КЛАСОВЕ, ОБЕКТИ, МЕТОДИ

гл.ас. д-р Мария Евтимова
каб.2305а
e- mail:mevtimova@tu-sofia.bg

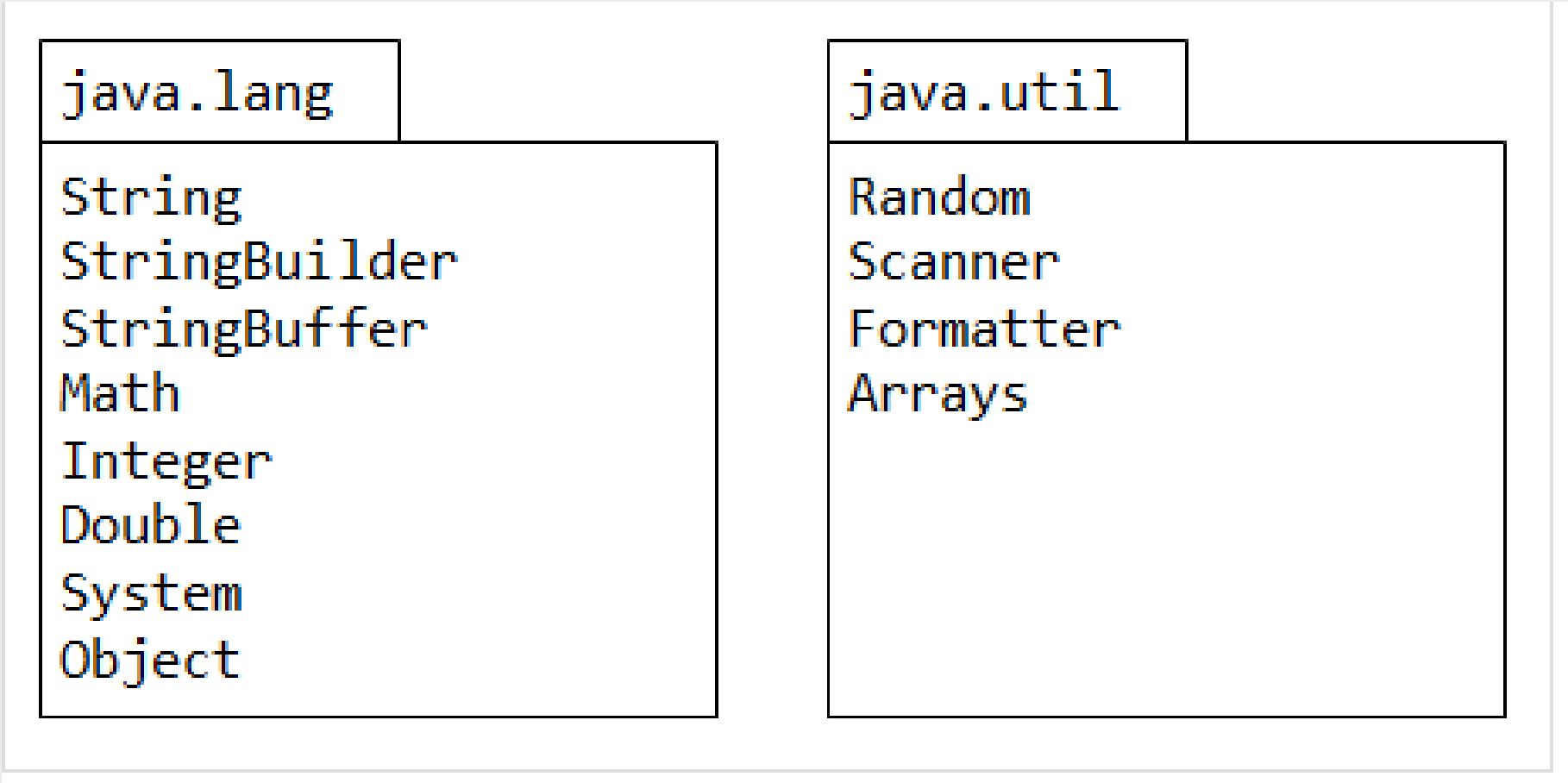
Импортиране на библиотеки

import име_на_пакет.*име_на_клас*;

