



Faculty of Software Engineering and Computer Systems

Programming

Lecture #1

Syntax constructions. Methods. Arrays.

Instructor of faculty
Pismak Alexey Evgenievich
Kronverksky Pr. 49, 1331 room
pismak@itmo.ru

Saint-Petersburg

Syntax constructions

Conditional expressions

```
if ( condition ) expr
```

```
if ( condition ) expr else expr_2
```

```
if ( condition ) expr else if ( condition ) expr_x ...
```

```
1.  final int LIMIT_TEMPERATURE = 25;  
2.  
3.  int t = 21;  
4.  boolean isSwitchedOff = false;  
5.  
6.  if (t > LIMIT_TEMPERATURE) {  
7.      ...  
8.  }
```

Dangerous!

boolean conditional = a * b > c;

if (conditional) { ... }

if (conditional == true) { ... }

if (conditional != false) { ... }

if (String.valueOf(conditional).equals("true")) { ... }

if (conditional == true && conditional != false) { ... }



Ternary operator

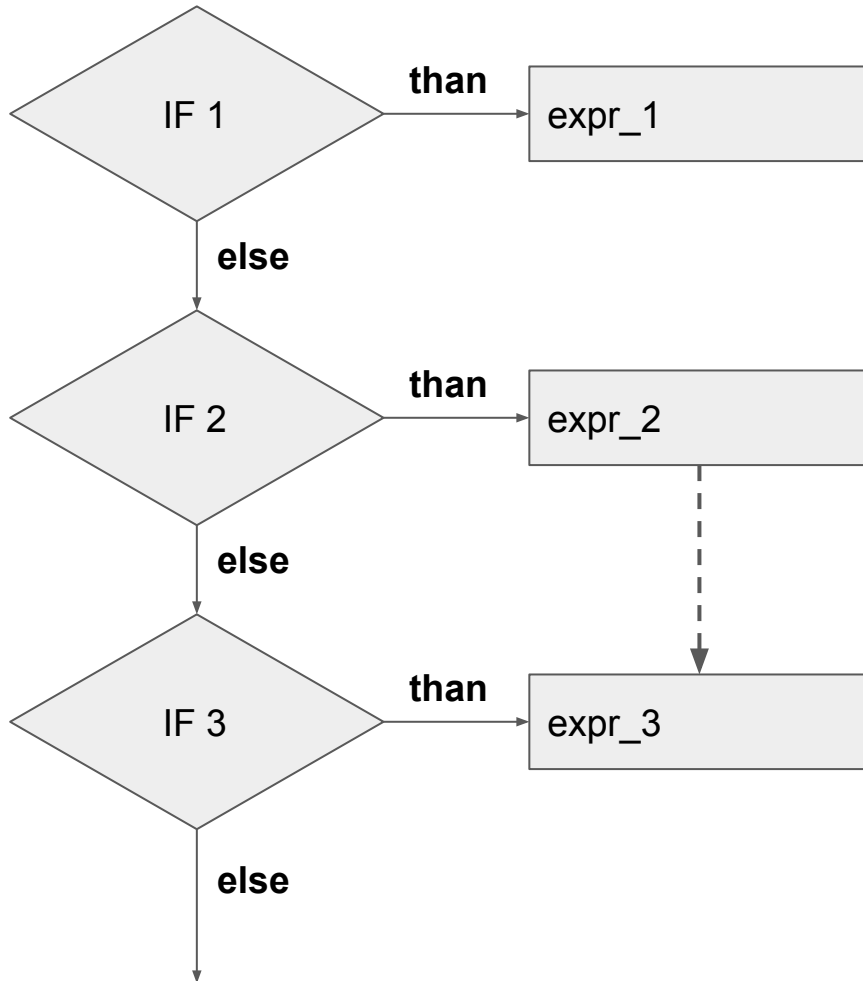
```
condition ? expression : expression;
```

```
// expression must return value
```

```
int delta = x > 0 ? x : Math.abs(x);
```

