **final**.

```
public final class Person {  
}
```


Полиморфизм

```
public class Dog extends Animal
{public String name;
private int age;
public void voice(){
System.out.println("Bark Bark");
}
public void voice(String s)
{}
public void voice(int a)
{}
}
```

Перегрузка: имя метода остается неизменным, но параметры, возвращаемый тип и количество параметров могут изменяться.