



Lesson 3

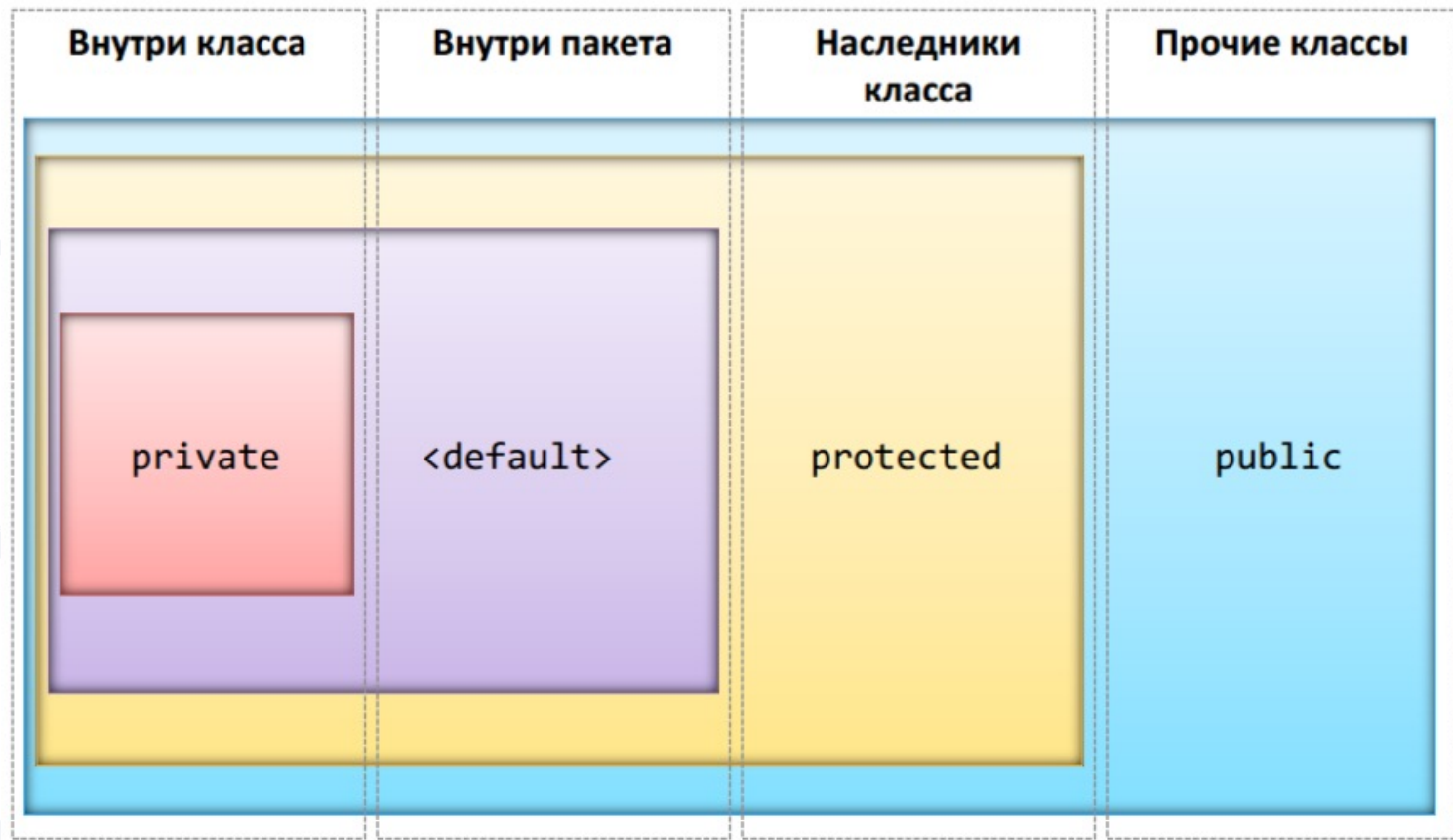
26.05.2022



Operators	Associativity	Type
++ --	Right to left	Unary postfix
++ -- + - ! (type)	Right to left	Unary prefix
/ * %	Left to right	Multiplicative
+ -	Left to right	Additive
< <= > >=	Left to right	Relational
== !=	Left to right	Equality
&	Left to right	Boolean Logical AND
^	Left to right	Boolean Logical Exclusive OR
	Left to right	Boolean Logical Inclusive OR
&&	Left to right	Conditional AND
	Left to right	Conditional OR
?:	Right to left	Conditional
= += -= *= /= %=	Right to left	Assignment



