

Введение в программирование на Java

Лекция 5. Процедурное программирование. Методы.

Виталий Олегович Афанасьев

10 февраля 2025

Сравнение ссылочных типов

Сравнение ссылочных типов (1)

Сравнение ссылочных типов через оператор `==` сравнивает ссылки, а не значения.

```
1 int[] firstArr = { 1, 2, 3 };  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11
```

Сравнение ссылочных типов (1)

Сравнение ссылочных типов через оператор `==` сравнивает ссылки, а не значения.

```
1 int[] firstArr = { 1, 2, 3 };  
2 int[] secondArr = { 1, 2, 3 };  
3  
4  
5  
6  
7  
8  
9  
10  
11
```

Сравнение ссылочных типов (1)

Сравнение ссылочных типов через оператор `==` сравнивает ссылки, а не значения.

```
1 int[] firstArr = { 1, 2, 3 };  
2 int[] secondArr = { 1, 2, 3 };  
3 boolean eq1 = firstArr == secondArr; // false
```

Сравнение ссылочных типов (1)

Сравнение ссылочных типов через оператор `==` сравнивает ссылки, а не значения.

```
1 int[] firstArr = { 1, 2, 3 };  
2 int[] secondArr = { 1, 2, 3 };  
3 boolean eq1 = firstArr == secondArr; // false  
4 secondArr = firstArr;
```

Сравнение ссылочных типов (1)

Сравнение ссылочных типов через оператор `==` сравнивает ссылки, а не значения.

```
1 int[] firstArr = { 1, 2, 3 };  
2 int[] secondArr = { 1, 2, 3 };  
3 boolean eq1 = firstArr == secondArr; // false  
4 secondArr = firstArr;  
5 boolean eq2 = firstArr == secondArr; // true
```

Сравнение ссылочных типов (1)

Сравнение ссылочных типов через оператор `==` сравнивает ссылки, а не значения.

```
1 int[] firstArr = { 1, 2, 3 };
2 int[] secondArr = { 1, 2, 3 };
3 boolean eq1 = firstArr == secondArr; // false
4 secondArr = firstArr;
5 boolean eq2 = firstArr == secondArr; // true
6
7 String firstStr = "Hello";
8
9
10
11
```


Сравнение ссылочных типов (1)

Сравнение ссылочных типов через оператор == сравнивает ссылки, а не значения.

```
1 int[] firstArr = { 1, 2, 3 };
2 int[] secondArr = { 1, 2, 3 };
3 boolean eq1 = firstArr == secondArr; // false
4 secondArr = firstArr;
5 boolean eq2 = firstArr == secondArr; // true
6
7 String firstStr = "Hello";
8 String secondStr = scanner.nextLine(); // Вводим строку "Hello"
9
10
11
```

Сравнение ссылочных типов (1)

Сравнение ссылочных типов через оператор == сравнивает ссылки, а не значения.

```
1 int[] firstArr = { 1, 2, 3 };
2 int[] secondArr = { 1, 2, 3 };
3 boolean eq1 = firstArr == secondArr; // false
4 secondArr = firstArr;
5 boolean eq2 = firstArr == secondArr; // true
6
7 String firstStr = "Hello";
8 String secondStr = scanner.nextLine(); // Вводим строку "Hello"
9 boolean eq3 = firstStr == secondStr; // Может быть как false, так
   и true (если строка будет кэширована)
10
11
```

Сравнение ссылочных типов (1)

Сравнение ссылочных типов через оператор == сравнивает ссылки, а не значения.

```
1 int[] firstArr = { 1, 2, 3 };
2 int[] secondArr = { 1, 2, 3 };
3 boolean eq1 = firstArr == secondArr; // false
4 secondArr = firstArr;
5 boolean eq2 = firstArr == secondArr; // true
6
7 String firstStr = "Hello";
8 String secondStr = scanner.nextLine(); // Вводим строку "Hello"
9 boolean eq3 = firstStr == secondStr; // Может быть как false, так
    и true (если строка будет закеширована)
10 secondStr = firstStr;
11
```

