

# Дополнительные сведения о методах и классах

- В этой лекции возобновляется рассмотрение классов и методов.
- Сначала будет показано, каким образом контролируется доступ к членам класса, а затем рассмотрены особенности передачи и возврата объектов из методов, перегрузки методов, использования рекурсии и ключевого слова `static`.
- Кроме того, в этой лекции будут представлены вложенные классы и методы с аргументами переменной длины.

# Темы лекции 11. Дополнительные сведения о методах и классах

- Управление доступом к членам классов
- Передача объектов при вызове методов
- Возврат объектов из методов
- Перегрузка методов
- Перегрузка конструкторов
- Применение рекурсии
- Использование ключевого слова `static`
- Использование аргументов переменной длины

# Управление доступом к членам классов

Поддержка свойства инкапсуляции в классе дает два главных преимущества:

- класс связывает данные с кодом
- класс предоставляет средства для управления доступом к его членам

Ограничение доступа к членам класса является основополагающей частью объектно-ориентированного программирования, поскольку оно позволяет исключить неверное использование объекта

# Управление доступом к членам классов

Управление доступом к членам класса в Java осуществляется с помощью трех модификаторов доступа (называемых также спецификаторами):

- `public`
- `private`
- `protected`
- а так же модификатор по-умолчанию

Если все классы в программе относятся к одному пакету, то отсутствие модификатора доступа равнозначно указанию модификатора `public` по-умолчанию

# Управление доступом к членам классов

```
class MyClass
{
    private int alpha; // закрытый доступ
    public int beta; // открытый доступ
    int gamma; // тип доступа по умолчанию (по
существу, открытый)
/* Методы доступа к переменной alpha. Члены класса
могут обращаться к закрытым членам того же класса.
*/
    void setAlpha(int a)
        { alpha = a; }
    int getAlpha()
        { return alpha; }
}
```

# Передача объектов при вызове методов

Существует два способа передачи аргументов методу:

- ВЫЗОВ ПО ЗНАЧЕНИЮ
- ВЫЗОВ ПО ССЫЛКЕ



# Передача объектов при вызове методов

```
// Простые типы данных передаются методам по значению
class Test { /* Этот метод не может изменить значения
аргументов, передаваемых ему при вызове. */
void noChange(int i, int j)
{ i = i + j; j = -j; } }
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.noChange(a, b);
  System.out.println("a and b after call: " + a + " " +
b); } }
```

**a and b before call: 15 20 a and b after call: 15 20**

# Передача объектов при вызове методов

```
// Объекты передаются методам по ссылке, class Test {  
int a, b;  
Test(int i, int j) { a = i; b = j;}  
/* Передача объекта методу. Теперь переменные ob.a b и  
ob.b из передаваемого объекта можно изменить. */  
void change(Test ob) {  
    ob.a = ob.a + ob.b;  
    ob.b = -ob.b;}}  
class CallByRef { public static void main(String args[])  
{ Test ob = new Test(15, 20);  
    System.out.println("ob.a and ob.b before call: " +  
ob.a + " " + ob.b);  
    ob.change(ob);  
    System.out.println("ob.a and ob.b after call: " +  
ob.a + " " + ob.b); } }
```

**ob.a and ob.b before call: 15 20**

**ob.a and ob.b after call: 35 -20**



# Возврат объектов из методов

Метод может возвращать данные любого типа, включая и  
ТИПЫ КЛАССОВ

```
// Возврат объекта типа String,  
class ErrorMsg { String msgs[] = { "Output Error", "Input  
Error", "Disk Full", "Index Out-Of-Bounds" };  
// вернуть объект типа String в виде сообщения об ошибке  
String getErrorMsg(int i)  
{ if(i >=0 & i < msgs.length) return msgs[i];  
else return "Invalid Error Code"; } }  
  
class ErrMsg {  
public static void main(String args[])  
{ ErrorMsg err = new ErrorMsg();  
System.out.println(err.getErrorMsg(2));  
System.out.println(err.getErrorMsg(19)); } }
```

**Disk Full**

**Invalid Error Code**

# Перегрузка методов

- Для того чтобы перегрузить метод, достаточно объявить его новый вариант, отличающийся от уже существующих, а все остальное сделает компилятор.
- Возвращаемый тип не предоставляет достаточных сведений для принятия решения о том, какой именно метод должен быть использован.
- Нужно лишь соблюсти одно условие: тип и/или число параметров (сигнатура) в каждом из перегружаемых методов должны быть разными.
- При вызове метода выполняется лишь тот его вариант, в котором параметры соответствуют передаваемым аргументам.