Модификаторы доступа



2 типа преобразований



```
graph TD; A[2 типа преобразований] --> B[Автоматическое преобразование<br/>( неявное )]; A --> C[Приведение типов<br/>( явное )];
```

Автоматическое
преобразование
(неявное)

Приведение
типов
(явное)

byte



short

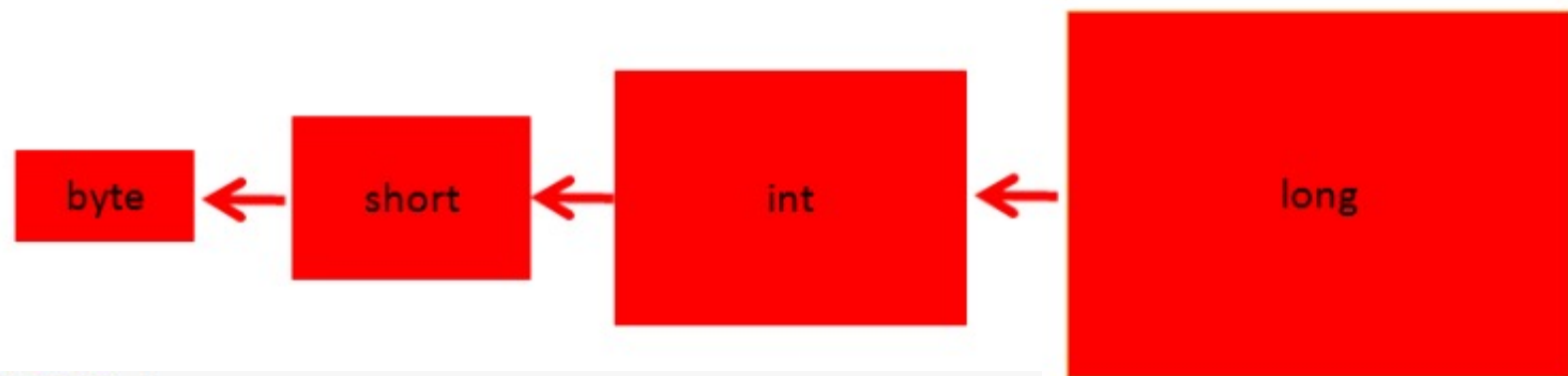


int



long

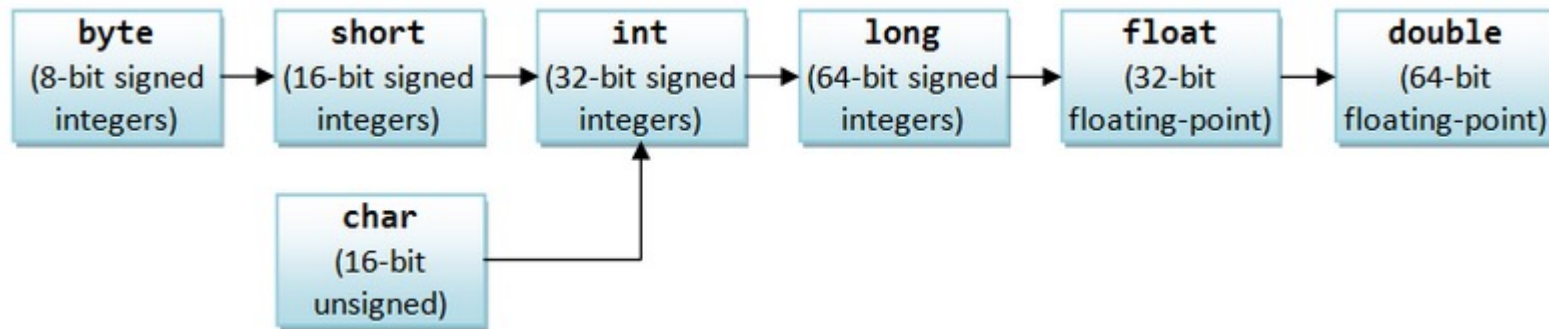
```
1 class Test {  
2  
3     public static void main(String[] args) {  
4         byte a = 15;  
5         int b = a;  
6         System.out.println(b);  
7     }  
8 }
```



```
class Test {  
    public static void main(String[] args) {  
        double a=11.2345;  
        int b=(int)a;  
        System.out.println(b); // в консоли получится число 11  
    }  
}
```

a = (int) b;