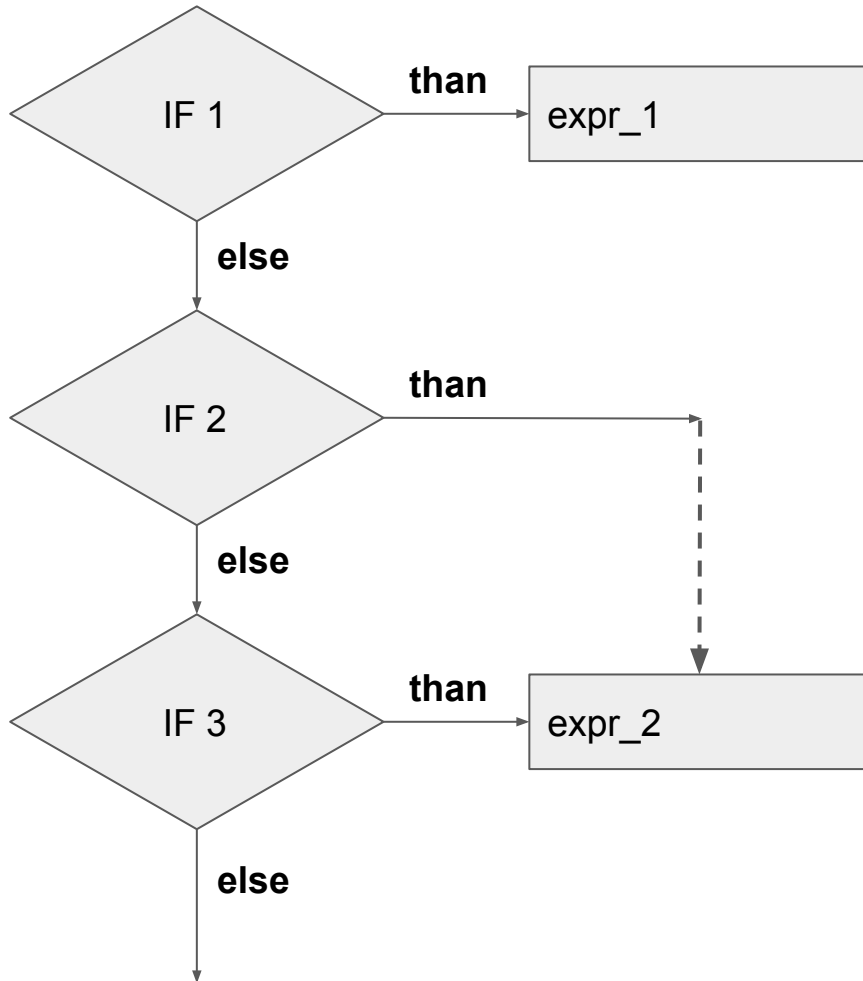
```
1. int delta;  
2. if(x > 0) {  
3.     delta = x;  
4. } else {  
5.     delta = Math.abs(x);  
6. }
```

Multivariate branching (classic)



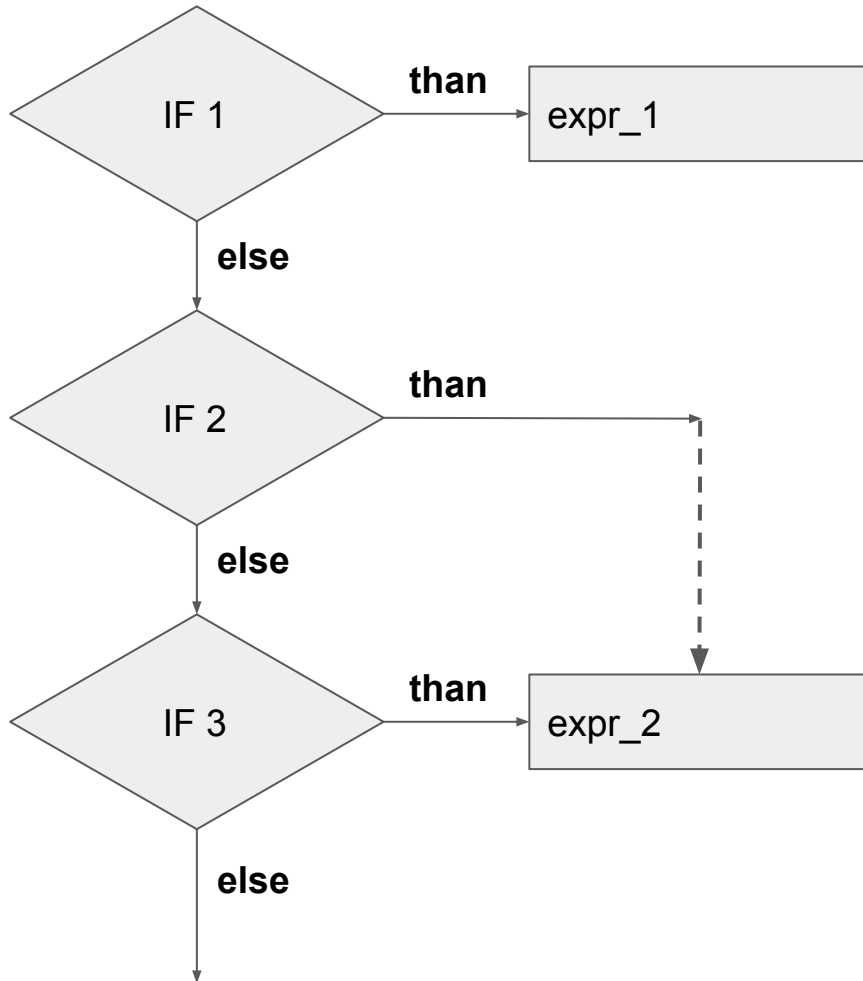
```
switch ( x ) {  
  
  case 1 : expr_1;  
    break;  
  
  case 2 :  
  case 3 : expr_2;  
  
  default : expr_n;  
  
}
```