# Перегрузка методов

```
// Перегрузка методов,  
class Overload  
{ // Первый вариант метода.  
void ovlDemo() { System.out.println("No  
parameters"); }  
// перегрузить метод ovlDemo с одним параметром типа  
int. // Второй вариант метода.  
void ovlDemo(int a)  
{ System.out.println("One parameter: " + a); }  
// перегрузить метод ovlDemo с двумя параметрами  
типа int. // Третий вариант метода.  
int ovlDemo(int a, int b)  
{ System.out.println("Two parameters: " + a + " " +  
b) ; return a + b; } }
```

# Перегрузка конструкторов

```
class MyClass
{ int x;
// Конструкторы перегружаются разными способами.
MyClass()
{ System.out.println("Inside MyClass() .");
x = 0 ; }
MyClass(int i)
{ System.out.println("Inside MyClass(int) . ");
x = i; }
MyClass(double d)
{ System.out.println("Inside MyClass(double) .");
x = (int) d; }
MyClass(int i, int j)
{ System.out.println("Inside MyClass(int, int) .");
x = i * j; }
}
```

# Перегрузка конструкторов

```
class OverloadConsDemo
{
    public static void main(String args[])
    {
        MyClass t1 = new MyClass();
        MyClass t2 = new MyClass(88);
        MyClass t3 = new MyClass(17.23);
        MyClass t4 = new MyClass(2, 4);
        System.out.println("t1.x: " + t1.x);
        System.out.println("t2.x: " + t2.x);
        System.out.println("t3.x: " + t3.x);
        System.out.println("t4.x: " + t4.x);
    }
}
```

# Перегрузка конструкторов

Inside MyClass().

Inside MyClass(int).

Inside MyClass(double).

Inside MyClass(int, int).

t1.x: 0

t2.x: 88

t3.x: 17.23

t4.x: 8

# Перегрузка конструкторов

```
// Инициализация одного объекта посредством другого,  
class Summation { int sum;  
// построить объект из целочисленного значения  
Summation(int num) {  
    sum = 0;  
    for(int i=1; i <= num; i++)  
        sum += i;  
}  
// Построение одного объекта из другого.  
Summation(Summation ob) { sum = ob.sum;  
}  
class SumDemo { public static void main(String args[])  
{  
    Summation s1 = new Summation(5);  
    Summation s2 = new Summation(s1);  
    System.out.println("s1.sum: " + s1.sum);  
    System.out.println("s2.sum: " + s2.sum);  
}}
```

# Применение рекурсии

- Если метод вызывает самого себя, то такой процесс называется рекурсией, а метод, вызывающий самого себя, — рекурсивным.

```
class Factorial {  
    // Рекурсивный метод  
    int factR(int n)  
    {  
        int result;  
        if(n==1) return 1;  
        else return factR(n-1) * n;}  
    // Итеративный способ  
    int factI(int n)  
    {  
        int t, result;  
        result = 1;  
        for(t=1; t <= n; t++)  
            result *= t;  
        return result; } }
```



# Применение рекурсии

```
class Recursion {  
    public static void main(String args[]) {  
        Factorial f = new Factorial();  
        System.out.println("Factorials using recursive  
method.");  
        System.out.println("Factorial of 5 is " +  
f.factR(5));  
        System.out.println("Factorials using iterative  
method.");  
        System.out.println("Factorial of 5 is " +  
f.factI(5));  
    }  
}
```

Factorials using recursive method.

Factorial of 5 is 120

Factorials using iterative method.

Factorial of 5 is 120

# Использование ключевого слова `static`

- Если член класса объявляется как `static`, он становится доступным до создания любых объектов своего класса и без ссылки на какой-нибудь объект.
- С помощью ключевого слова `static` можно объявлять как переменные, так и методы.
- Наиболее характерным примером члена типа `static` служит метод `main()`, который объявляется таковым потому, что он должен вызываться виртуальной машиной Java в самом начале выполняемой программы.

# Использование ключевого слова `static`

```
// Применение статической переменной,  
class StaticDemo  
{    int x; // обычная переменная экземпляра  
    static int y;  
    // статическая переменная — это одна копия,  
    совместно используемая всеми объектами.  
    // вернуть сумму значений переменной экземпляра x  
    и статической переменной y.  
    int sum()  
    {  
        return x + y;  
    }  
}
```

# Использование ключевого слова `static`

```
class SDemo {  
    public static void main(String args[]) {  
        StaticDemo obi = new StaticDemo();  
        StaticDemo ob2 = new StaticDemo();  
        // У каждого объекта имеется своя копия переменной,  
        ob1.x = 10;  
        ob2.x = 20;  
        System.out.println("Of course, ob1.x and ob2.x " +  
"are independent.");  
        System.out.println("obi.x: " + ob1.x + "\tob2.x: "  
+ ob2.x);  
        // Все объекты совместно пользуются одной общей  
        // копией статической переменной.
```