Сравнение ссылочных типов (1)

Сравнение ссылочных типов через оператор == сравнивает ссылки, а не значения.

```
1 int[] firstArr = { 1, 2, 3 };
2 int[] secondArr = { 1, 2, 3 };
3 boolean eq1 = firstArr == secondArr; // false
4 secondArr = firstArr;
5 boolean eq2 = firstArr == secondArr; // true
6
7 String firstStr = "Hello";
8 String secondStr = scanner.nextLine(); // Вводим строку "Hello"
9 boolean eq3 = firstStr == secondStr; // Может быть как false, так
    и true (если строка будет закеширована)
10 secondStr = firstStr;
11 boolean eq4 = firstStr == secondStd; // true
```

Сравнение ссылочных типов (2)

Вместо оператора `==` для сравнения значений следует использовать методы типа `equals`.

```
1 int[] firstArr = { 1, 2, 3 };
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

Сравнение ссылочных типов (2)

Вместо оператора `==` для сравнения значений следует использовать методы типа `equals`.

```
1 int[] firstArr = { 1, 2, 3 };  
2 int[] secondArr = { 1, 2, 3 };  
3  
4  
5  
6  
7
```

Сравнение ссылочных типов (2)

Вместо оператора `==` для сравнения значений следует использовать методы типа `equals`.

```
1 int[] firstArr = { 1, 2, 3 };  
2 int[] secondArr = { 1, 2, 3 };  
3 boolean eq1 = Arrays.equals(firstArr, secondArr); // true  
4  
5  
6  
7
```

Сравнение ссылочных типов (2)

Вместо оператора `==` для сравнения значений следует использовать методы типа `equals`.

```
1 int[] firstArr = { 1, 2, 3 };
2 int[] secondArr = { 1, 2, 3 };
3 boolean eq1 = Arrays.equals(firstArr, secondArr); // true
4
5 String firstStr = "Hello";
6
7
```


Сравнение ссылочных типов (2)

Вместо оператора `==` для сравнения значений следует использовать методы типа `equals`.

```
1 int[] firstArr = { 1, 2, 3 };
2 int[] secondArr = { 1, 2, 3 };
3 boolean eq1 = Arrays.equals(firstArr, secondArr); // true
4
5 String firstStr = "Hello";
6 String secondStr = scanner.nextLine(); // Вводим строку "Hello"
7
```

Сравнение ссылочных типов (2)

Вместо оператора `==` для сравнения значений следует использовать методы типа `equals`.

```
1 int[] firstArr = { 1, 2, 3 };
2 int[] secondArr = { 1, 2, 3 };
3 boolean eq1 = Arrays.equals(firstArr, secondArr); // true
4
5 String firstStr = "Hello";
6 String secondStr = scanner.nextLine(); // Вводим строку "Hello"
7 boolean eq2 = firstStr.equals(secondStr); // true
```

Процедурное программирование

Постановка проблемы

```
1 public static void main(String[] args) {
2     Scanner scanner = new Scanner(System.in);
3     double a = scanner.nextDouble();
4     double b = scanner.nextDouble();
5     double c = scanner.nextDouble();
6
7     if (Math.abs(a) < EPS) {
8         if (Math.abs(b) < EPS) {
9             if (Math.abs(c) < EPS) {
10                 System.out.println("Бесконечно много решений");
11             } else {
12                 System.out.println("Нет решений в вещественных числах");
13             }
14         } else {
15             double x = -c / b;
16             System.out.printf("Одно решение: X = %.6f\n", x);
17         }
18     } else {
19         double discriminant = b * b - 4 * a * c;
20         if (discriminant > EPS) {
21             double x1 = (-b - Math.sqrt(discriminant)) / (2 * a);
22             double x2 = (-b + Math.sqrt(discriminant)) / (2 * a);
23             System.out.printf("Два решения: X1 = %.6f, X2 = %.6f\n", x1, x2);
24         } else if (Math.abs(discriminant) < EPS) {
25             double x = -b / (2 * a);
26             System.out.printf("Одно решение: X = %.6f\n", x);
27         } else {
28             System.out.println("Нет решений в вещественных числах");
29         }
30     }
31 }
```

Постановка проблемы