Multivariate branching (upgrade#1)



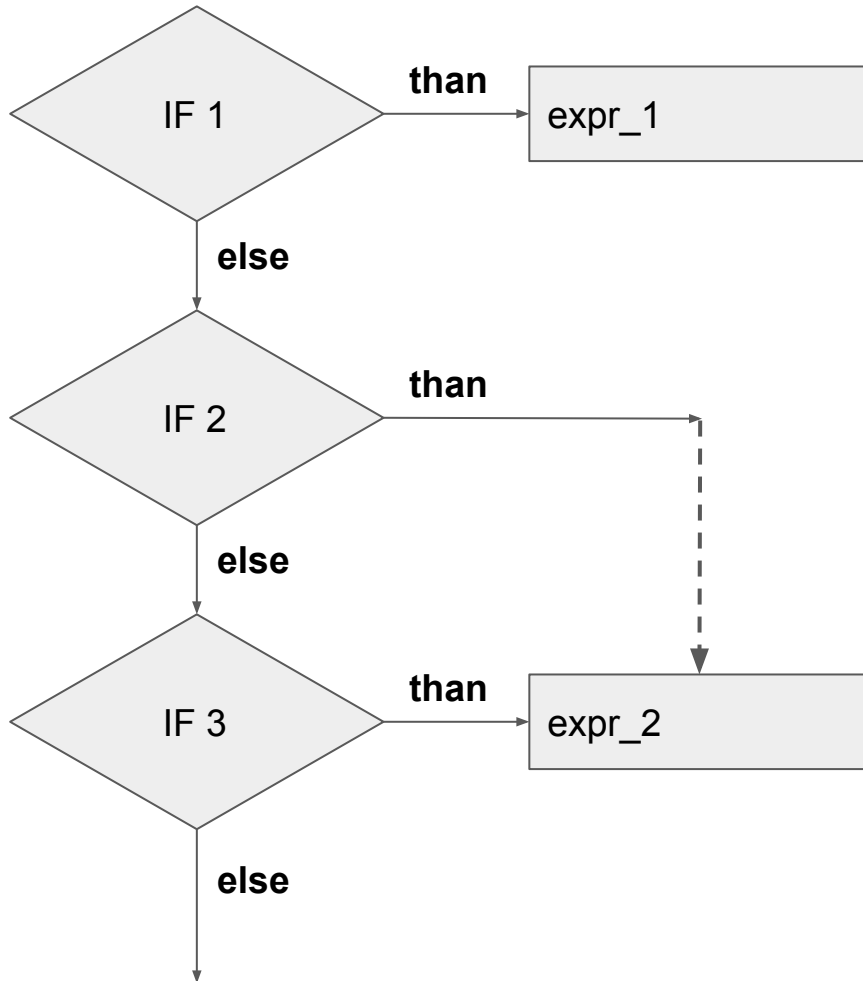
```
switch ( x ) {  
  
  case 1 : expr_1;  
    break;  
  
  case 2,3 : expr_2;  
  
  default : expr_n;  
  
}
```

Multivariate branching (upgrade#2)