(целевой_тип) значение



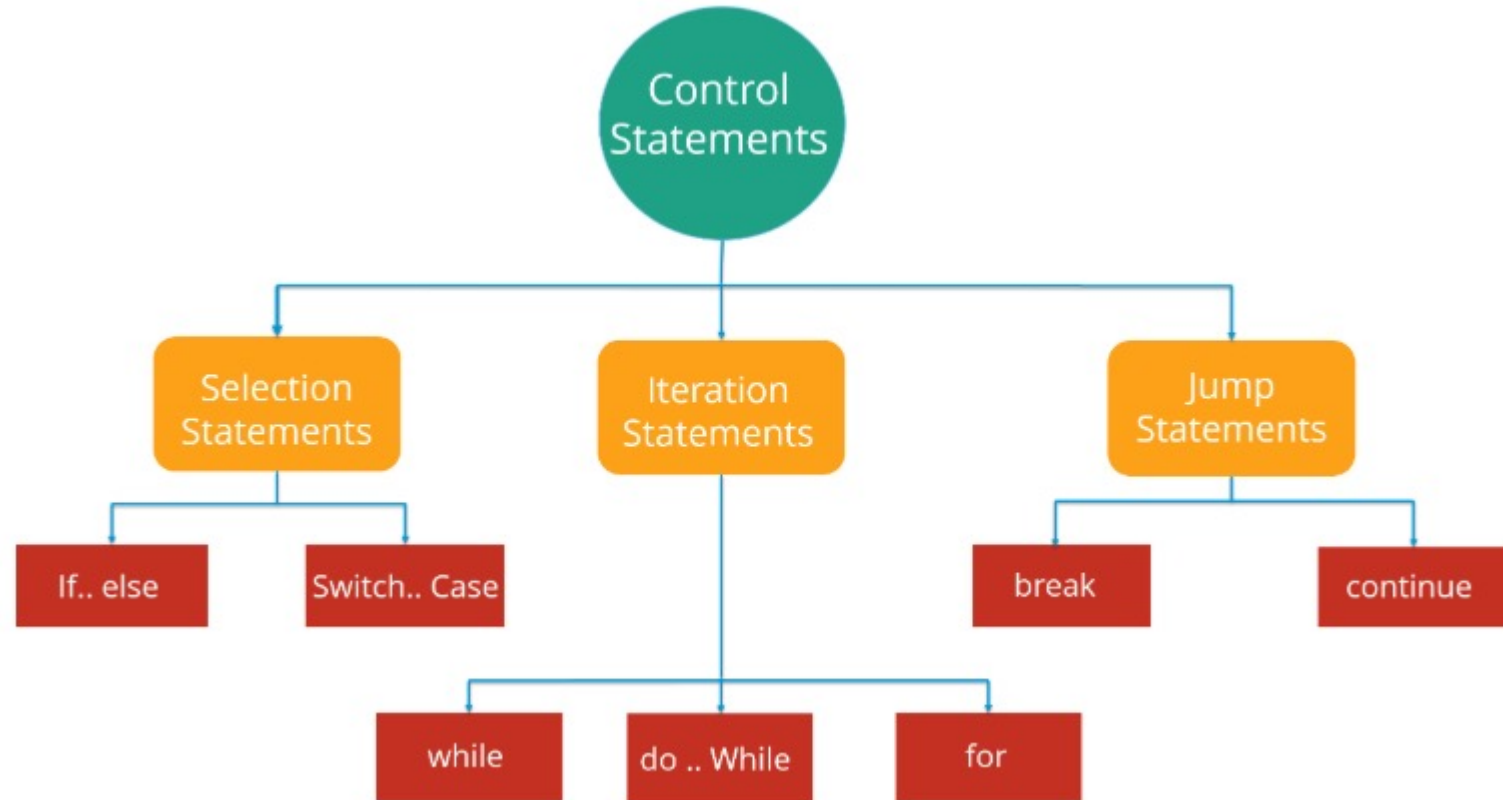
Неявное преобразование типов выполняется в случае если выполняются условия:

