Praktische Softwaretechnologie

Lecture 2.

Károly Bósa
(Karoly.Bosa@jku.at)

Research Institute for Symbolic Computation
(RISC)

Books

- James Gosling, Bill Joy, Guy Steele
  The JavaTM Language Specification

Books

- James Gosling, Bill Joy, Guy Steele
  The JavaTM Language Specification

- James Gosling, Bill Joy, Guy Steele, Gilad Bracha
  The JavaTM Language Specification (2nd/3rd edition)
  (online)

Books

- James Gosling, Bill Joy, Guy Steele
  The JavaTM Language Specification

- James Gosling, Bill Joy, Guy Steele, Gilad Bracha
  The JavaTM Language Specification (2nd/3rd edition)
  (online)

- Ken Arnold, James Gosling, David Holmes
  The JavaTM Programming Language

Books

- James Gosling, Bill Joy, Guy Steele
  The JavaTM Language Specification

- James Gosling, Bill Joy, Guy Steele, Gilad Bracha
  The JavaTM Language Specification (2nd/3rd edition) (online)

- Ken Arnold, James Gosling, David Holmes
  The JavaTM Programming Language

- S. Zakhour, S. Hommel, et al: *The JavaTM Tutorial* (online)

Books

- James Gosling, Bill Joy, Guy Steele
  The JavaTM Language Specification

- James Gosling, Bill Joy, Guy Steele, Gilad Bracha
  The JavaTM Language Specification (2nd/3rd edition) (online)

- Ken Arnold, James Gosling, David Holmes
  The JavaTM Programming Language

- S. Zakhour, S. Hommel, et al: *The JavaTM Tutorial* (online)

- Xiaoping Jia: *Object-Oriented Software Development Using Java – Principles. . .*

Books

- James Gosling, Bill Joy, Guy Steele
  The JavaTM Language Specification

- James Gosling, Bill Joy, Guy Steele, Gilad Bracha
  The JavaTM Language Specification (2nd/3rd edition) (online)

- Ken Arnold, James Gosling, David Holmes
  The JavaTM Programming Language

- S. Zakhour, S. Hommel, et al: *The JavaTM Tutorial* (online)

- Xiaoping Jia: *Object-Oriented Software Development Using Java – Principles. . .*

- Bruce Eckel: *Thinking in Java* (3rd edition online)

History of Java

- It began as “Oak” created by James Gosling in 1991 (the first version of Emacs)
- The first public version was issued in 1995
- Until the end of 1995: Integration into Netscape (JavaScript, too)
- The definition of the language in 1996 from Gosling, Bill Joy, (BSD Unix, csh, vi, a part of TCP/IP,. . .), Guy Steele (Common LISP Book, Scheme,. . .)