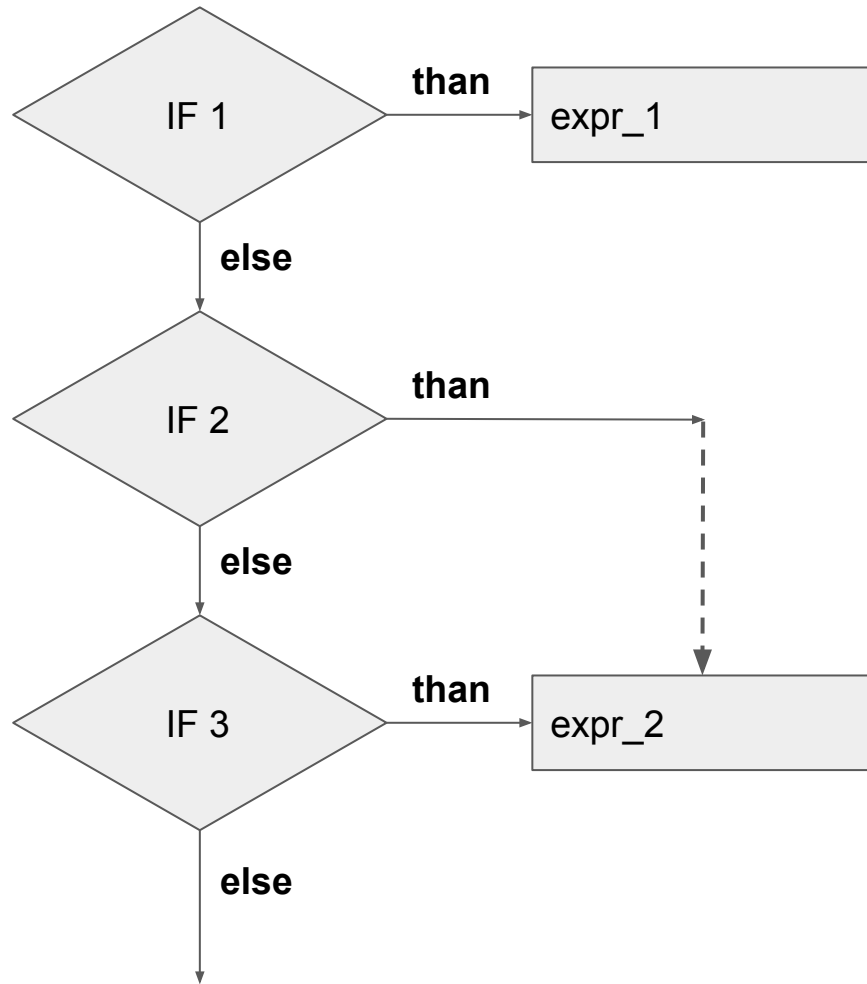
```
switch ( x ) {  
  case 1 -> expr_1;  
  case 2,3 -> expr_2;  
  default -> expr_n;  
}
```


Multivariate branching (upgrade#3)



```
var y = switch ( x ) {  
  case 1 -> 10;  
  case 2,3 -> 20;  
  default -> 100;  
};
```

Multivariate branching (upgrade#4)



```
var y = switch ( x ) {  
  
  case 1 -> 1;  
  
  case 2,3 -> {  
    // calc result  
    yield result;  
  };  
  
  default -> 10;  
};
```

Indefinite loops

```
while ( condition ) expression;
```

```
while ( true ) {  
    // do something  
}
```

```
while ( x > 0 ) {    // this may never executed  
    // code here  
}
```

Indefinite loops

```
do expression while ( condition );
```

```
do {  
    // do something  
} while ( true );
```

```
do {    // executed at least once  
    // code here  
} while ( x > 0 );
```

Definite loop `for`

```
for ( init_block; condition; calc_block ) expression;
```

```
/*  
 * Print numerals  
 */  
for (int i = 0; i < 10; ++i) {  
  
    System.out.println( i );  
  
}
```

```
// square table  
for (int i = 0, j = 0; i < 10 && j < 10; ++i, ++j) {  
  
    System.out.println("%d * %d = %d", i, j, i * j );  
  
}
```