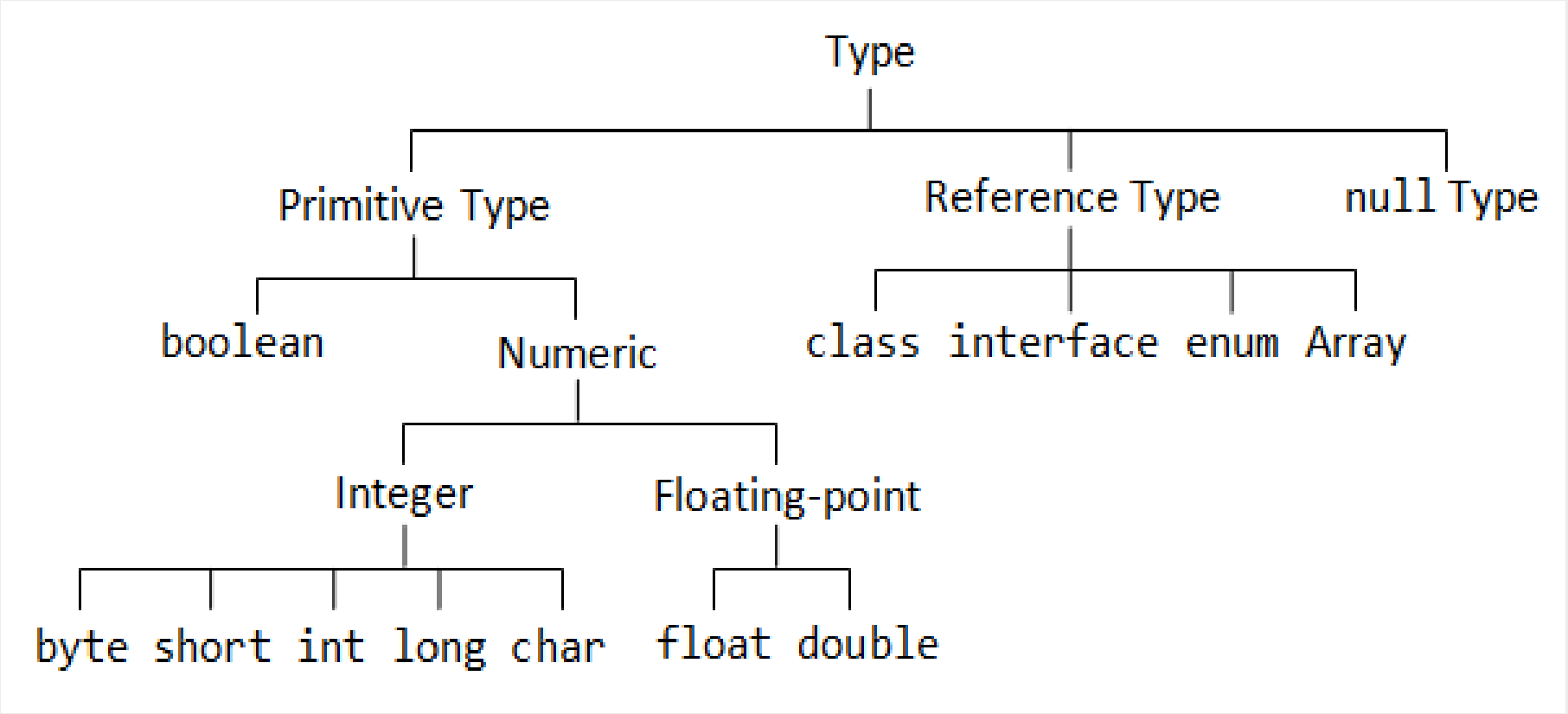
import име_на_пакет.*

Пример: import java.util.*;

