Faculty of Software Engineering and Computer Systems

Programming

Lecture #1
Syntax constructions. Methods. Arrays.

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

Saint-Petersburg

Syntax constructions

Statements vs Expressions

```
1.  System.out.println("!") ;     // method invocation statement
2.  if ( a > 0 ) z = y + a ;      // if statement
3.  while (true) counter++ ;      // while statement
4.  f = a*x*x + b*x + c ;         // assignment statement
```

```
1.  (float) Math.cos(x)           // cast expression
2.  a * ( x + b) > 0              // relational expression
3.  isDone && ! hasErrors         // logical expression
4.  a * x * x + b * x + c         // arithmetic expression
```

Conditional statement

```
if ( condition ) statement


if ( condition ) statement else statement_2


if ( condition ) statement else if ( condition ) statement_x
...
```

```
final int TEMPERATURE_LIMIT = 25;
int currentTemperature = 21;
boolean isSwitchedOff = false;

if(currentTemperature > TEMPERATURE_LIMIT) {
    …
    isSwitchedOff = true;
}
```

Dangerous!

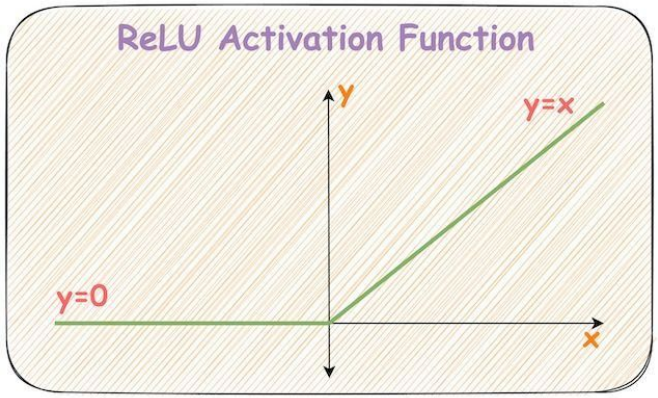
```
boolean conditional = a * b > c;

if ( conditional ) { … }
```

```
if ( conditional == true ) { … }
if ( conditional != false ) { … }
```

```
if ( String.valueOf(conditional).equals("true") )
if ( String.valueOf(conditional).length() == 4 )
if ( conditional == true && conditional != false )
```

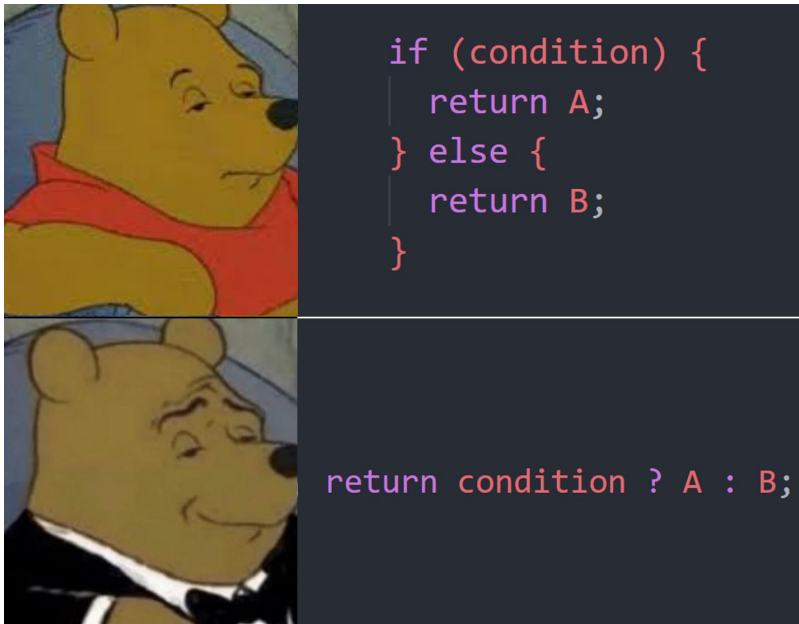
Ternary operator

```
condition ? expression : expression;

// expression must return value
```