Iterable loop `for`

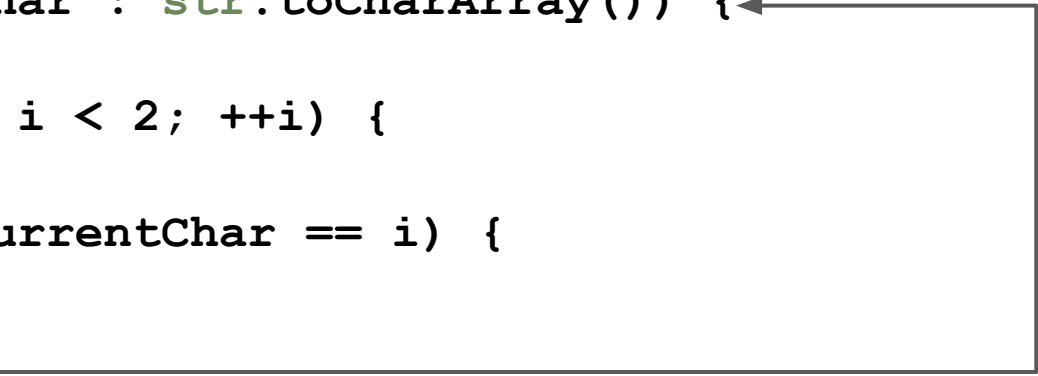
```
for ( def_variable : set ) expression;
```

```
int[] array = // initialization array ;
```

```
for (int element : array) {  
    if (element != 0) {  
        System.out.print(element) ;  
    }  
}
```

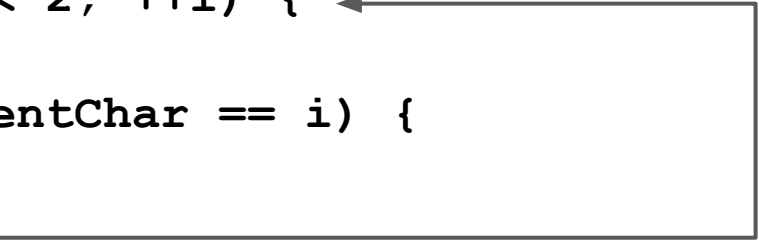
Interrupt execution

```
1. String str = "some string";  
2.  
3. for (char currentChar : str.toCharArray()) {  
4.  
5.     for(int i = 0; i < 2; ++i) {  
6.  
7.         if ((int)currentChar == i) {  
8.  
9.             break;  
10.  
11.         }  
12.     }  
13. }
```



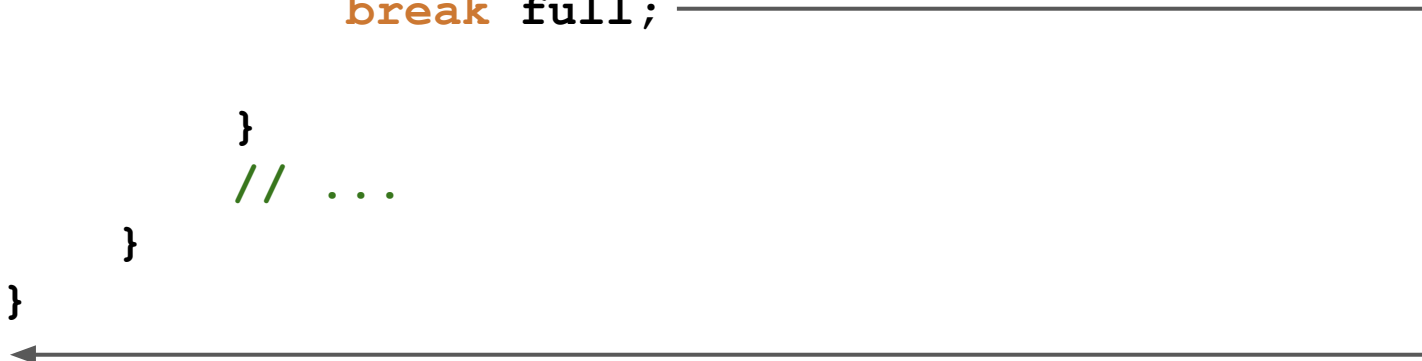
Interrupt execution

```
1. String str = "some string";
2.
3. for (char currentChar : str.toCharArray()) {
4.
5.     for(int i = 0; i < 2; ++i) { ←
6.
7.         if ((int)currentChar == i) {
8.
9.             continue;
10.
11.         }
12.         // ...
13.     }
14. }
```



Interrupt all loops

```
1. String str = "some string";
2.
3. full:
4. for (char c : str.toCharArray()) {
5.
6.     for(int i = 0; i < 2; ++i) {
7.
8.         if ((int)c == i) {
9.
10.            break full;
11.
12.        }
13.        // ...
14.    }
15. }
```



The diagram illustrates the execution of the provided Java code. A horizontal line extends from the `break full;` statement on line 10 to the right edge of the code block. From there, a vertical line descends to the bottom edge, and a horizontal line continues to the left, ending in an arrowhead pointing towards the `full:` label on line 3. This visualizes the `break` statement's effect of exiting the entire loop structure, not just the inner loop.