# Использование ключевого слова `static`

```
System.out.println("The static variable y is shared.");  
StaticDemo.y = 19;  
System.out.println("Set StaticDemo.y to 19.");  
System.out.println("ob1.sum() : " + ob1.sum());  
System.out.print("\tob2.sum(): " + ob2.sum());  
StaticDemo.y = 100;  
System.out.println("Change StaticDemo.y to 100");  
System.out.println("ob1.sum() : " + ob1.sum());  
System.out.print("\tob2.sum(): " + ob2.sum());}}
```

Of course, `obl.x` and `ob2.x` are independent,

`obl.x: 10`            `ob2.x: 20`

The static variable `y` is shared.

Set `StaticDemo.y` to 19.

`obi.sum(): 29`    `ob2.sum(): 39`

Change `StaticDemo.y` to 100

`obi.sum(): 110`    `ob2.sum(): 120`

# Использование ключевого слова `static`

- В методе типа `static` допускается непосредственный вызов только других методов типа `static`.
- Для метода типа `static` непосредственно доступными оказываются только другие данные типа `static`, определенные в его классе.
- В методе типа `static` должна отсутствовать ссылка `this`.

```
class StaticError
{
    int denom = 3;    // обычная переменная экземпляра
    static int val = 1024; // статическая переменная
    static int valDivDenom()
    {
        return val/denom; // не подлежит компиляции! К
        нестатическим переменным нельзя обращаться из
        статического метода.
    }
}
```

# Использование ключевого слова `static`

Статический блок выполняется при первой загрузке класса, еще до того, как класс будет использован для каких-нибудь других целей.

```
class StaticBlock
{
    static double rootOf2;
static // Этот блок выполняется при загрузке класса.
{
    System.out.println("Inside static block.");
    rootOf2 = Math.sqrt(2.0);}
StaticBlock(String msg)
    { System.out.println (msg) ; } }
class SDemo3 {public static void main(String args[]) {
StaticBlock ob = new StaticBlock("Inside Constructor.");
System.out.println("Sq.root of 2 is " +
StaticBlock.rootOf2);}}
```

Inside static block.

Inside Constructor .

Square root of 2 is 1.4142135623730951

# Использование аргументов переменной длины

```
class VarArgs
{
    // Метод vaTest() с аргументами переменной длины,
    static void vaTest(int ... v)
    {
        System.out.println("Number of args: " + v.length);
        System.out.println("Contents: ") ;
        for(int i=0; i < v.length; i++)
            System.out.println(" arg " + i + ": " + v[i]);
    }
    public static void main(String args[])
    {
        vaTest(10);           // 1 аргумент
        vaTest(1, 2, 3);      // 3 аргумента
        vaTest();              // без аргументов
    }
}
```



# Использование аргументов переменной длины

Number of args: 1

Contents:

arg 0: 10

Number of args: 3

Contents:

arg 0: 1

arg 1: 2

arg 2: 3

Number of args: 0

Contents:

# Использование аргументов переменной длины

```
class VarArgs2 {  
    // Здесь msg - обычный аргумент, v - аргумент перемен.длины.  
    static void vaTest(String msg, int ... v) {  
        System.out.println(msg + v.length);  
        System.out.println("Contents: ") ;  
        for(int i=0; i < v.length; i++)  
            System.out.println(" arg " + i + ": " + v[i]);  
        System.out.println();  
    }  
  
    public static void main(String args[])  
    {  
        vaTest("One vararg: ", 10);  
        vaTest("Three varargs: ", 1, 2, 3);  
        vaTest("No varargs: ");  
    }  
}
```

# Использование аргументов переменной длины

One vararg: 1

Contents:

arg 0: 10

Three varargs: 3

Contents:

arg 0: 1

arg 1: 2

arg 2: 3

No varargs: 0

Contents:

# Использование аргументов переменной длины

- Аргумент переменной длины должен быть указан последним.
- Аргументы переменной длины можно указать в методе только один раз

```
int dolt(int a, int b, double c, int ... vals,  
boolean stopFlag)
```

```
int dolt(int a, int b, double c, int ... vals,  
double ... morevals)
```

# Использование аргументов переменной длины

```
class VarArgs4 {  
    // Использование аргумента переменной длины типа int.  
    static void vaTest(int ... v) {  
        // ...  
    }  
    // Использование аргумента переменной длины типа boolean.  
    static void vaTest(boolean ... v) {  
        // ...  
    }  
    public static void main(String args[])  
    {  
        vaTest(1, 2, 3); // OK  
        vaTest(true, false, false); // OK  
        vaTest(); // Ошибка вследствие неоднозначности!  
    }  
}
```