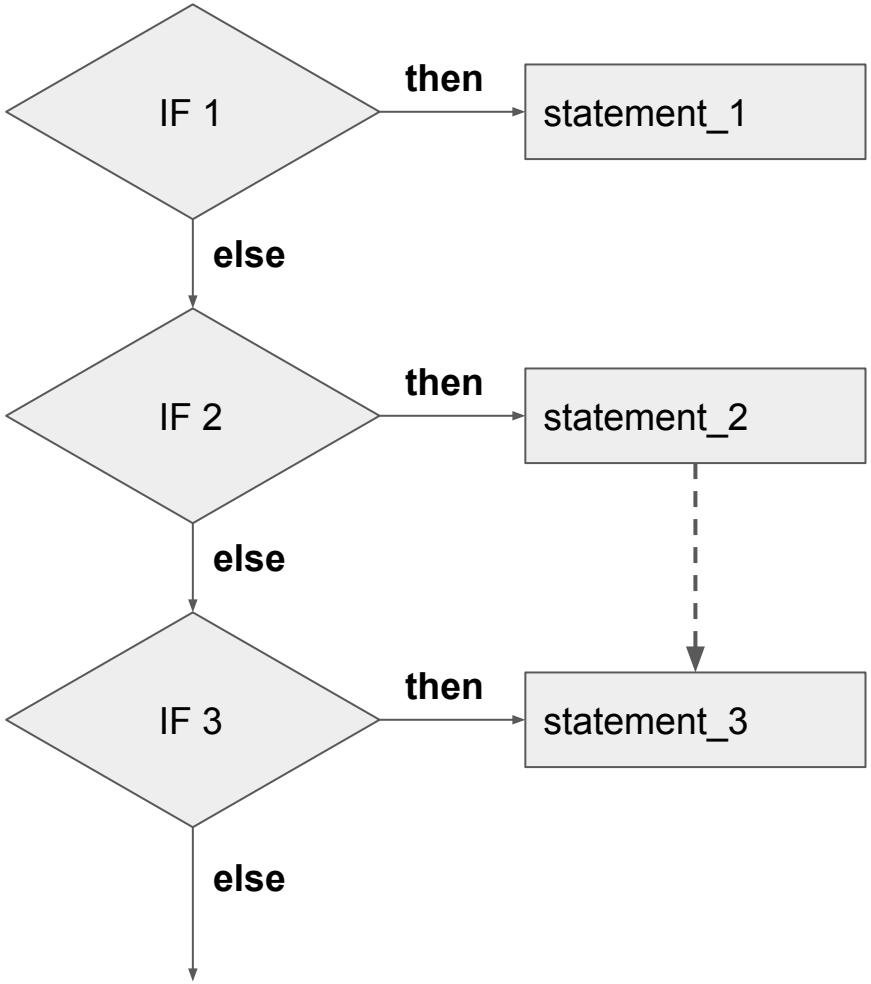



```
int relu = x > 0 ? x : 0;
```

```
int relu;
if(x > 0) {
  relu = x;
} else {
  relu = 0;
}
```

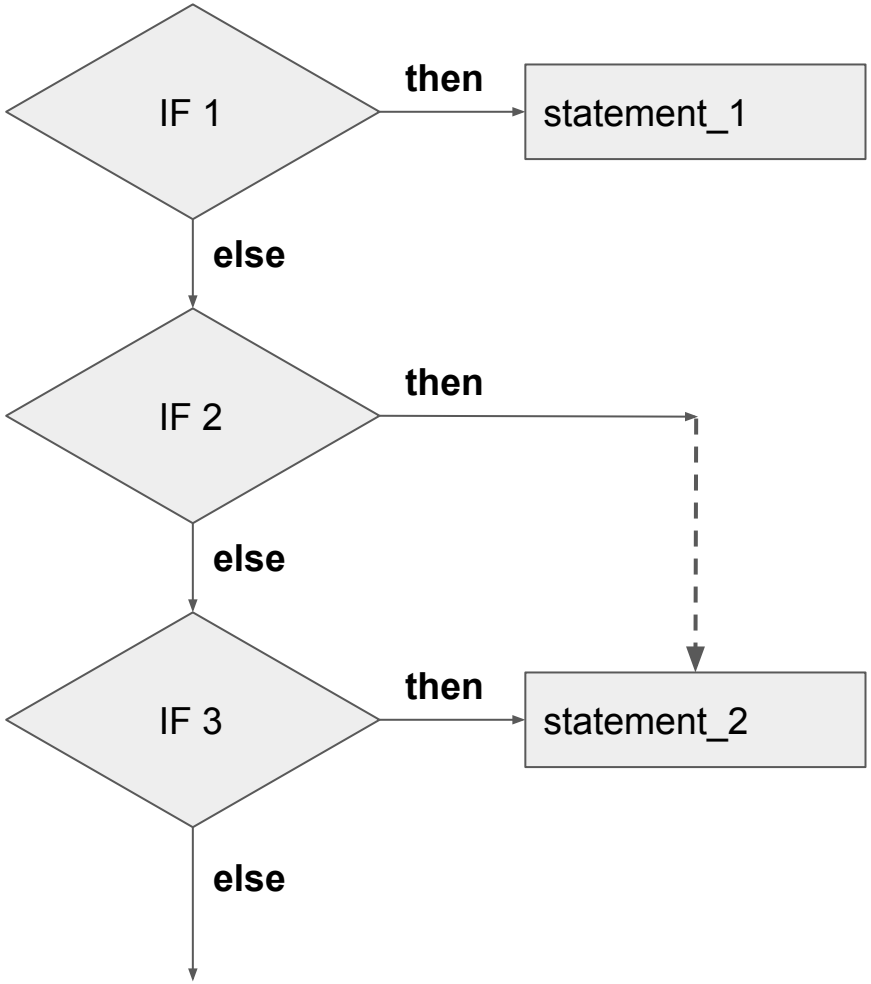


Multivariate branching (classic)