Blocks and scope

```
{  
    // expressions, operators etc.  
}
```

```
if ( condition ) expression  —————→  if ( condition ) {  
                                          // code  
                                          }
```

```
public static void main (String[] args) {  
    // code  
    {  
        // code  
    }  
    // code  
}
```

“Subprograms” (methods)

```
public static void printMessage (String msg) {  
  
    System.out.println (msg);  
  
}  
  
public static void main(String[] args) {  
  
    printMessage ("I am liquid");  
  
}
```

Methods

```
public static int cube (int arg) {  
  
    return arg * arg * arg;  
  
}
```

```
public static void main(String[] args) {  
  
    printMessage ("5^3 = " + cube(5));  
  
}
```

```
public static void printMessage (String msg) {  
  
    System.out.println (msg);  
  
}
```

How to write a method with arguments like this `System.out.printf` ?

```
public static void main(String[] args) {
```

```
    System.out.printf("5^3 = %d", cube(5));
```



String + one argument

```
    System.out.printf("%d = %d", cube(5), 5*5*5);
```

```
}
```

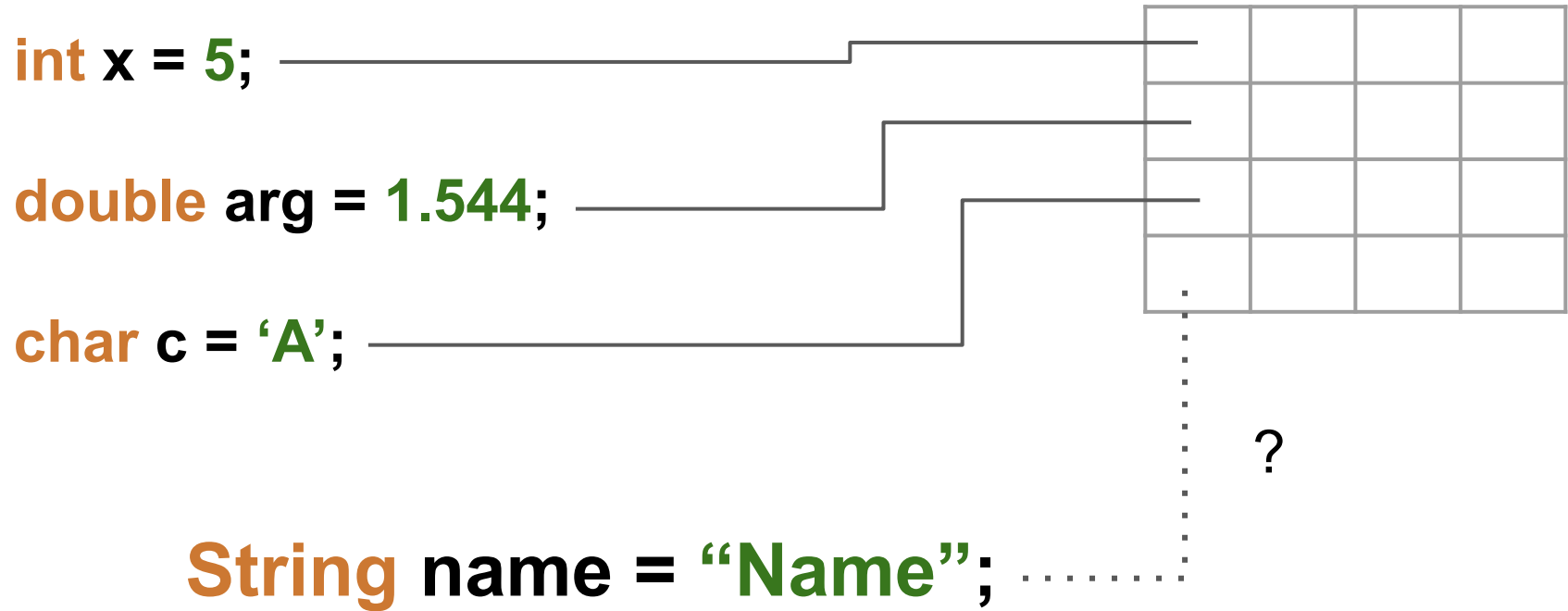


String + two arguments

Variable arguments (VARARGS)


```
public static void main(String[] args) {  
  
    printMessage ("This ", "Bob");  
    printMessage ("Я", "угадаю", "как", "тебя", "зовут");  
}  
  
public static void printMessage (String ... msgs) {  
    for(String s : msgs) {  
        System.out.println (s);  
    }  
}
```

Reference and value data types



Ref- and valuable data types

String name “Name”



The diagram illustrates the memory layout of a String. A variable named 'name' of type 'String' is shown on the left. An arrow points from the variable to a 4x4 grid of cells. The bottom row of the grid contains the characters 'N', 'a', 'm', and 'e', representing the string 'Name'. The other three rows of the grid are empty.