```
1 public static void main(String[] args) {
2     Scanner scanner = new Scanner(System.in);
3     double a = scanner.nextDouble();
4     double b = scanner.nextDouble();
5     double c = scanner.nextDouble();
6
7     if (Math.abs(a) < EPS) {
8         if (Math.abs(b) < EPS) {
9             if (Math.abs(c) < EPS) {
10                 System.out.println("Бесконечно много решений");
11             } else {
12                 System.out.println("Нет решений в вещественных числах");
13             }
14         } else {
15             double x = -c / b;
16             System.out.printf("Одно решение: X = %.6f\n", x);
17         }
18     } else {
19         double discriminant = b * b - 4 * a * c;
20         if (discriminant > EPS) {
21             double x1 = (-b - Math.sqrt(discriminant)) / (2 * a);
22             double x2 = (-b + Math.sqrt(discriminant)) / (2 * a);
23             System.out.printf("Два решения: X1 = %.6f, X2 = %.6f\n", x1, x2);
24         } else if (Math.abs(discriminant) < EPS) {
25             double x = -b / (2 * a);
26             System.out.printf("Одно решение: X = %.6f\n", x);
27         } else {
28             System.out.println("Нет решений в вещественных числах");
29         }
30     }
31 }
```

Постановка проблемы

```
1 public static void main(String[] args) {
2     Scanner scanner = new Scanner(System.in);
3     double a = scanner.nextDouble();
4     double b = scanner.nextDouble();
5     double c = scanner.nextDouble();
6
7     if (Math.abs(a) < EPS) {
8         if (Math.abs(b) < EPS) {
9             if (Math.abs(c) < EPS) {
10                 System.out.println("Бесконечно много решений");
11             } else {
12                 System.out.println("Нет решений в вещественных числах");
13             }
14         } else {
15             double x = -c / b;
16             System.out.printf("Одно решение: X = %.6f\n", x);
17         }
18     } else {
19         double discriminant = b * b - 4 * a * c;
20         if (discriminant > EPS) {
21             double x1 = (-b - Math.sqrt(discriminant)) / (2 * a);
22             double x2 = (-b + Math.sqrt(discriminant)) / (2 * a);
23             System.out.printf("Два решения: X1 = %.6f, X2 = %.6f\n", x1, x2);
24         } else if (Math.abs(discriminant) < EPS) {
25             double x = -b / (2 * a);
26             System.out.printf("Одно решение: X = %.6f\n", x);
27         } else {
28             System.out.println("Нет решений в вещественных числах");
29         }
30     }
31 }
```

Процедурное программирование

При процедурном программировании программа разбивается на подпрограммы (subroutines, a.k.a. процедуры, функции, методы). Подпрограммы вызывают друг друга, чтобы решить итоговую задачу.

Цели разбиения кода на подпрограммы:

- **Декомпозиция (модульность).** Программа разбивается на более мелкие части, которые проще разрабатывать, тестировать, читать, отлаживать и т.п. Каждая подпрограмма выполняет свою задачу и только её. Декомпозиция также позволяет **локализовать данные**, необходимые каждой подпрограмме.
- **Переиспользование кода.** Вместо дублирования повторяющихся участков кода, можно сгруппировать их в одну подпрограмму, которую вызвать несколько раз из основной программы. Пример: функции стандартной библиотеки `Math.abs`, `Math.sqrt`, и т.д.

Методы стандартной библиотеки

С несколькими методами мы уже знакомы:

- `Math.sqrt`, вычисляющий квадратный корень числа (и прочие методы класса `Math`).
- `println` на объекте `System.out`, выводящий текст на экран.
- `Character.isDigit`, проверяющий является ли символ цифрой.
- `Integer.parseInt`, переводящий строку в число типа `int`.
- `Arrays.toString`, переводящий массив в строку.

Пример выполнения программы с несколькими методами