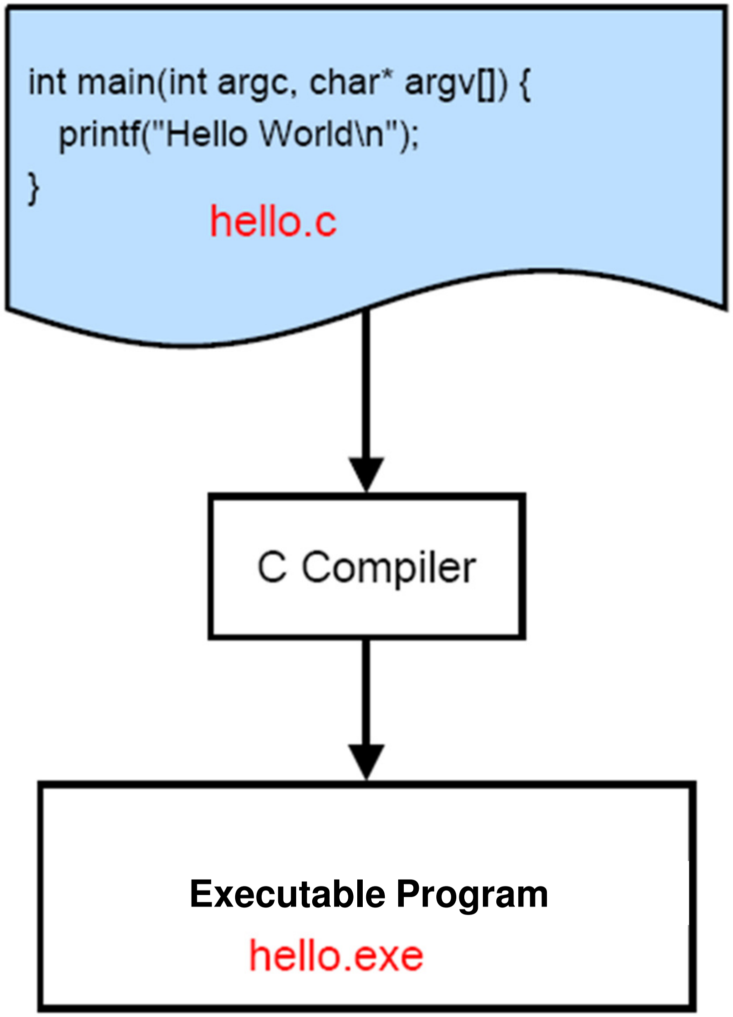
History of Java

- It began as “Oak” created by James Gosling in 1991 (the first version of Emacs)
- The first public version was issued in 1995
- Until the end of 1995: Integration into Netscape (JavaScript, too)
- The definition of the language in 1996 from Gosling, Bill Joy, (BSD Unix, csh, vi, a part of TCP/IP,. . .), Guy Steele (Common LISP Book, Scheme,. . .)

For comparison:

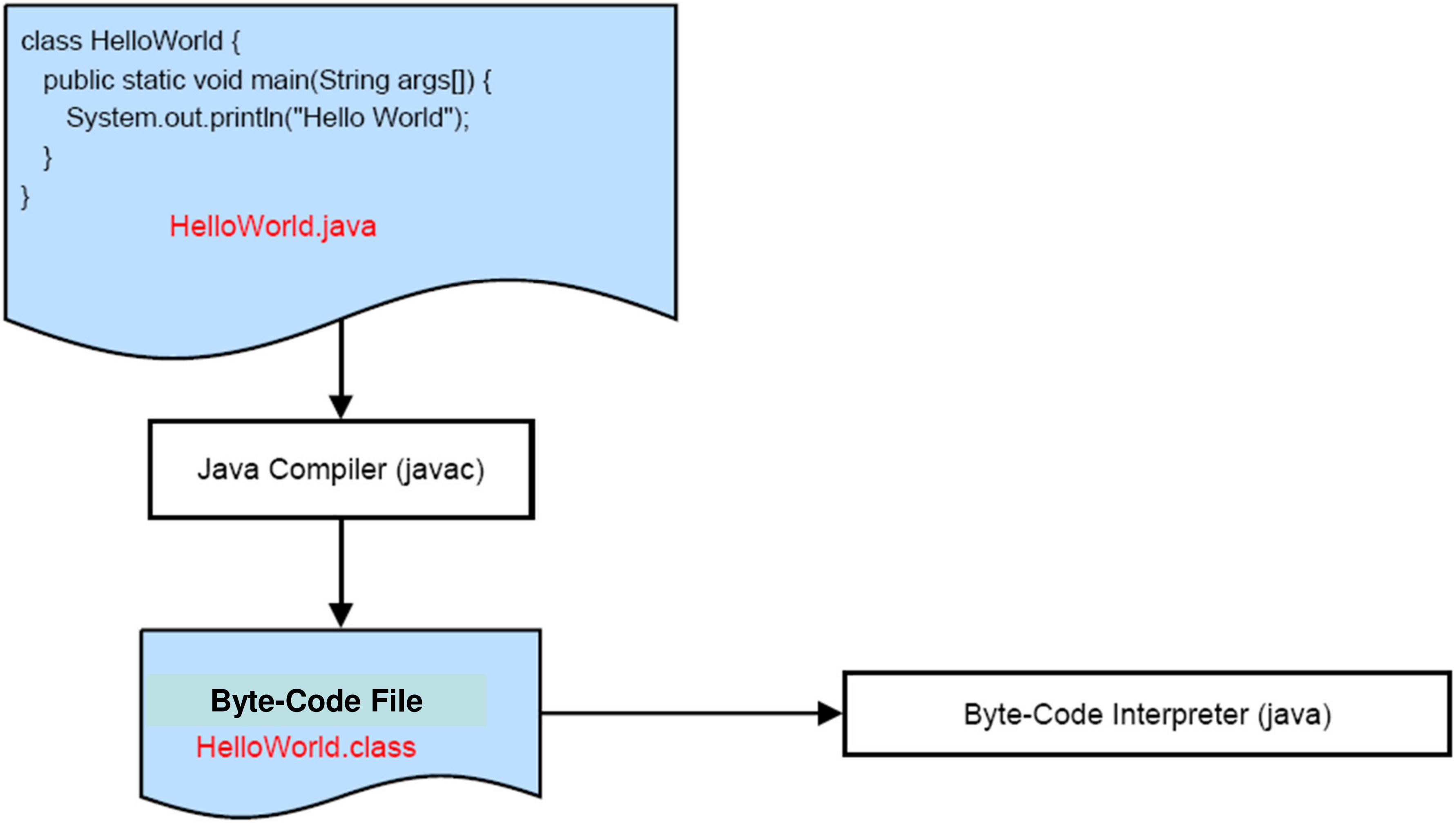
- The beginning of the World Wide Web 1990-1991
- Netscape: 1994
- Internet Explorer: 1995