N	a	m	e

```
String name1 = new String(“Name”);
```

```
String name2 = new String(“Name”);
```

```
System.out.println ( name1 == name2 );
```


Ref- and valuable data types

String name **“Name”**

N	a	m	e

```
String name1 = new String(“Name”);  
String name2 = new String(“Name”);
```

```
String name3 = “Name”;  
String name4 = “Name”;
```

Operator `new`

```
int[] y = new int[2];
```

```
String str = new String("I am liquid");
```

How reset reference variable to uninitialized value?

Value 'null'

```
int[] y = null;
```

```
String str = null;
```

```
y.length    // after this operations
```

```
str.trim()  // will errors
```

Arrays



```
int[] a;
```

```
int b[];
```

```
int[] x = {5, 2};
```

```
int[] y = new int[2];
```

Arrays

```
int[] x = {5, 2};
```

```
int count = x.length; // property
```

```
java.util.Arrays // work with array
```

```
    sort
```

```
    search
```

```
    copy
```

Arrays

Sort:

Arrays.sort (...)

Arrays.parallelSort (...)

Search:

Arrays.binarySearch (...)

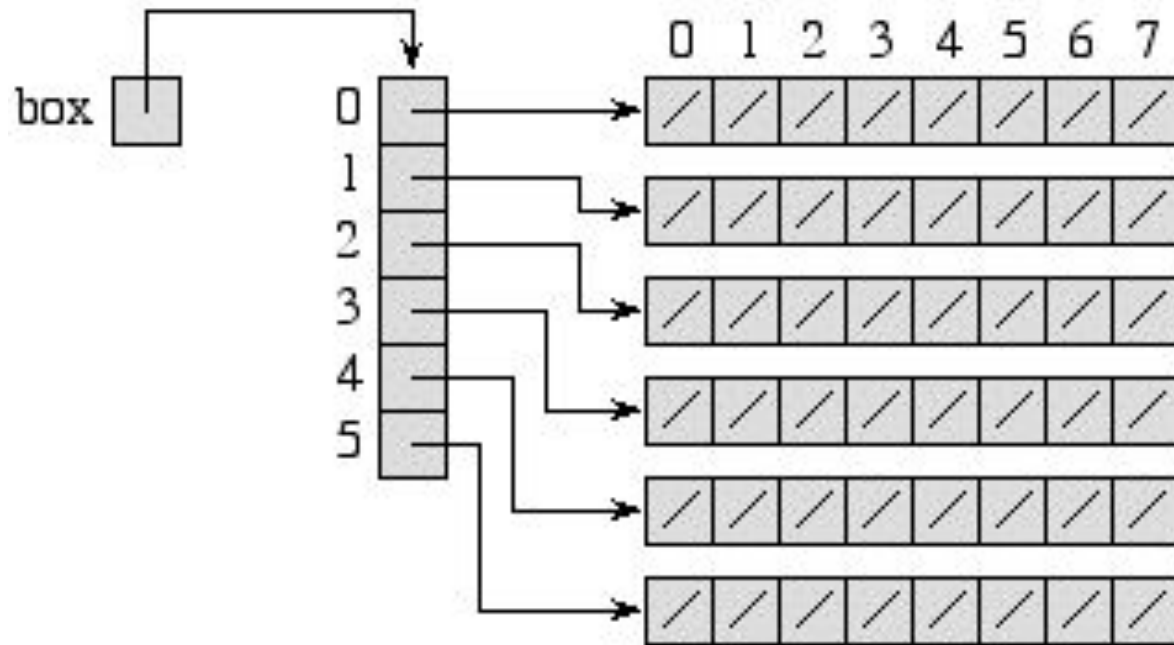
Copy:

System.arraycopy (...)

Arrays.copyOfRange (...)

filling, applying specific math expression, set default values etc.