Вградени класове в JAVA

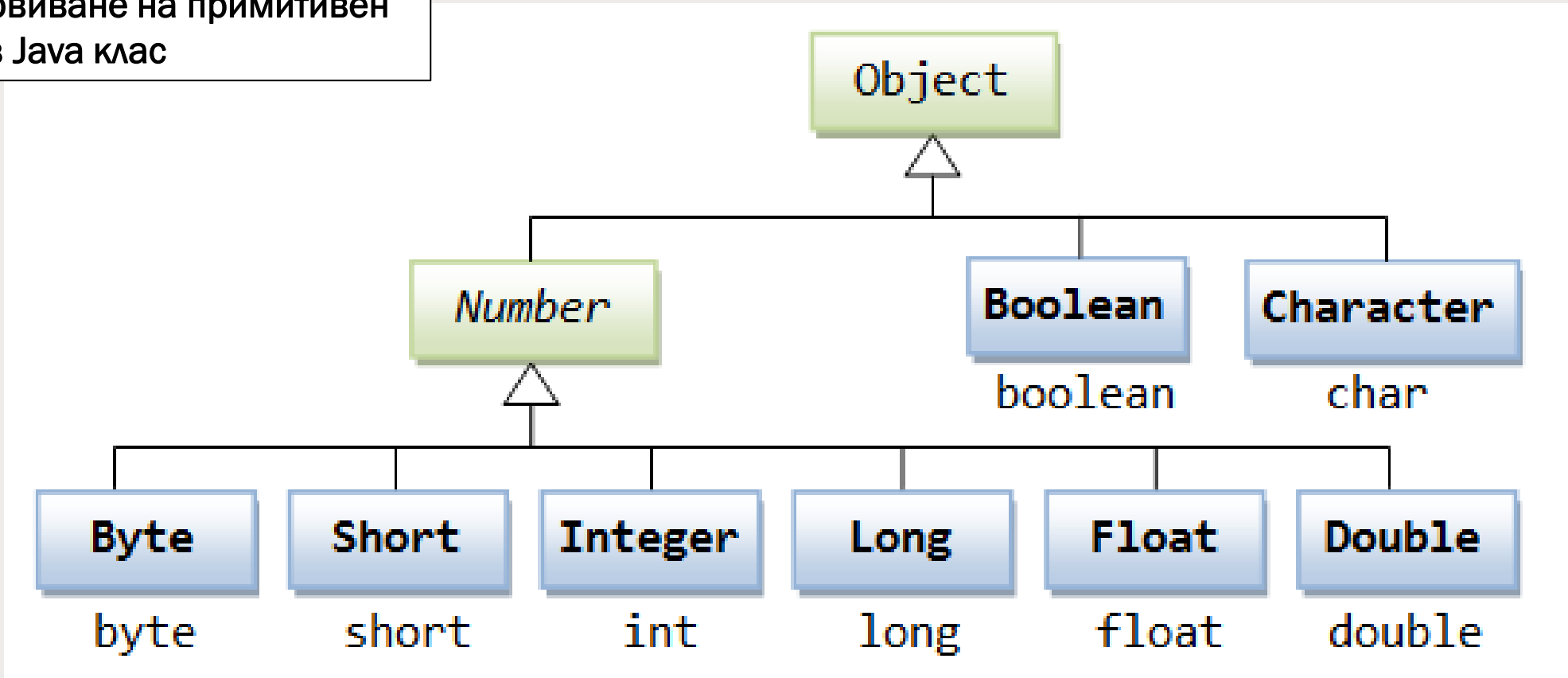


Видове променливи



Клас обвивка за примитивни типове

Класа обвивка се използва за обвиване на примитивен тип в Java клас



Обвивка посредством конструктор

// Обвивка на int примитивна стойност в Integer обект

- Integer aIntObj = new Integer(5566);
- Double aDoubleObj = new Double(55.66);
- Character aCharObj = new Character('z');
- Boolean aBooleanObj = new Boolean(true);