Compilation of a C Program



```
int main(int argc, char* argv[]) {
   printf("Hello World\n");
}
```

Compilation of a Java Program



```
class HelloWorld {
  public static void main(String args[]) {
    System.out.println("Hello World");
  }
}
```

HelloWorld.java

Java Compiler (javac)

Byte-Code File

HelloWorld.class

Byte-Code Interpreter (java)

Consequence of Byte-Code

- .class files are platform independent:
  - It can run in different systems
  - The compiler is platform independent

Consequence of Byte-Code

- .class files are platform independent:
  - It can run in different systems
  - The compiler is platform independent
- The byte-code is very compact:
  - Useful for network transfer

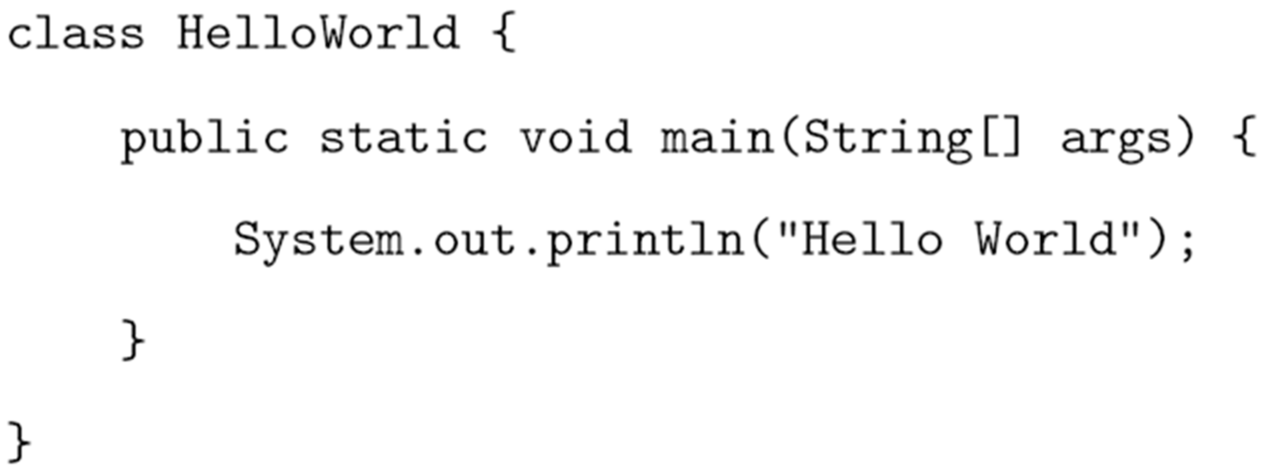
Consequence of Byte-Code

- .class files are platform independent:
  - It can run in different systems
  - The compiler is platform independent
- The byte-code is very compact:
  - Useful for network transfer
- The interpreter is able to revise the access rights
  - It is not necessary to trust in foreign codes