```
1 public class Main {
2     public static void main(String[] args) {
3         System.out.println("Первая строка в методе main");
4
5         sayHello();
6         sayHello();
7
8         System.out.println("Вторая строка в методе main");
9         sayHello();
10
11        System.out.println("Третья строка в методе main");
12    }
13
14    public static void sayHello() {
15        System.out.println("Привет");
16    }
17 }
```

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

Первая строка в методе main

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

Первая строка в методе main

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

Первая строка в методе main

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

Первая строка в методе main
Привет

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

Первая строка в методе main
Привет

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

Первая строка в методе main
Привет

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

Первая строка в методе main
Привет

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

Первая строка в методе main
Привет
Привет

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

```
Первая строка в методе main  
Привет  
Привет
```

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

Первая строка в методе main
Привет
Привет
Вторая строка в методе main

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

```
Первая строка в методе main  
Привет  
Привет  
Вторая строка в методе main
```

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

Первая строка в методе main
Привет
Привет
Вторая строка в методе main

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

```
Первая строка в методе main  
Привет  
Привет  
Вторая строка в методе main  
Привет
```

Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

```
Первая строка в методе main  
Привет  
Привет  
Вторая строка в методе main  
Привет
```


Пример выполнения программы с несколькими методами

```
1 public class Main {  
2     public static void main(String[] args) {  
3         System.out.println("Первая строка в методе main");  
4  
5         sayHello();  
6         sayHello();  
7  
8         System.out.println("Вторая строка в методе main");  
9         sayHello();  
10  
11        System.out.println("Третья строка в методе main");  
12    }  
13  
14    public static void sayHello() {  
15        System.out.println("Привет");  
16    }  
17 }
```

```
Первая строка в методе main  
Привет  
Привет  
Вторая строка в методе main  
Привет  
Третья строка в методе main
```

Параметры и аргументы

Параметр — переменная, используемая при объявлении метода.

Аргумент — фактическое значение параметра при вызове метода.

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     int y = 42;  
4     printSum(x, y); // 52  
5     printSum(1, 2); // 3  
6 }  
7  
8 public static void printSum(int first, int second) {  
9     System.out.println(first + second);  
10 }
```

`first` и `second` — **параметры** метода `printSum`.

`x` и `y` — **аргументы** при первом вызове метода `printSum`.

`1` и `2` — **аргументы** при втором вызове метода `printSum`.

Возврат значения из метода

Методы могут возвращать значения в качестве результата выполнения. Для этого используется ключевое слово `return`.

При этом у метода должен быть указан тип возвращаемого значения.

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     int y = 42;  
4     int result1 = sum(x, y); // 52  
5     int result2 = sum(1, 2); // 3  
6 }  
7  
8 public static int sum(int first, int second) {  
9     return first + second;  
10 }
```

Тип void

Методы, которые не имеют возвращаемого значения, имеют тип void.

```
1 public static void main(String[] args) {
2     printPositiveOrNegative(123); // Напечатает Положительное
3     printPositiveOrNegative(-100); // Напечатает Отрицательное
4     printPositiveOrNegative(0); // Напечатает Ноль
5 }
6
7 public static void printPositiveOrNegative(int x) {
8     if (x > 0) {
9         System.out.println("Положительное");
10    } else if (x < 0) {
11        System.out.println("Отрицательное");
12    } else {
13        System.out.println("Ноль");
14    }
15 }
```

Оператор return без значения (1)

В методах с типом `void` можно использовать оператор `return` без значения.

Это полезно, если нужно прервать выполнение метода при каком-то условии.

```
1 public static void main(String[] args) {
2     printSumIfPositive(10, -2); // Напечатает 8
3     printSumIfPositive(10, -20); // Не напечатает ничего
4 }
5
6 public static void printSumIfPositive(int first, int second) {
7     int sum = first + second;
8     if (sum <= 0) {
9         return;
10    }
11    System.out.println(sum);
12 }
```

Оператор return без значения (2)

Одно из распространённых применений — при валидации входных значений.