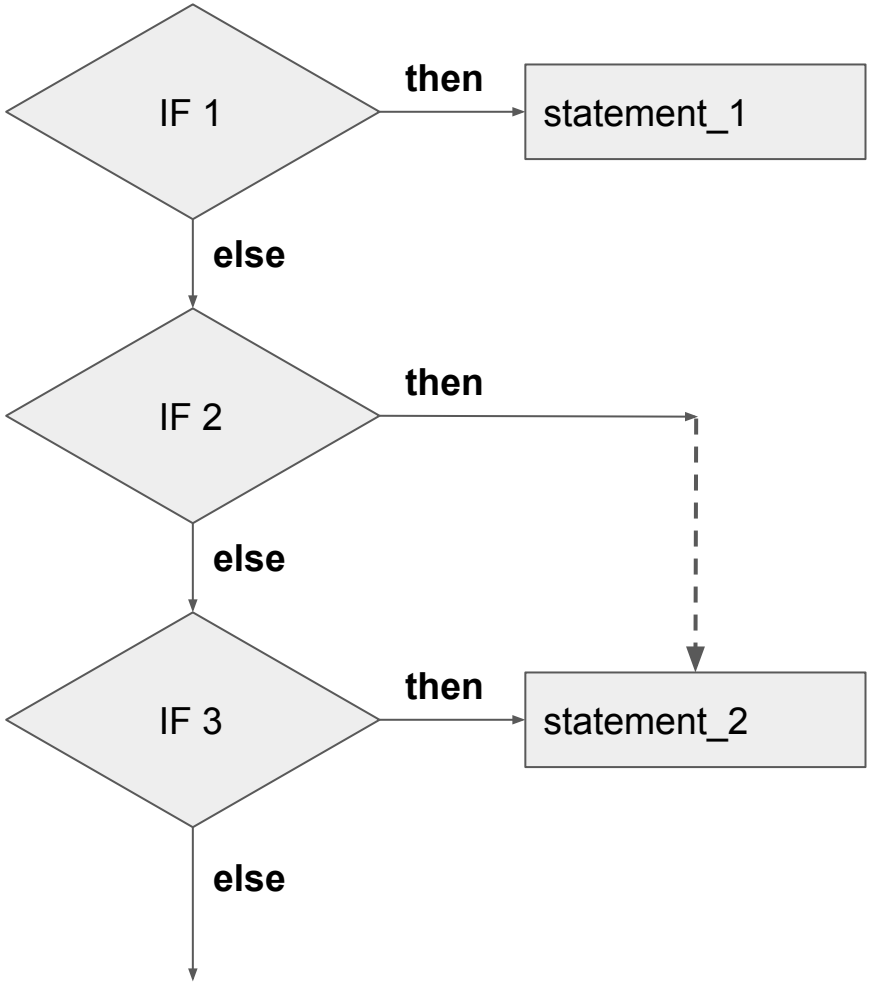

```
switch ( x ) {

case 1 : statement_1;
    break;

case 2 :
case 3 : statement_2;

default : statement_n;

}
```

Multivariate branching (upgrade#1)