Оба типа совместимы

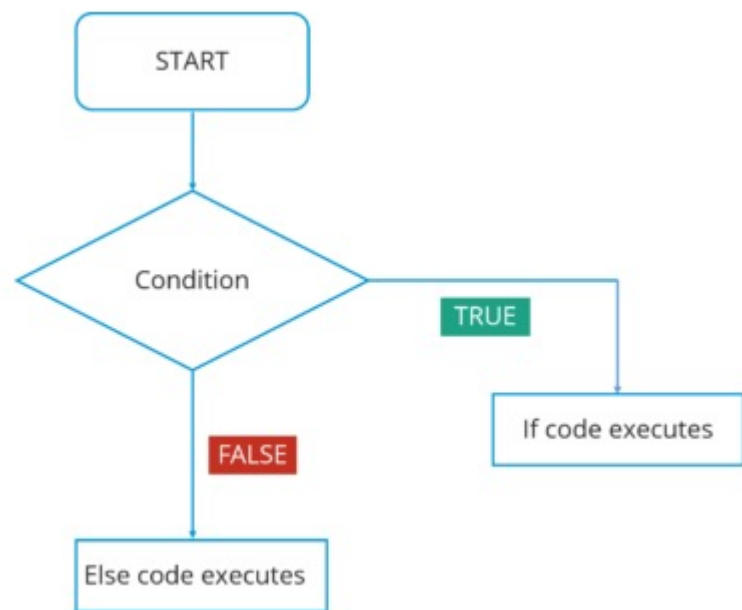
Длина целевого типа больше или равна длине исходного типа

Во всех остальных случаях должно использоваться **явное преобразование типов**.

Java control structure

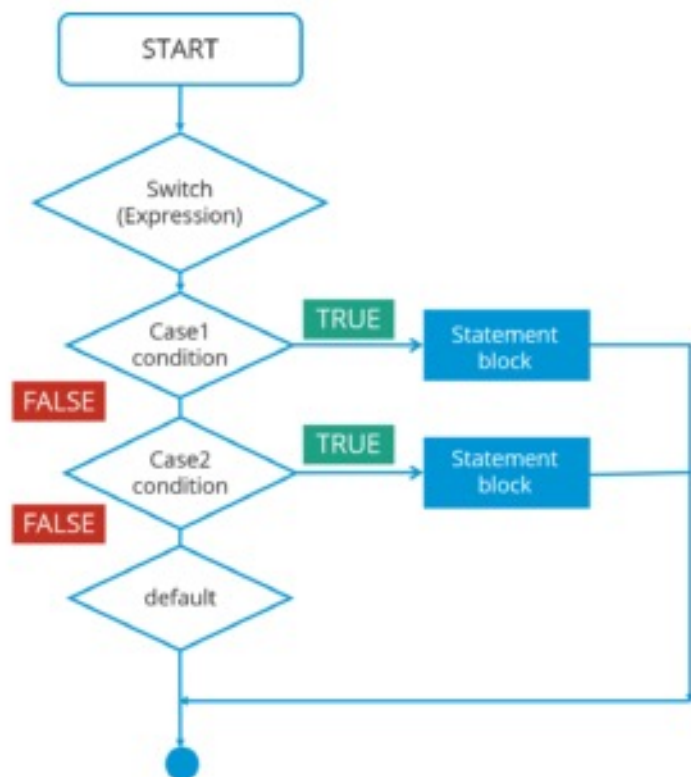


if-else statements



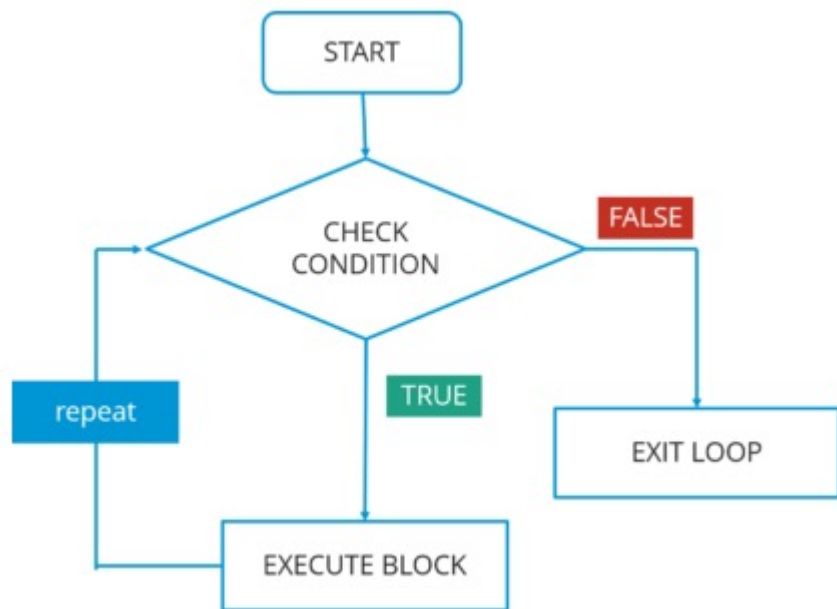
```
1 public class Compare {  
2     int a=10,  
3     int b=5;  
4  
5     if(a>b)  
6     { // if condition  
7         System.out.println(" A is greater than B");  
8     }  
9     else  
10    { // else condition  
11        System.out.println(" B is greater");  
12    }  
13 }
```