```
1 public static void main(String[] args) {  
2     Scanner scanner = new Scanner(System.in);  
3     int x = scanner.nextInt();  
4     if (x < 0) {  
5         System.out.println("Ожидалось неотрицательное число");  
6         return;  
7     }  
8     ...  
9 }
```

Способы передачи параметров

Различают (как минимум) два способа передачи параметров:

- **По значению (pass by value)**. В функцию передаётся копия значения. Изменение параметра никак не влияет на аргумент.
- **По ссылке (pass by reference)**. В функцию передаётся ссылка на значение. Изменение параметра меняет аргумент.

Передача параметров по значению

В Java используется только **передача по значению**.

Присваивание значения параметру никак не повлияет на аргумент при вызове.

```
1 public static void main(String[] args) {  
2     int x = 42;  
3     System.out.println(x); // 42  
4     changeParam(x);  
5     System.out.println(x); // 42  
6 }  
7  
8 public static void changeParam(int param) {  
9     param = 100500;  
10 }
```


Передача параметров по ссылке

Пример языка, где используется передача параметров по ссылке — C++.

```
1 void swap(int& first, int& second) {  
2     int tmp = first;  
3     first = second;  
4     second = tmp;  
5 }  
6  
7 int main() {  
8     int x = 42;  
9     int y = 10;  
10    // x == 42, y == 10  
11    swap(x, y);  
12    // x == 10, y == 42  
13    return 0;  
14 }
```

Передача значений ссылочных типов как аргументов

При использовании ссылочных типов в качестве аргументов, передаётся **ссылка** на значение, а не его копия.

Таким образом, можно реализовать методы, меняющие состояние объекта. Пример из стандартной библиотеки — `Arrays.sort`.

```
1 public static void main(String[] args) {
2     int[] arr = { 1, 2, 3, 4, 5 };
3     System.out.println(arr[0]); // 1
4     setToFirst(arr, 100);
5     System.out.println(arr[0]); // 100
6     setToFirst(arr, 42);
7     System.out.println(arr[0]); // 42
8 }
9
10 public static void setToFirst(int[] arr, int value) {
11     arr[0] = value;
12 }
```

Variadic arguments (1)

Java позволяет объявлять методы с переменным количеством параметров (т.н. variadic arguments, varargs).

Для этого в качестве параметра указывается тип... имя, например `int... values`. Т.е. все переменные параметры имеют один тип.

```
1 public static void main(String[] args) {  
2     printValues(1, 2, 3); // Напечатает [1, 2, 3]  
3     printValues(0, 5, 10, 42, 100, -1, 8); // Напечатает [0, 5,  
        10, 42, 100, -1, 8]  
4     printValues(); // Напечатает []  
5 }  
6  
7 public static void printValues(int... values) {  
8     System.out.println(Arrays.toString(values));  
9 }
```

Variadic arguments (2)

Можно объявить параметры до variadic параметра. Но после — нельзя.

```
1 public static void main(String[] args) {
2     printValues("FIRST", 1, 2, 3); // Напечатает FIRST [1, 2, 3]
3     printValues("SECOND", 0, 5); // Напечатает SECOND [0, 5]
4     printValues("THIRD"); // Напечатает THIRD []
5 }
6
7 public static void printValues(String first, int... other) {
8     System.out.println(first + " " + Arrays.toString(other));
9 }
10
11 // Ошибка: параметр после vararg
12 public static void printValuesBad(int... other, String last)
```

Сигнатура функции

Сигнатура функции — часть объявления функции, которая позволяет отличать одну функцию от другой.

В Java сигнатурой выступает имя метода и типы параметров.

Компилятор запрещает объявлять два метода с одинаковой сигнатурой.

```
1 public static String first(int x) // OK
```

```
2
```

```
3
```

```
4
```

```
5
```

```
6
```

```
7
```

```
8
```

```
9
```

```
10
```

```
11
```

```
12
```

Сигнатура функции

Сигнатура функции — часть объявления функции, которая позволяет отличать одну функцию от другой.

В Java сигнатурой выступает имя метода и типы параметров.

Компилятор запрещает объявлять два метода с одинаковой сигнатурой.