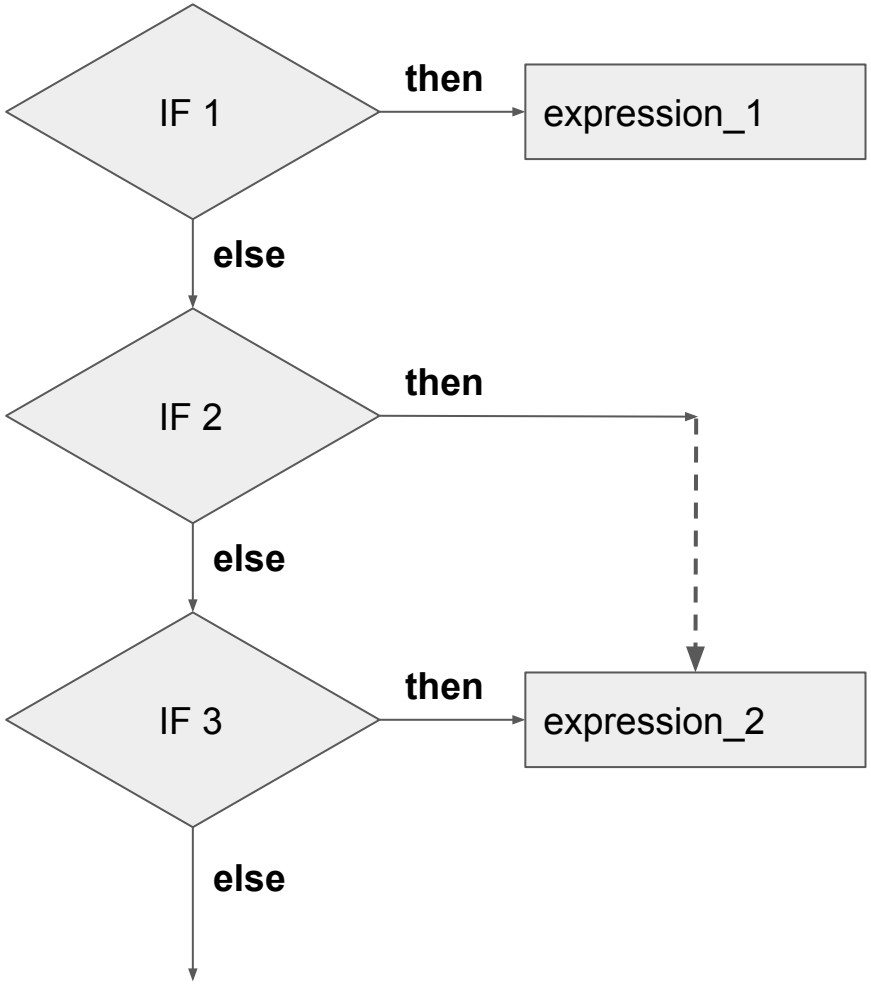

```
switch ( x ) {

case 1 : statement_1;
    break;

case 2,3 : statement_2;

default : statement_n;

}
```

Multivariate branching (upgrade#2)



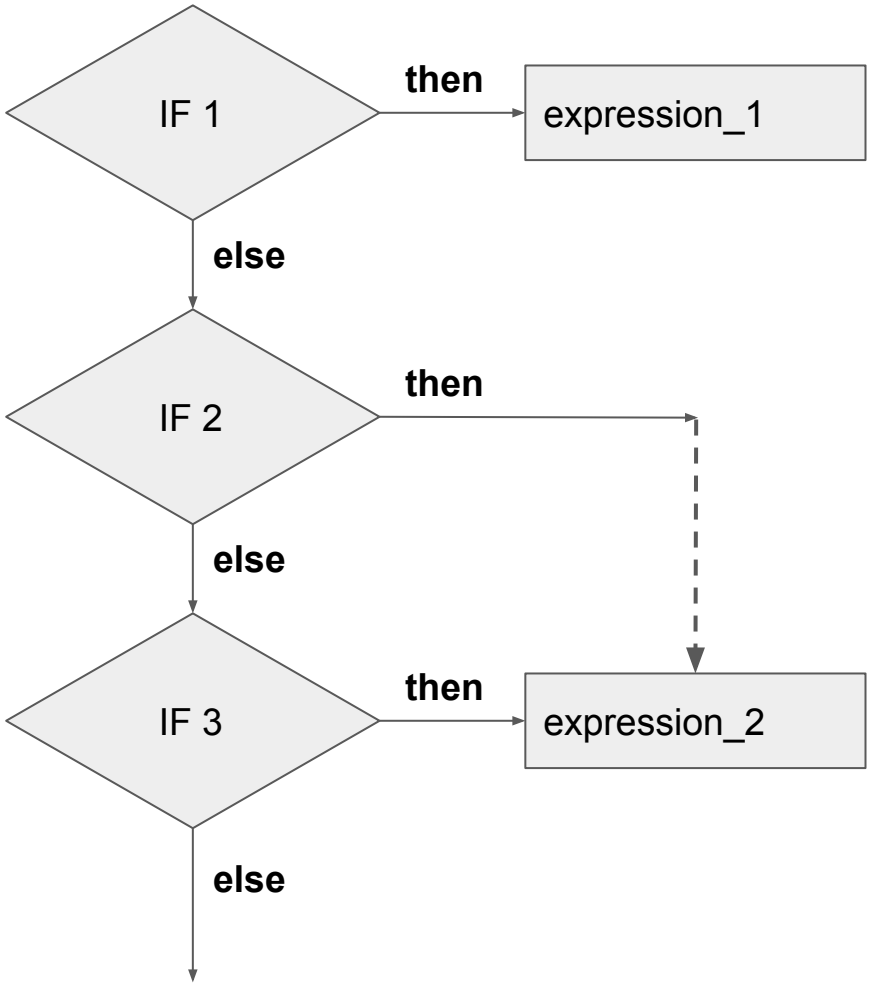

```
switch ( x ) {

case 1 -> statement_1;

case 2,3 -> statement_2;

default -> statement_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 ) statement;



while ( true ) {

  // do something

}



while ( x > 0 ) {      // this may never executed

    // code here

}
```

Indefinite loops

```
do statement 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 ) statement;


/*
 * 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.printf("%d * %d = %d", i, j, i * j );

}
```