Switch case



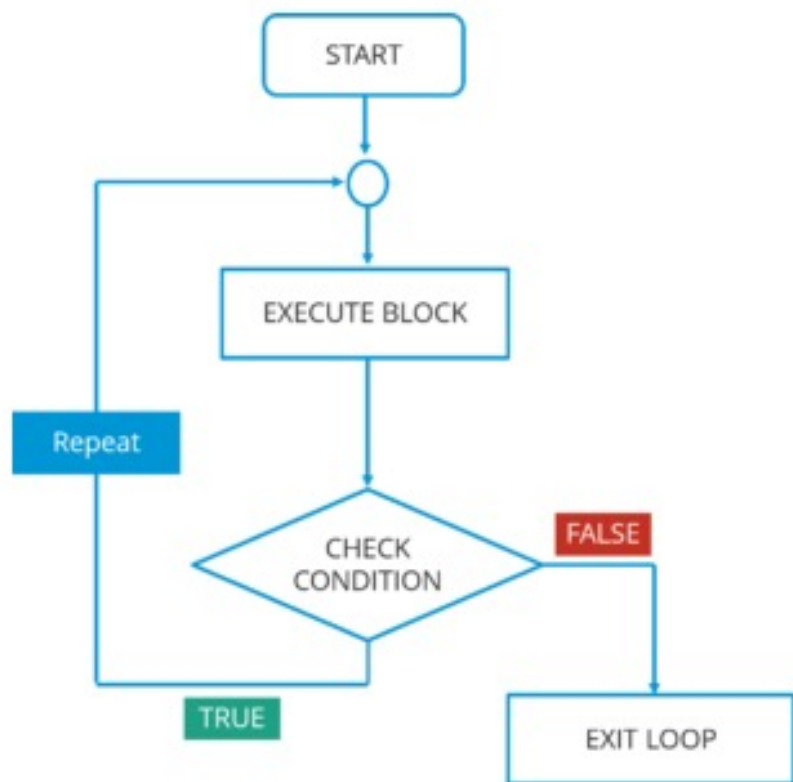
```
1 public class SwitchExample {  
2     int week=7;  
3     String weeknumber;  
4  
5     switch(week){    // switch case  
6     case 1:  
7         weeknumber="Monday";  
8         break;  
9  
10    case2:  
11        weeknumber="tuesday";  
12        break;  
13  
14    case3:  
15        weeknumber="wednesday";  
16        break;  
17  
18    default:    // default case  
19        weeknumber="invalid week";  
20        break;  
21    }  
22    System.out.println(weeknumber);  
23 }  
24 }
```

While statement



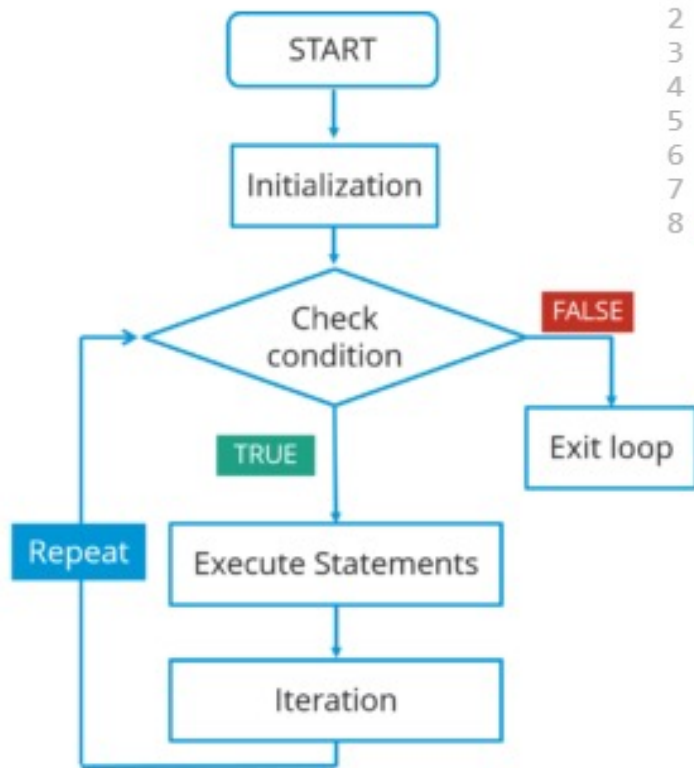
```
1 public class WhileExample {  
2     public static void main(String args[]) {  
3         int a=5;  
4         while(a<10) //while condition  
5         {  
6             System.out.println("value of a" +a);  
7             a++;  
8             System.out.println("  
9             ");  
10        }  
11    }  
12 }
```