```
1 public static String first(int x) // OK
2 public static String second(int x) // OK
3
4
5
6
7
8
9
10
11
12
```

Сигнатура функции

Сигнатура функции — часть объявления функции, которая позволяет отличать одну функцию от другой.

В Java сигнатурой выступает имя метода и типы параметров.

Компилятор запрещает объявлять два метода с одинаковой сигнатурой.

```
1 public static String first(int x) // OK
2 public static String second(int x) // OK
3
4 public static String first(String s) // OK
```

Сигнатура функции

Сигнатура функции — часть объявления функции, которая позволяет отличать одну функцию от другой.

В Java сигнатурой выступает имя метода и типы параметров.

Компилятор запрещает объявлять два метода с одинаковой сигнатурой.

```
1 public static String first(int x) // OK
2 public static String second(int x) // OK
3
4 public static String first(String s) // OK
5
6 public static String second(int x, int y) // OK
7
8
9
10
11
12
```


Сигнатура функции

Сигнатура функции — часть объявления функции, которая позволяет отличать одну функцию от другой.

В Java сигнатурой выступает имя метода и типы параметров.

Компилятор запрещает объявлять два метода с одинаковой сигнатурой.

```
1 public static String first(int x) // OK
2 public static String second(int x) // OK
3
4 public static String first(String s) // OK
5
6 public static String second(int x, int y) // OK
7
8 public static String first(int x)
```

Сигнатура функции

Сигнатура функции — часть объявления функции, которая позволяет отличать одну функцию от другой.

В Java сигнатурой выступает имя метода и типы параметров.

Компилятор запрещает объявлять два метода с одинаковой сигнатурой.

```
1 public static String first(int x) // OK
2 public static String second(int x) // OK
3
4 public static String first(String s) // OK
5
6 public static String second(int x, int y) // OK
7
8 public static String first(int x)
9 // Ошибка: такая функция уже существует
10
11
12
```

Сигнатура функции

Сигнатура функции — часть объявления функции, которая позволяет отличать одну функцию от другой.

В Java сигнатурой выступает имя метода и типы параметров.

Компилятор запрещает объявлять два метода с одинаковой сигнатурой.

```
1 public static String first(int x) // OK
2 public static String second(int x) // OK
3
4 public static String first(String s) // OK
5
6 public static String second(int x, int y) // OK
7
8 public static String first(int x)
9 // Ошибка: такая функция уже существует
10
11 public static int first(int x)
12
```

Сигнатура функции

Сигнатура функции — часть объявления функции, которая позволяет отличать одну функцию от другой.

В Java сигнатурой выступает имя метода и типы параметров.

Компилятор запрещает объявлять два метода с одинаковой сигнатурой.

```
1 public static String first(int x) // OK
2 public static String second(int x) // OK
3
4 public static String first(String s) // OK
5
6 public static String second(int x, int y) // OK
7
8 public static String first(int x)
9 // Ошибка: такая функция уже существует
10
11 public static int first(int x)
12 // Ошибка: возвращаемый тип не входит в сигнатуру
```

Перегрузка методов

Java позволяет **перегружать** методы, т.е. определять методы с одинаковыми именами, но разными наборами параметров.

```
1 public static void main(String[] args){
2     System.out.println(sum(1, 2)); // 3
3     System.out.println(sum(3, 4, 5)); // 12
4     System.out.println(sum("AB", "BA")); // ABBA
5 }
6 public static int sum(int a, int b) {
7     return a + b;
8 }
9 public static int sum(int a, int b, int c) {
10     return a + b + c;
11 }
12 public static String sum(String a, String b) {
13     return a + b;
14 }
```

Рекурсия

Рекурсия — вызов функции из самой себя.

Пример: факториал числа:

- $n! = 1$, при $n = 0$
- $n! = n * (n - 1)!$, при $n > 0$