Arrays

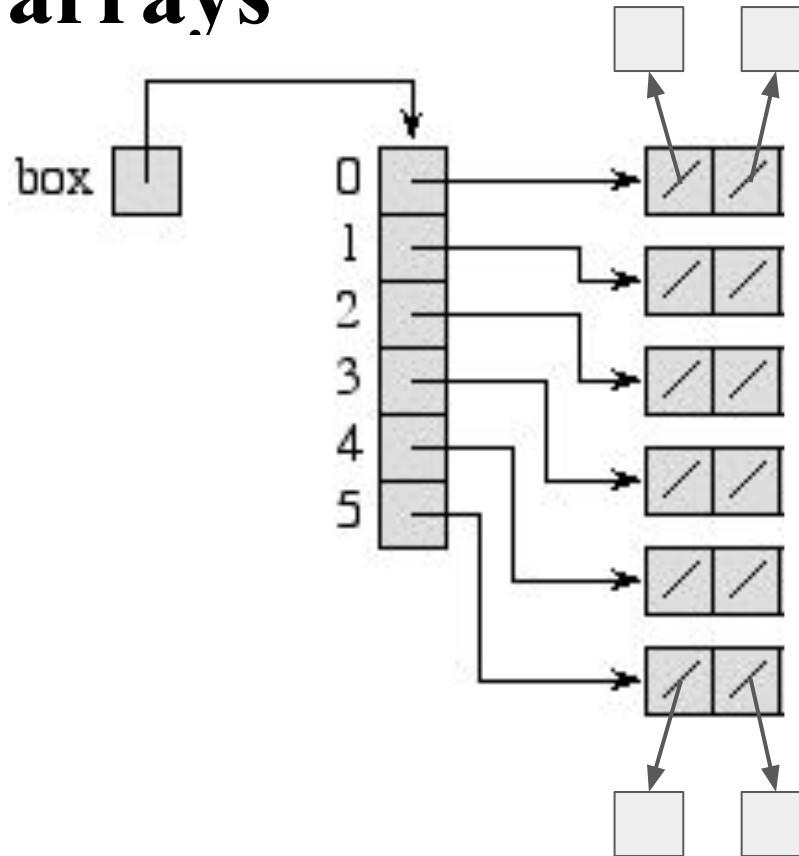


```
int[][] matrix;
```

```
int[][] box = new int[6][8];
```

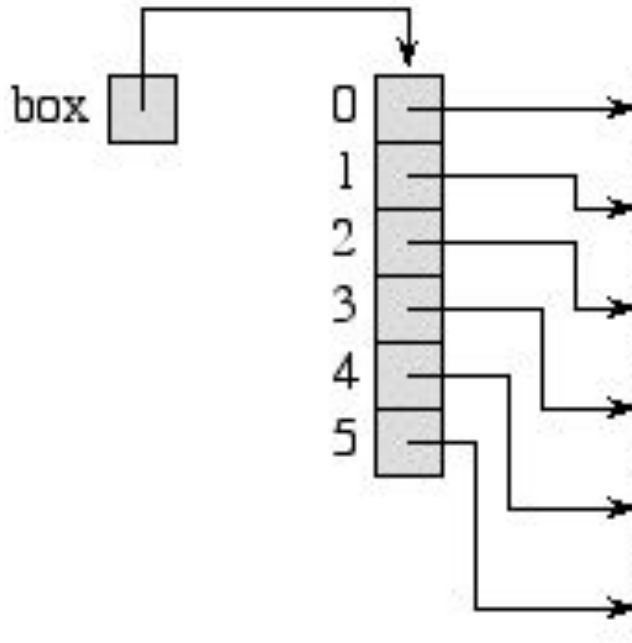
```
int[][] box = { {1, 2, 3}, {4, 5, 6} };
```

Crazy arrays



```
int[][][] box = new int[6][2][1];
```


Arrays



```
int[][] box;
```

```
int[] box[] = new int[6][8];
```

```
int[][] box = new int[6][];
```

```
int box[][] = { {1, 2, 3}, {4, 5, 6} };
```

Utils. Classpath & Imports

```
1. import static java.lang.Math.*;
2.
3. /**
4.  * Безысходники (game of words: sources + hopelessness)
5.  */
6. public class PracticMath {
7.
8.     public static void main(String[] args) {
9.
10.         double x = 5.1, y = 3.57;
11.
12.         double res = sin(( x + 1) / 3*PI) * 8*cos(y) ;
13.
14.     }
15.
16. }
```

Utils. Classpath & Imports

```
1. import java.util.Arrays;
2.
3. /**
4.  * Безысходники (game of words: sources + hopelessness)
5.  */
6. public class PracticArrays {
7.
8.     public static void main(String[] args) {
9.
10.         int[] x = new int[2];
11.
12.         Arrays.fill(x, 10);
13.
14.     }
15.
16. }
```