Разопаковане посредством метода xxxValue()

- public byte byteValue() – връща опакована "numeric" стойност, като байт
- public short shortValue()
- public abstract int intValue()
- public abstract long longValue()
- public abstract float floatValue()
- public abstract double doubleValue()

charValue()/ booleanValue()

- public char charValue()
- public boolean booleanValue()

Примери

- Integer intObj = new Integer(556677);

```
int i = intObj.intValue();
short s = intObj.shortValue();
byte b = intObj.byteValue();
```

- Double doubleObj = new Double(55.66);

```
double d = doubleObj.doubleValue();
int i1 = doubleObj.intValue();
```

- Character charObj = new Character('z'); // разопаковане

```
char c = charObj.charValue();
```

- Boolean booleanObj = new Boolean(false);

```
boolean b1 = booleanObj.booleanValue();
```

Константи MIN_VALUE,MAX_VALUE и SIZE

Всички освен Boolean

- public static final *type* MIN_VALUE // Минимална стойност
- public static final *type* MAX_VALUE // Максимална стойност
- public static final int SIZE // Брой битове

Float и Double само

- public static final int MAX_EXPONENT // Максимална експонента
- public static final int MIN_EXPONENT //Минимална експонента

Примери за Integer клас

```
System.out.println(Integer.MAX_VALUE); // 2147483647

System.out.println(Integer.MIN_VALUE); // -2147483648

System.out.println(Integer.SIZE); // 32
```

Примери за Double клас

```
System.out.println(Double.MAX_VALUE); // 1.79769313486231570E308

System.out.println(Double.MIN_VALUE); // 4.9E-324

System.out.println(Double.SIZE); // 64

System.out.println(Double.MAX_EXPONENT); // 1023

System.out.println(Double.MIN_EXPONENT); // -1022
```

Статични методи за преобразуване на низове

- public static byte parseByte(String s)
- public static short parseShort(String s)
- public static int parseInt(String s)
- public static long parseLong(String s)
- public static float parseFloat(String s)
- public static double parseDouble(String s)
- public static boolean parseBoolean(String s)- връща true при низ

Примери

```
int i = Integer.parseInt("5566");

 i = Integer.parseInt("abcd"); // Грешка при форматиране
i = Integer.parseInt("55.66"); // Грешка при форматиране



double d = Double.parseDouble("55.66");
```

Автоматично опаковане и разопаковане (auto- boxing/auto- unboxing)

```
Integer intObj = new Integer(5566); // опаковане int до Integer
 int i = intObj.intValue();
```

Integer intObj = 5566; //автоматично опаковане от int до Integer

int i = intObj; //автоматично разопаковане от Integer до int

Класове и методи

- Всеки клас може да съдържа безброй много методи
- По време на изпълнението на програмата, метода може да се извиква безброй пъти
- Метода обикновено е написан в друг метод

Метод

```
връщан_тип име_на_метода(/*списък с аргументи */)
{
/*тяло на метода*/
}
```

Пример: int a=y.f(x);

Списък с аргументи

- определя каква информация се подава на метода

име_на_обект.име_на_метод(аргумент1,аргумент2,аргумент3)

Пример

```
void something(){
return;
}
```

```
int save(String str){
return str.length()*2;
}
```

колко байта са необходими за съхраняване на информация в даден обект от тип String

предефиниране / overriding

дефиниция- да се промени поведението на същесвуваща функция от базовия клас

функционалност- създаване на нова дефиниция на дадена функция в производния клас

Преопределяне на методи

Method Overloading

- **позволява едно и също име на метод да се използва с различни типове аргументи**

