Java
Методы с переменным числом параметров

Лекция #15

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

Содержание

- Методы с переменным числом параметров
- Перегрузка методов с переменным числом параметров
- Аргументы - экземпляры класса
- Аргументы - Object
- Примеры

Методы с переменным числом параметров

Методы с переменным числом параметров

Для того, чтобы задать переменное число аргументов в методе на языке Java используется оператор многоточие (…).

```
public void testMethod(double ... d) {

    //код метода
}
```

Работа с аргументом переменной длины производится аналогично работе с массивом.

Методы с переменным числом параметров

Чтобы узнать количество аргументов, переданных методу, необходимо воспользоваться свойством:

```
d.length
```

Получение значения, например, второго аргумента производится так (нумерация начинается с нуля):

```
d[1]
```

Методы с переменным числом параметров

Следует отдельно отметить, что в сигнатуре метода возможно использовать **только один** аргумент переменной длины!
Иначе возникнет ошибка.

Аргумент переменной длины можно использовать в сочетании с обычными параметрами, но важно соблюдать условие: аргумент переменной длины должен находится **на последнем месте** в списке параметров метода, иначе будет выведена ошибка и программа не скомпилируется.

Методы с аргументами переменной длины можно перегружать.

Методы с переменным числом параметров. Пример

```
public class Main{

    public static void printArgument(int... k) {
        System.out.println("Количество аргументов: " + k.length);

        for(int i = 0; i < k.length; i++) {
            System.out.println(k[i]);
        }
    }


    public static void main(String[] args) {

        printArgument(1,2,3,4,5);
    }
}
```

Методы с аргументами переменной длины можно перегружать

```
public static void printArgument( String message, int ... k) {

System.out.println(message);

    for(int i = 0; i < k.length; i++) {
        System.out.println(k[i]);
    }}

public static void printArgument( String ... s) {

    System.out.println("Количество аргументов: " + s.length);

    for (int i = 0; i < s.length; i++) {
        System.out.println(s[i]);
    }}
```

Пример. Максимум из произвольного числа аргументов

```
static double Max(double ... D)
{
    double max;
    if (D.length == 0) return 0;
    max = D[0];
    for (int i=0; i<D.length; i++)
        if (max < D[i]) max = D[i];
    return max;
}
```

```
System.out.println(Max(1,2,3,4,7,6,5));
```

Пример. Сортировка и записывание в массив

```
static float[] ToSortArrayFloat(float ... F) {
    float t; // вспомогательная переменная
    if (F.length<2) return F;

    // сортировка вставкой
    for (int i=0; i<F.length-1; i++)
        for (int j=i; j>=0; j--)
            if (F[j]<F[j+1])
            {
                t = F[j];
                F[j] = F[j+1];
                F[j+1] = t;
            }
    return F;
}
```

```
import java.util.Arrays;
…
float a[]=ToSortArrayFloat(1,7,2,6,3,5,4);
System.out.println(Arrays.toString(a));
```

Пример. Преобразование в двумерный массив. Часть 1

Объявление метода, который получает последовательность аргументов переменной длины и возвращает двумерный массив, сформированный из этих аргументов. Размерность результирующего массива m×n задается обычными аргументами.

```
static int[][] ConvertToArray2(int m, int n, int ... params) {

  int[][] A = new int[m][n]; // результирующий массив
    int row, col;

   // обнулить массив A
   for (int i=0; i<m; i++)
       for (int j=0; j<n; j++)
           A[i][j]=0;
```

Пример. Преобразование в двумерный массив. Часть 2

```
    int len = params.length;
    if (len>m*n) len=m*n;


    // сформировать массив A
    for (int i=0; i<len; i++)
    {
        row = i/n;
        col = i-row*n;
        A[row][col]=params[i];
    }


return A;
}
```

```
int[][] a
=ConvertToArray2(2,3,1,2,3,4,5,6);
System.out.println(a[1][2]); //6
```

Пример. Аргументы – экземпляры класса

```
class Book {
    String title; // заголовок книги
    String author; // название автора
    float price; // стоимость книги

    static Book GetMaxCostBook(Book ... B) {
        Book bk=null;

        if (B.length==0)
            return bk;

        bk = B[0];
        for (int i=1; i<B.length; i++)
            if (bk.price<B[i].price)
                bk = B[i];

        return bk;
    }}
```

```
Book needBook;
needBook = GetMaxCostBook(B1,B2,B3,B4);
```

Метод принимающий аргументы любого типа

В этом случае объявляется метод с аргументами переменной длины типа Object.
Как известно, в Java все классы являются производными от общего корневого класса Object.

```
// метод, который принимает аргументы переменной длины типа Object
static void ShowInfo(Object ... objects) {
    String s;

    // вывод информации об аргументах, переданных в метод
    for (Object o : objects)
        System.out.print(o.toString()+" ");
    System.out.println();
}
```

Метод принимающий аргументы любого типа

```
ShowInfo(1,2,3);

ShowInfo(3.14,9.81,2.718);

ShowInfo(Math.PI, Math.E);

ShowInfo('A','B','C','D');

ShowInfo("qwerty","ABC");
```

```
1 2 3

3.14 9.81 2.718

3.141592653589793 2.718281828459045

A B C D

qwerty ABC
```

Два способа для передачи переменной или объекта класса в функцию

Два способа передачи параметров в функцию

Передача по значению

В этом случае **значение аргумента копируется в формальный параметр** функции. Поскольку создается копия аргумента в функции, то все изменения над копией не повлияют на значение аргумента.

Передача по ссылке (по адресу)

В этом случае параметру передается ссылка на аргумент, который используется при вызове. По этой ссылке есть доступ к аргументу.
Таким образом, все изменения, сделанные в теле функции над значением параметра, будут изменять значение аргумента который был передан в функцию.

Два способа передачи параметров в функцию

Передача по значению

В этом случае **значение аргумента копируется в формальный параметр** функции. Поскольку создается копия аргумента в функции, то все изменения над копией не повлияют на значение аргумента.

```
public class test {


    public static void main(String[] args) {
        int x = 5;
        int y = byValue(x);
        System.out.println("x = "+x+" y = "+y);
    }
    public static int byValue(int x) {
        x += 10;
        return x;
    }}
```

x = 5 y = 15

Два способа передачи параметров в функцию

```
class Example {
    public int x;
    {
        x = 5;
    }
    public Example() {    }

    public Example(Example x) {
        x = x;
    }

    public Example(int x) {
        this.x = x;
    }
    public void setX(int i) {
        this.x = i;
    }

    public int getX() {
        return x;
    }
}
```

Два способа передачи параметров в функцию

```
public class test {

public static void main(String[] args) {

        Example x = new Example();

        Example y = new Example(x);
        System.out.println(" x = "+x.getX());
        System.out.println(" y = "+y.getX());

        x.setX(15);
        System.out.println(" x = "+x.getX());
        System.out.println(" y = "+y.getX());
    }
}
```

```
x = 5
y = 5
```

```
x = 15
y = 5
```

Два способа передачи параметров в функцию

Java передает все по значению

Для примитивных типов — вы передаете копию текущего значения, для ссылок на объекты — вы передаете копию ссылки (дистанционного управления).

Вы никогда не передаете объект.

Все объекты хранятся в куче. Всегда.

Спасибо за внимание!