```
1 public static void main(String[] args){
2     System.out.println(factorial(0)); // 1
3     System.out.println(factorial(3)); // 6
4     System.out.println(factorial(5)); // 120
5     System.out.println(factorial(10)); // 3628800
6 }
7
8 public static int factorial(int n) {
9     if (n <= 0) {
10         return 1;
11     }
12     return n * factorial(n - 1);
13 }
```

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main


```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - $x = 10$

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - $x = 10$
- foo

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - x = 10
- foo
 - x = 10

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - x = 10
- foo
 - x = 10
 - y = 20

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - x = 10
- foo
 - x = 10
 - y = 20
- bar

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - x = 10
- foo
 - x = 10
 - y = 20
- bar
 - x = 20

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - x = 10
- foo
 - x = 10
 - y = 20
- bar
 - x = 1020

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - x = 10
- foo
 - x = 10
 - y = 20
- bar
 - x = 1020

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - x = 10
- foo
 - x = 10
 - y = 20

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - $x = 10$

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - $x = 10$
- bar

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - $x = 10$
- bar
 - $x = -1$

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - x = 10
- bar
 - x = 999

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - x = 10
- bar
 - x = 999

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - $x = 10$

Стек вызовов

```
1 public static void main(String[] args) {  
2     int x = 10;  
3     foo(x);  
4     bar(-1);  
5     System.out.println(x);  
6 }  
7 public static void foo(int x) {  
8     int y = x * 2;  
9     bar(y);  
10 }  
11 public static void bar(int x) {  
12     x = x + 1000;  
13     System.out.println(x);  
14 }
```

- main
 - $x = 10$

Stack overflow

Если стек вызовов становится слишком глубоким, возникает исключение `StackOverflowError`.

```
1 public class Main {  
2     public static void main(String[] args) {  
3         foo();  
4     }  
5     public static void foo() {  
6         foo();  
7     }  
8 }
```

```
Exception in thread "main" java.lang.StackOverflowError  
    at Main.foo(Main.java:6)  
    at Main.foo(Main.java:6)  
    at Main.foo(Main.java:6)  
    ...
```

Декомпозиция (1)

Декомпозиция очень важна, т.к. сильно упрощает поддержку кода.

Некоторые рекомендации ("Чистый Код", Роберт Мартин):

- **Компактность.**
- **Правило одной операции.**
 - Одна секция на функцию.
 - Один уровень абстракции на функцию.
- **Содержательные имена.**
 - Функции, выполняющие действия, следует именовать глаголами: `println`, `sayHello`, `sort`.
 - Функции, вычисляющие результат (без побочных действий), можно именовать существительными или глаголами: `abs`, `getAbsoluteValue`, `calculateAbsoluteValue`.
 - Параметры и локальные переменные следует именовать существительными: `userName`, `sum`, `velocity`.
 - Функции, параметры, переменные с типом `boolean` следует именовать вопросами с ответом "да/нет": `isDigit`, `isPositive`, `isLowerCase`.

Не нужно превращать рекомендации в карго-культ!

Любым рекомендациям нужно следовать обдуманно.

Декомпозиция (3)

```
1 public static void main(String[] args) {
2     Scanner scanner = new Scanner(System.in);
3     double a = scanner.nextDouble();
4     double b = scanner.nextDouble();
5     double c = scanner.nextDouble();
6     if (Math.abs(a) < EPS) {
7         solveLinear(b, c);
8     } else {
9         solveQuadratic(a, b, c);
10    }
11 }
12 public static void solveLinear(double a, double b) {
13     if (Math.abs(a) < EPS) {
14         ...
15     } else {
16         double x = -b / a;
17         System.out.printf("Одно решение: X = %.6f\n", x);
18     }
19 }
20 public static void solveQuadratic(double a, double b, double c) {
21     double discriminant = b * b - 4 * a * c;
22     if (discriminant > EPS) {
23         double x1 = (-b - Math.sqrt(discriminant)) / (2 * a);
24         double x2 = (-b + Math.sqrt(discriminant)) / (2 * a);
25         System.out.printf("Два решения: X1 = %.6f, X2 = %.6f\n", x1, x2);
26     }
27     ...
28 }
```