- **може да се използва за всеки метод**

Преопределяне базирано на подредбата на аргументи

```
public class OverloadingArg{
static void print(String str, int i){
System.out.println(“String:”+str+”,int:”+i);
}
static void print(int i, String str){
System.out.println(“int:”+”,String”+str);
}
public static void main(String[] args){
print(“String”,12);
print(80,”Integer”);
}}
```

Преопределяне с примитиви

- void func(){}
- int func(){}

Garbage collector

- **възстановява паметта, която вече не се използва от програмата ви**

```
protected void finalize( ) {
//кода за финализиране
}
```

finalize() дава възможност да се изпълни важно изчистване на паметта по време на работата на garbage collector

Final
клас, променлива, метод

- final класа не може да бъде под клас
- final метода не може да бъде пренаписан в под клас
- final променлива не може да бъде сменена с нова стойност

Final променлива

// клас java.lang.Math

- public static final double PI = 3.141592653589793;
- public static final double E = 2.718281828459045;

// клас java.lang.Integer

- public static final int MAX_VALUE = 2147483647;
- public static final int MIN_VALUE = -2147483648;
- public static final int SIZE = 32;

Пример

```
public class FinalReference{
public static void main (String[] args)
final StringBuffer sb= new StringBuffer(“Student”);
//final референтен тип
sb.append(“,life”);
//може да се смени съдържанието на референцията
System.out.println(”The object is\” + sb+”\”);
```

Final не може да се пренаписва

```
class Bike{
  final  void  drive(){
System.out.println(“driving”);
}
}
    class  Vehicle extends Bike{
   void  drive(){
System.out.println(“moving with 40kmph");
}
   public  static  void  main(String args[]){
   Vehicle vehicle= new Vehicle();
   vehicle.drive();
   }
}
```

Ключова дума this

- “този обект” или “текущият обект”
- връща референция към текущия обект

при static методите няма this

Употреба на this

Пример с this

```
class Person {
int age;
Person(int age) {
this.age = age;
} }
```

Вътрешни класове

- писането на клас в друг клас се нарича вътрешен клас, а класа в който се пише външен клас

```
class Outer{
   class Inner{
   }
}
```