Do-while statement:



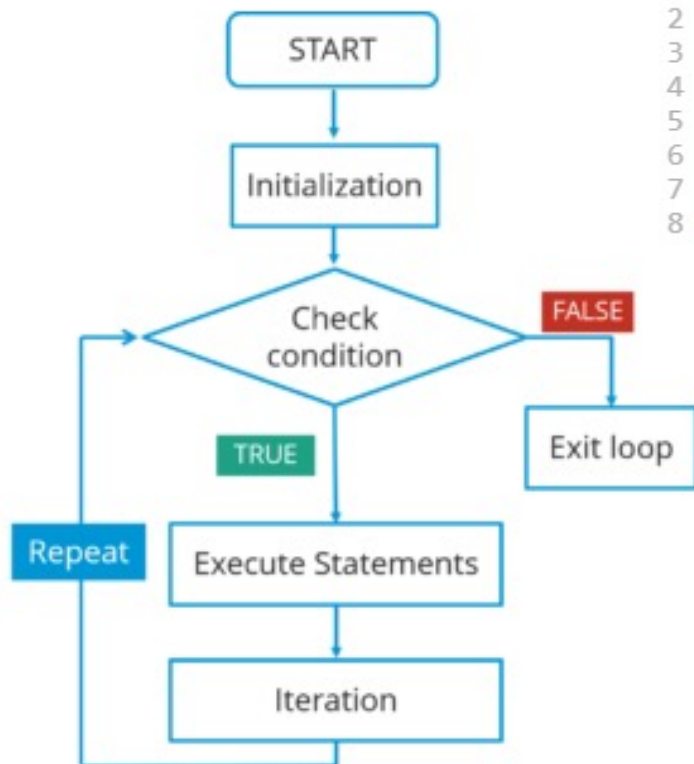
```
1 public class DoWhileExample {  
2     public static void main(string args[]){  
3         int count=1;  
4         do {                               // do statement  
5             System.out.println("count is:"+count);  
6             count++;  
7         }  
8         while (count<10)                   // while condition  
9     }  
10 }
```

For statement



```
1 public class ForExample {  
2     public static void main(String args[]) {  
3         for(int i=0; i<=10; i++) // for condition  
4         {  
5             System.out.println(i);  
6         }  
7     }  
8 }
```

For statement



```
1 public class ForExample {  
2     public static void main(String args[]) {  
3         for(int i=0; i<=10; i++) // for condition  
4         {  
5             System.out.println(i);  
6         }  
7     }  
8 }
```

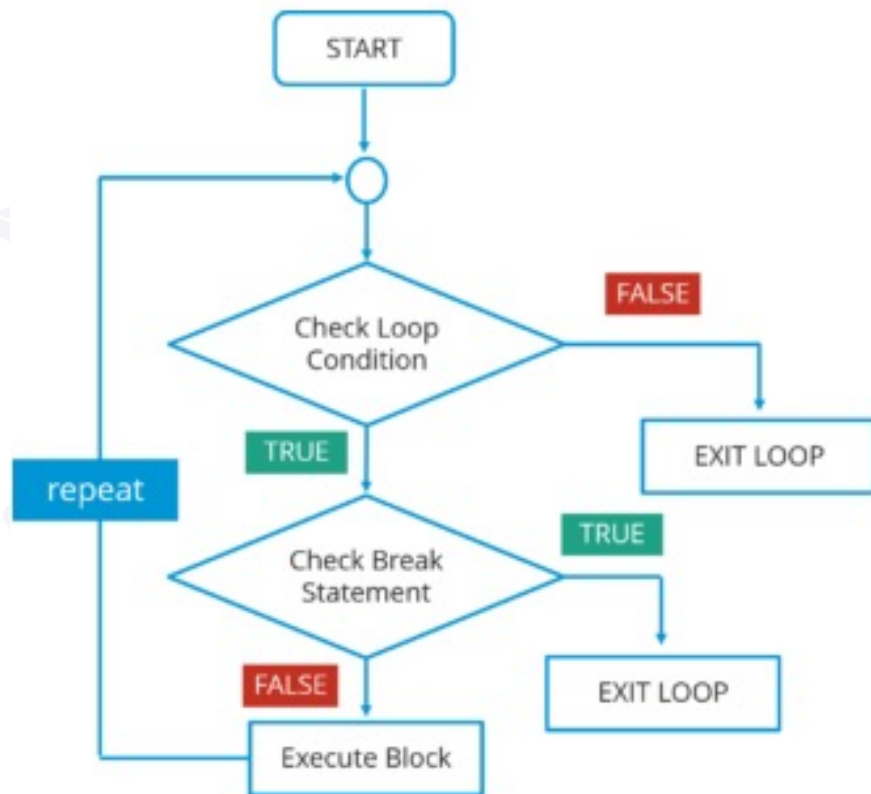
Foreach

- это разновидность цикла for
- используется для перебора элементов массива или коллекции