Definite loop `for`

```
for (;;) {

    System.out.println( “я жидкий” );

}
```

1. won’t compile
2. won’t run
3. will run once
4. other

Iterable loop `for`

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

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

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

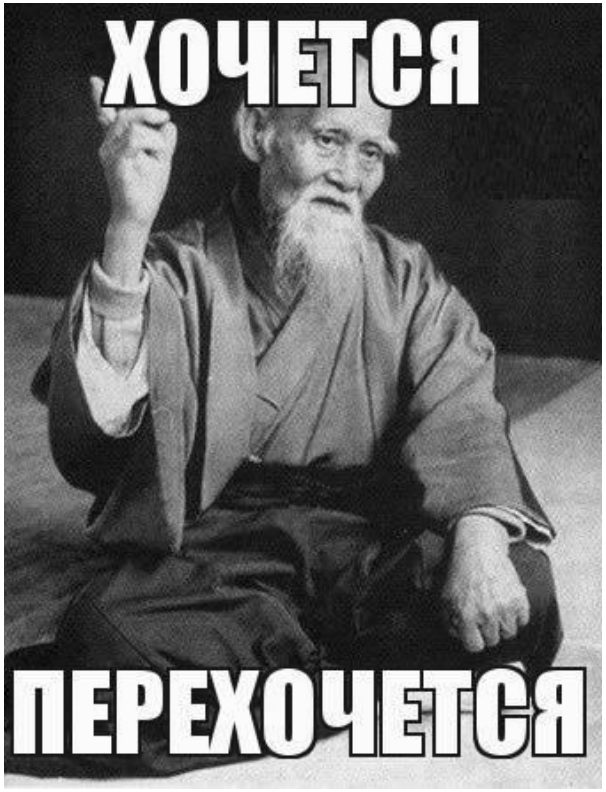
Interrupt execution

```
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

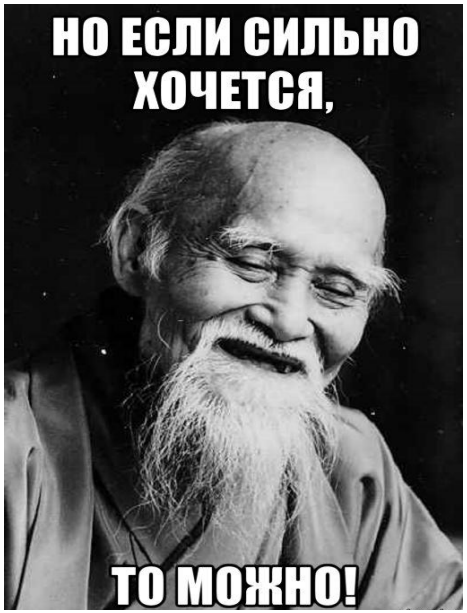
```
1.  String str = "some string";
2.
3.  for (char currentChar : str.toCharArray()) {
4.
5.      for(int i = 0; i < 2; ++i) {
6.
7.          if ((int)currentChar == i) {
8.
9.              continue;
10.
11.         }
12.         // ...
13.     }
14. }
```

but… if you want to exit from all loops??



* If you want, you won’t want

Interrupt all loops




```
String str = “some string”;

full:
for (char c : str.toCharArray()) {

    for(int i = 0; i < 2; ++i) {

        if ((int)c == i) {

            break full;