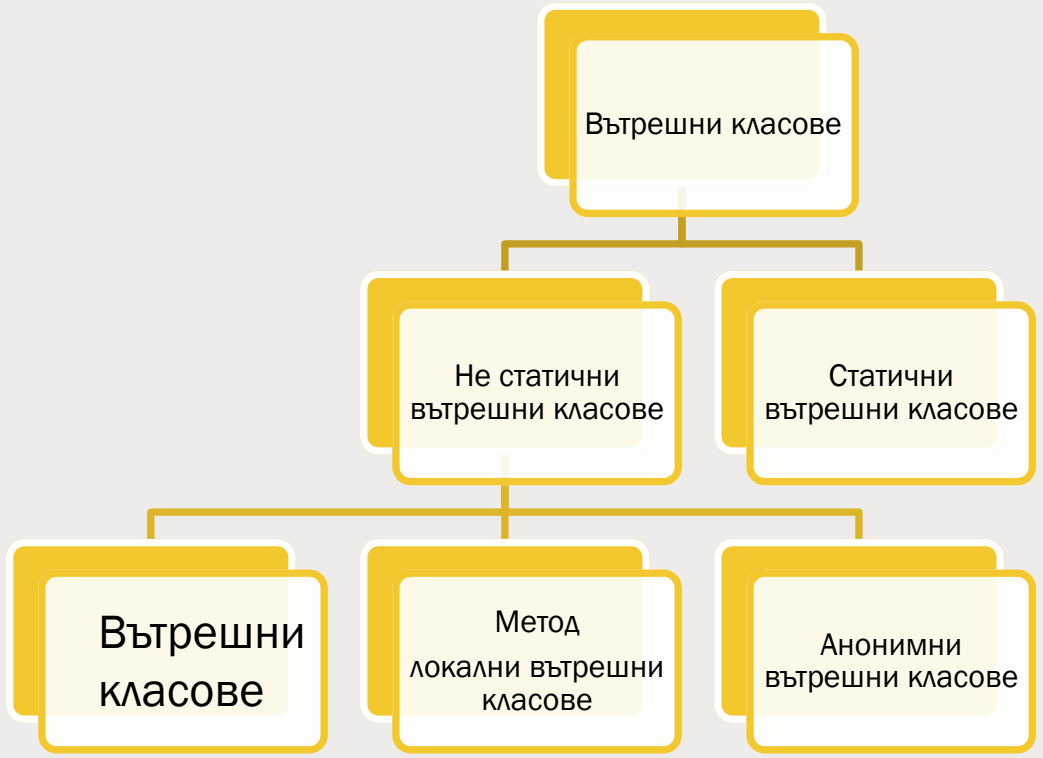
Свойства на вложени(вътрешни) класове

- вложения клас е член на външния клас
- вложения клас може да има достъп до частните членове (променливи / методи) на обграждащия го външен клас
- вложения класа НЕ е подклас на външния клас.

Употреба на вътрешни класове

- **Да контролира визуализациите** (на променливите и методите на членството) между вътрешния или външния клас
- Да се постави код на дефиницията на класа по- близо до мястото, където ще се използва, за да стане програмата **по- лесна за разбиране**
- **За управление на namespace**

Видове вътрешни класове



```
class Outer{
 int num;
 // вътрешен клас
 private class Inner {
 public void print() {
 System.out.println("This is an inner class");
} }
 // достъп до вътрешен клас от метод
void display_Inner() {
Inner inner = new Inner();
inner.print();
}}
```

```
 public class My_class {
 public static void main(String args[]) {
// създаване на външен клас
 Outer outer = new Outer();
// достъп до метода display_Inner()
outer.display_Inner();
 }}
```

Метод- локален вътрешен клас

```
public class Outerclass {
 void my_Method() {
int num = 12;
class MethodInner {
public void print() {
System.out.println("This is method inner class "+num);
 }
}
```

```
//достъп до вътрешния клас
MethodInner inner = new MethodInner();
 inner.print();
}
public static void main(String args[])
{
 Outerclass outer = new Outerclass();
 outer.my_Method(); } }
```

Анонимен вътрешен клас – без име на класа

```
AnonymousInner an_inner = new AnonymousInner() {
 public void my_method() {
 ........


.........
   }
};
```

Анонимен вътрешен клас

```
abstract class AnonymousInner {

public abstract void mymethod();

}

public class Outer_class {

public static void main(String args[]) {

AnonymousInner inner = new
AnonymousInner() {

public void mymethod() {

System.out.println("This is an example of
anonymous inner class");

}

 };

inner.mymethod();

 }

 }
```

Статичен вътрешен клас

```
class MyOuter {
static class Nested {
 }
 }
```

Статичен вътрешен клас

```
public class Outer {
 static class Nested {
 public void my_method() {
 System.out.println(„Static nested class");
 } }
public static void main(String args[]) {
 Outer.Nested nested = new Outer.Nested();
 nested.my_method();
 } }
```

Ключова дума super

- референтна променлива, която се използва за препращане на обект от непосредствен родителски клас
- при създаване на екземпляр на подклас, екземпляр на родителски клас се създава неявно, който се посочва от super референтната променлива

Употреба на думата super

- super може да се използва за препращане на променлива на потребителски модел на родителски клас.
- super може да се използва за извикване на метод на непосредствен родителски клас.
- super () може да се използва за извикване на конструктор на непосредствен родителски клас.

Пример

```
class  Animal{
String color="white";
}
class Cat extends Animal{
String color="black";
void printColor(){
System.out.println(color);//изписване на цвета от клас Cat
System.out.println(super.color);//изписване на цвета от клас  Animal
}
}
class TestSuper1{
public static void main(String args[]){
Cat c=new Cat();
c.printColor();
}}
```