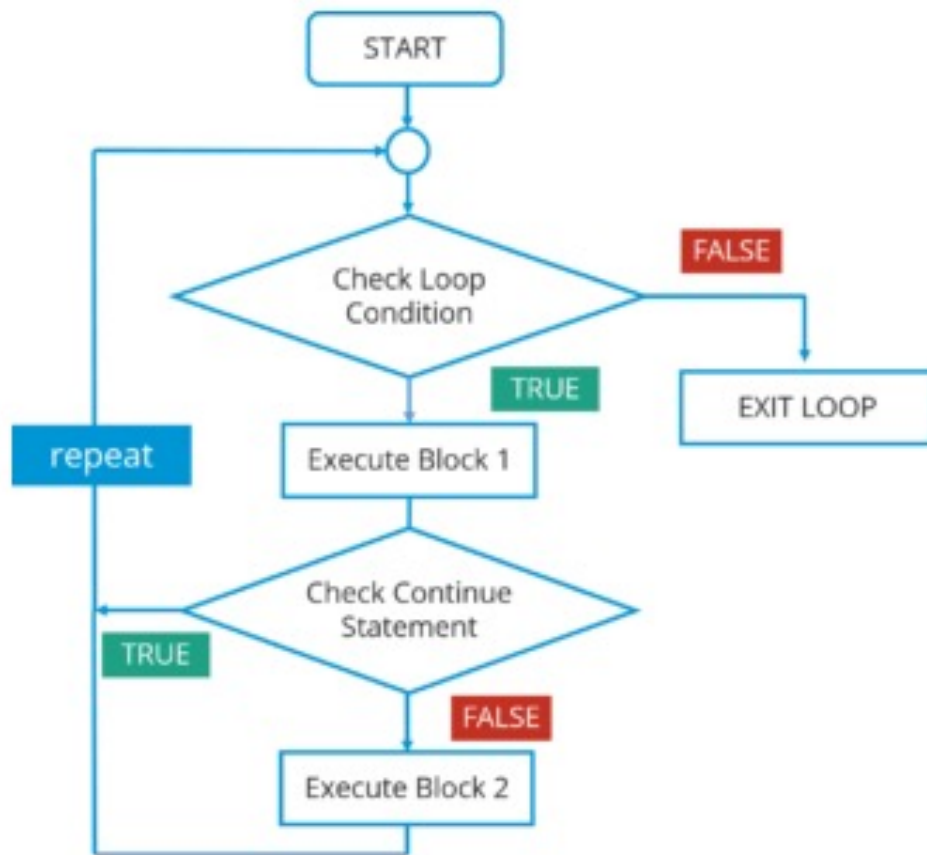
```
1 class Test {  
2  
3     public static void main(String[] args) {  
4         int[] array = {51,136, 387};  
5  
6         for (int i = 0; i < array.length; i++)  
7             System.out.println(array[i]);  
8     }  
9 }  
10 }
```

```
1 class Test {  
2  
3     public static void main(String[] args) {  
4         int[] array = {51,136,387};  
5  
6         for (int i:array) {  
7             System.out.println(i);  
8         }  
9     }  
10 }
```

Break statement



Continue statement





Массивы

Массив — это структура данных, в которой хранятся элементы одного типа. Его можно представить, как набор пронумерованных ячеек, в каждую из которых можно поместить какие-то данные (один элемент данных в одну ячейку). Доступ к конкретной ячейке осуществляется через её номер. Номер элемента в массиве также называют **индексом**.