        }
        // ...
    }
}
```

Blocks and scope

```
{
    // statements, declarations
}
```

```
if ( condition ) expression
```

→

```
if ( condition ) {
    // code
}
```

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

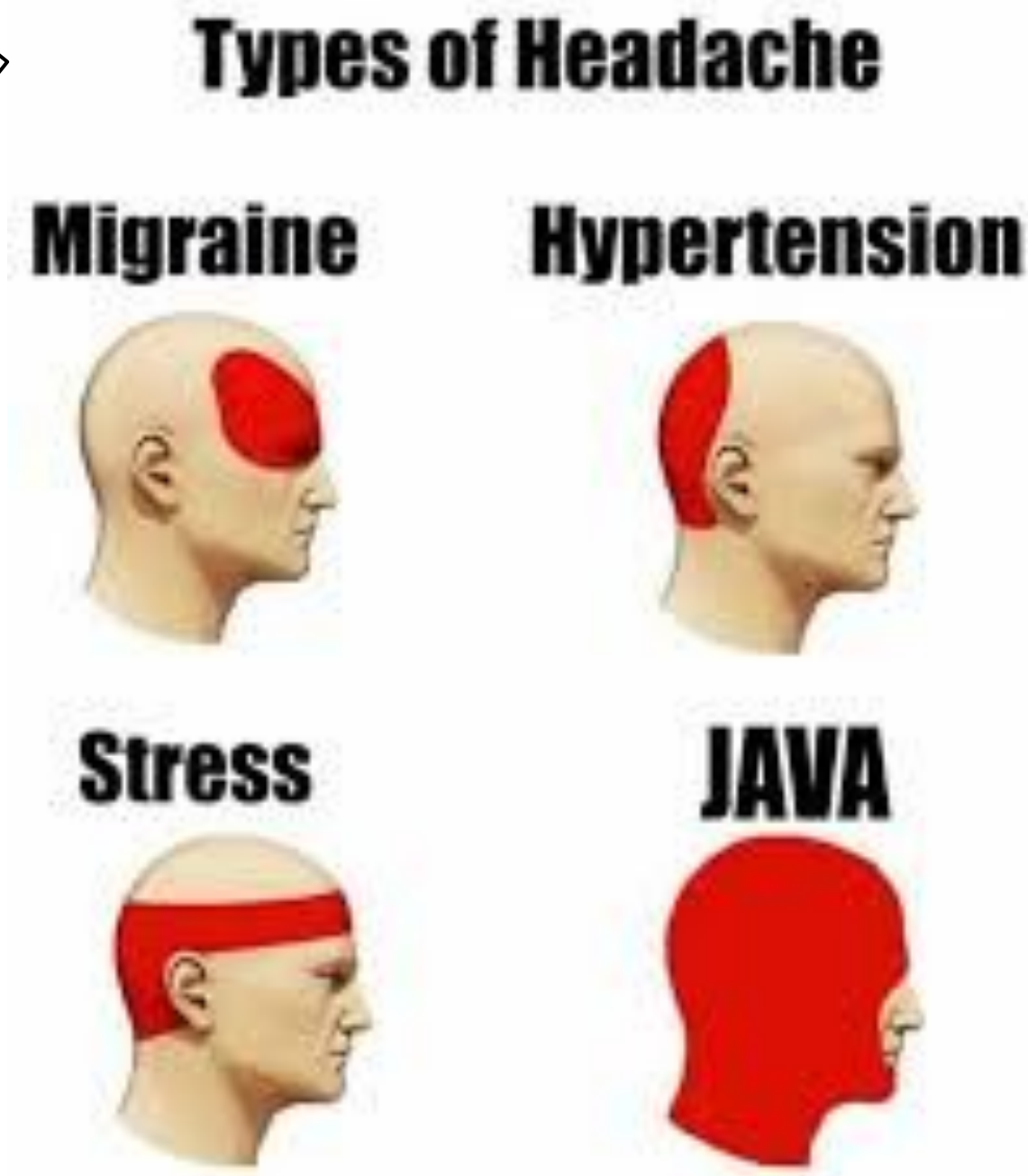
“Subprograms” (methods)

```
public static void printMessage (String msg) {

    System.out.println (msg);

}

public static void main(String[] args) {

    printMessage (“I am liquid”);

}
```

Methods

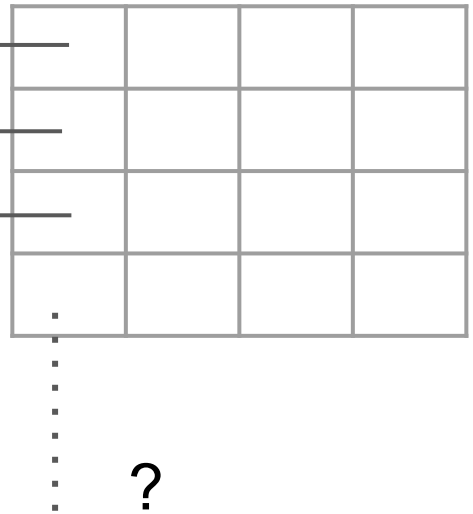
```
public static int cube (int arg) {
    return arg * arg * arg;
}

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

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

Questions?



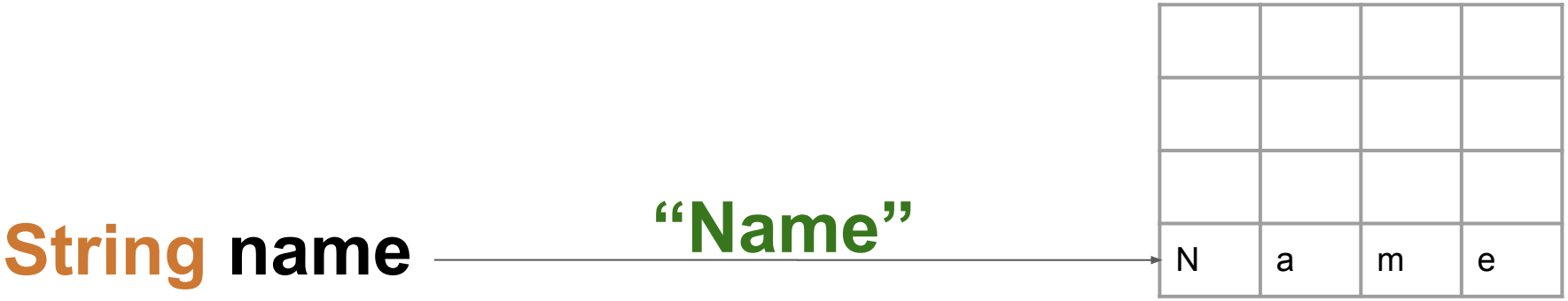
Reference and value data types

```
int x = 5;
double arg = 1.544;
char c = 'A';
```



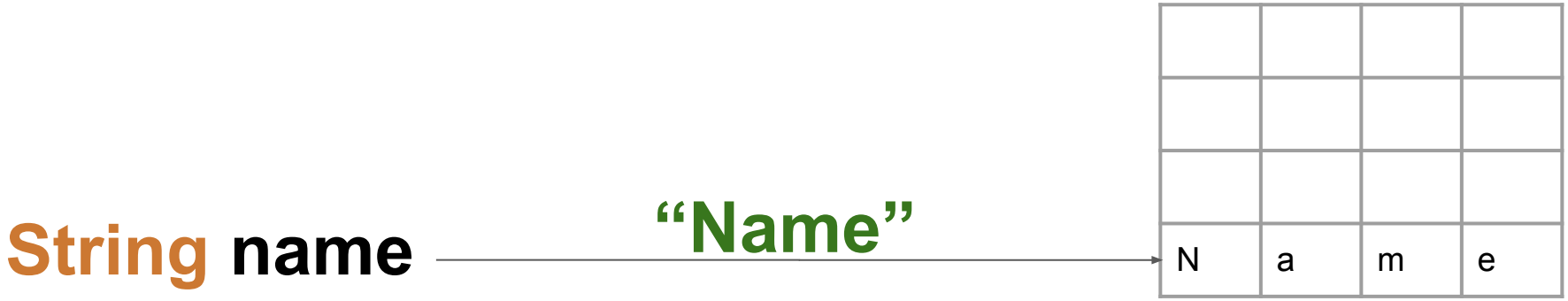

```
String name = "Name";
```

Ref- and valuable data types




```
String name1 = new String("Name");

String name2 = new String("Name");

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

Ref- and valuable data types




```
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 be 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 (…)

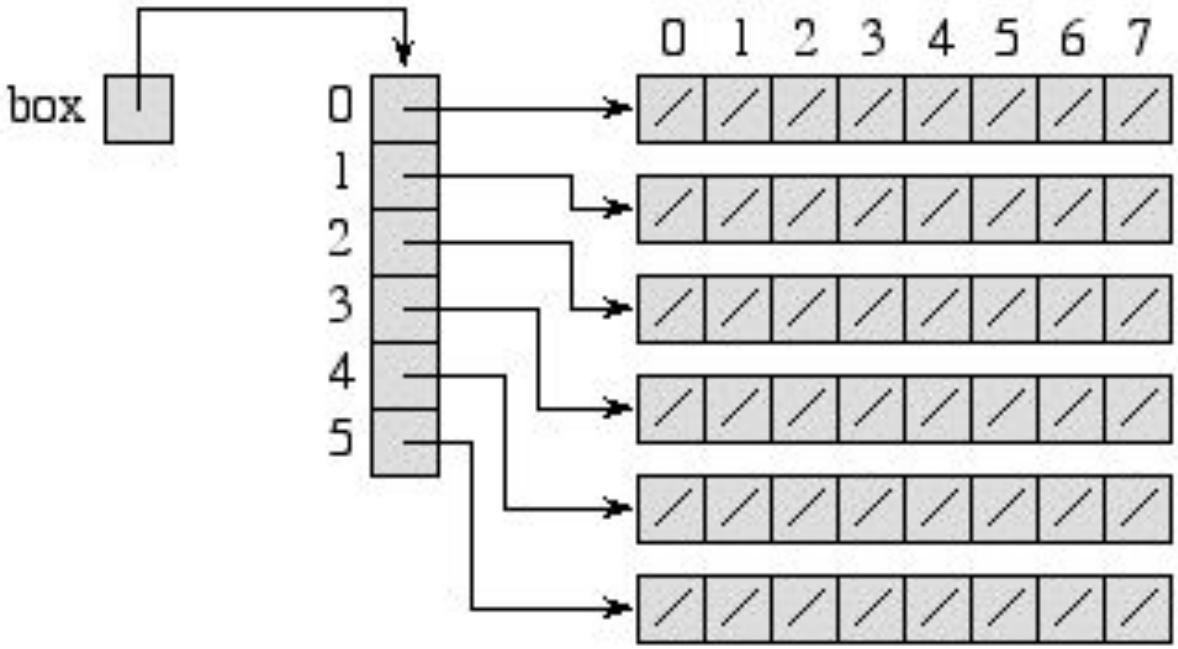
Copy:

System.arrayCopy (…)

Arrays.copyOfRange (…)