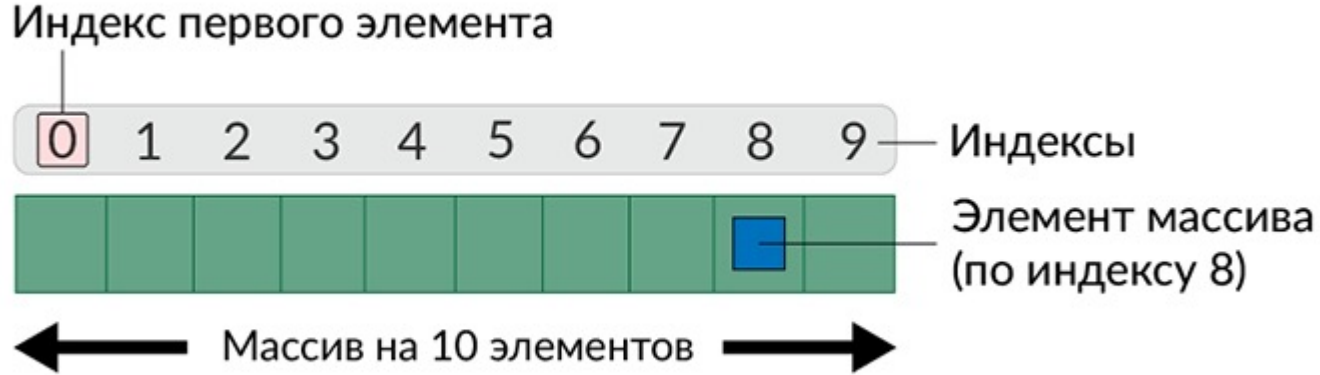
Массивы целых чисел




Объявление массивов

№	Объявление массива	Примеры	Комментарий
1.	<code>dataType[] arrayName;</code>	<code>int[] myArray</code>	Желательно объявлять массив именно таким способом, это Java-стиль
2.	<code>dataType arrayName[];</code>	<code>int myArray[]</code>	Унаследованный от C/C++ способ объявления массивов, который работает и в Java

После создания массива с помощью `new`, в его ячейках записаны значения по умолчанию. Для численных типов (как в нашем примере) это будет 0, для `boolean` — `false`, для ссылочных типов — `null`.



Как мы уже говорили выше, длина массива — это количество элементов, под которое рассчитан массив. **Длину массива нельзя изменить после его создания.** В Java элементы массива нумеруются с нуля. То есть, если у нас есть массив на 10 элементов, то первый элемент массива будет иметь индекс 0, а последний — 9.



1. Массив - это переменная, в которую можно положить не одно, а сразу несколько значений.

2. Все элементы массива имеют одинаковый тип.

3. При создании пустого массива элементам присваиваются значения в зависимости от типа данных массива:

для int - 0

для float, double - 0.0

для String - значение null

для char - \0

для boolean - значение false

4. Размер массива нельзя изменить после его создания.

Многомерные массивы

Массив, элементами которого являются другие массивы, то есть массив массивов, называется двумерным. Не во всех языках многомерные массивы имеют такую структуру, но в Java дела обстоят именно так.


```
Data_type[dimension1][dimension2][dimensionN] array_name =  
new data_type[size1][size2]....[sizeN];
```

Где **Data_type** — это тип элементов в массиве. Может быть примитивным или ссылочным (классом).

Количество пар скобок с **dimension** внутри — размерность массива (в нашем случае — N).

array_name — название массива

size1...sizeN — количество элементов в каждом из измерений массива.



Двумерный массив в Java — это массив массивов, то есть в каждой его ячейке находится ссылка на некий массив. Но гораздо проще его представить в виде таблицы, у которой задано количество строк (первое измерение) и количество столбцов (второе измерение). Двумерный массив, у которого все строки имеют равное количество элементов, называется **прямоугольным**.

Объявление, создание и инициализация двумерных массивов

Процедура объявления и создания двумерного массива практически такая же, как и в случае одномерного:

```
1 int[][] twoDimArray = new int[3][4];
```

```
1 int [][] twoDimArray = {{5,7,3,17}, {7,0,1,12}, {8,1,2,3}};
```

ДВУМЕРНЫЙ МАССИВ

Индексы строк ---- ↓

	0	1	2	3	←---- Индексы столбцов
0	5	7	3	17	
1	7	0	1	12	twoDimArray[1][2] //1
2	8	1	2	3	twoDimArray[2][3] //3

Всего в массиве **3 строки** и **4 столбца**, то есть $3 \times 4 = 12$ элементов


```
1 int [][] twoDimArray = {{5,7,3,17}, {7,0,1,12}, {8,1,2,3}}; //объявили массив и заполнили его элементами
2 for (int i = 0; i < 3; i++) { //идём по строкам
3     for (int j = 0; j < 4; j++) { //идём по столбцам
4         System.out.print(" " + twoDimArray[i][j] + " "); //вывод элемента
5     }
6     System.out.println(); //перенос строки ради визуального сохранения табличной формы
7 }
```

```
1 int[][] myArray = {{18,28,18},{28,45,90},{45,3,14}};
2 System.out.println(Arrays.deepToString(myArray));
```