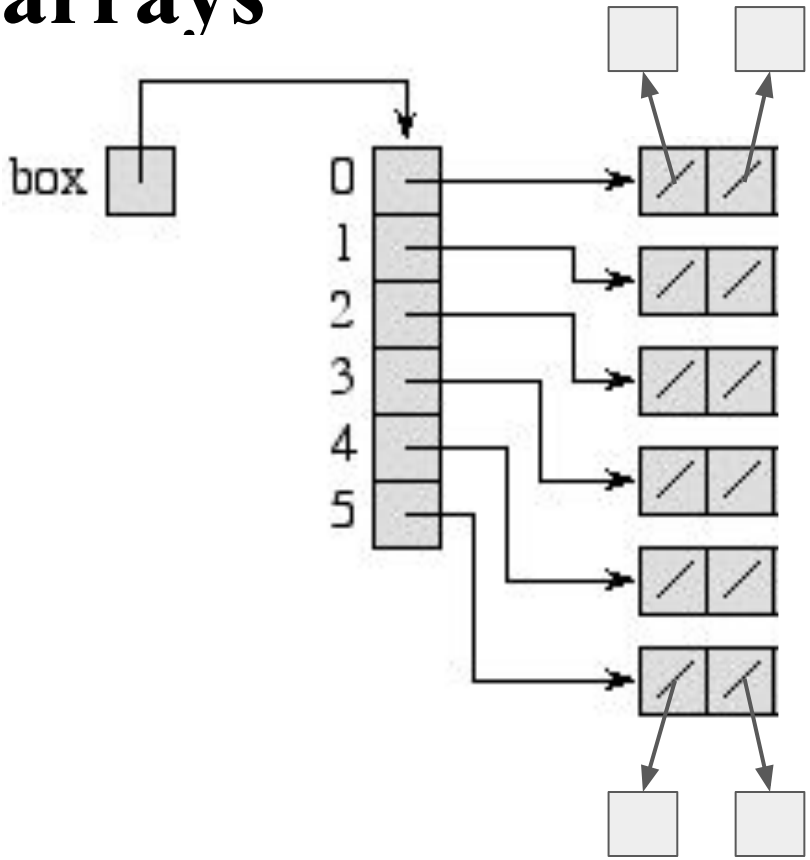
filling, applying specific math expression, set default values etc.

Arrays



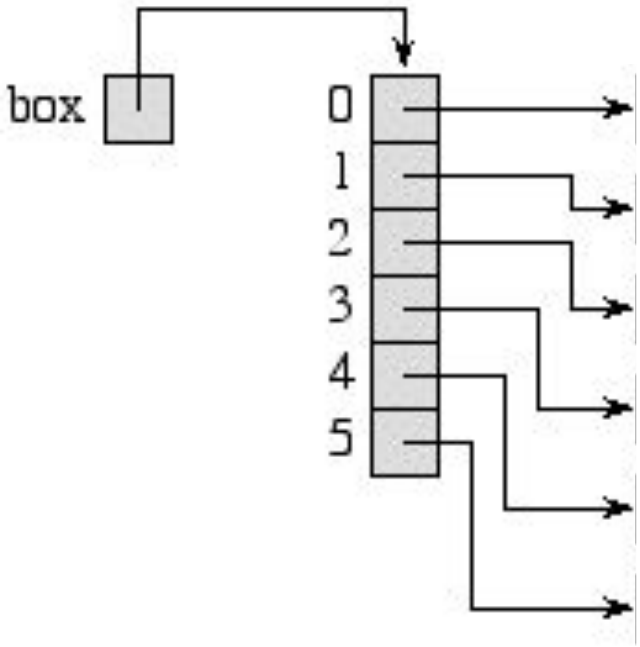

```
int[][] matrix;

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} };
```

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

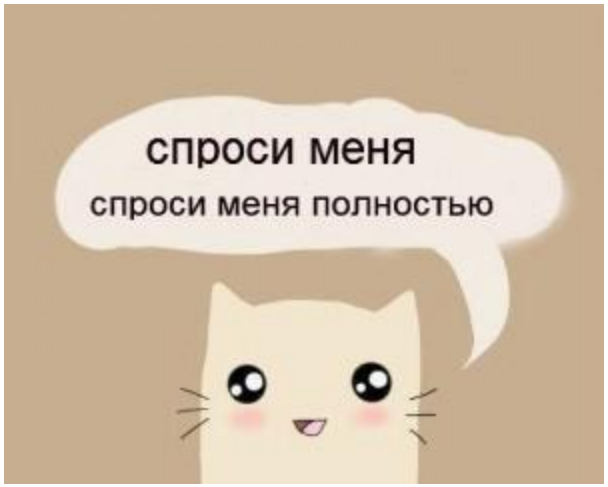
```
public static void main(String[] args) {

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

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

}
```

String + one argument

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);
    }
}
```



* ask me questions

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. }
```