Consequence of Byte-Code

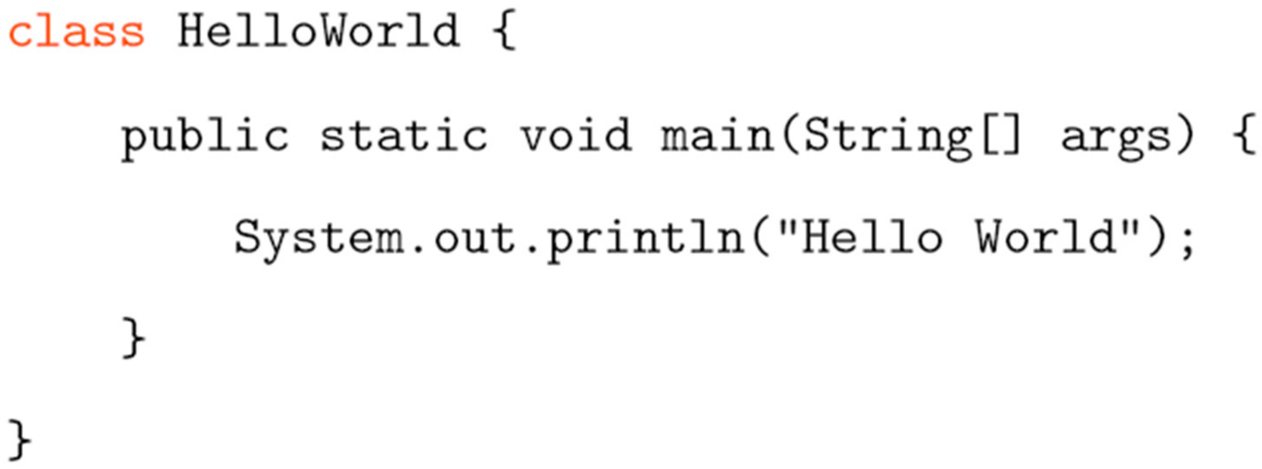
- .class files are platform independent:
  - It can run in different systems
  - The compiler is platform independent
- The byte-code is very compact:
  - Useful for network transfer
- The interpreter is able to revise the access rights
  - It is not necessary to trust in foreign codes
- It is not so fast as machine language
  - But it is fast with JIT

For Instance: HelloWorld



```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

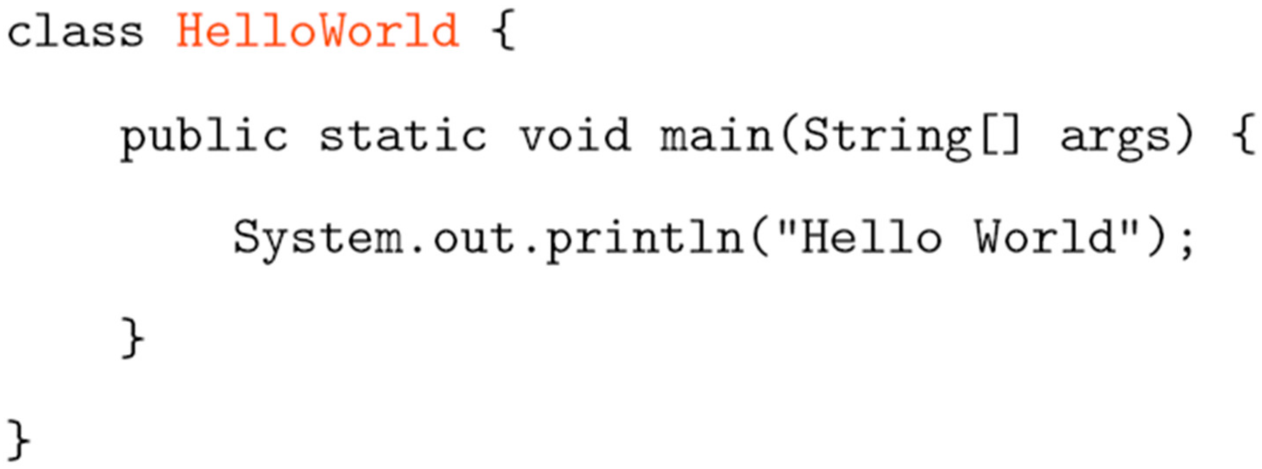
For Instance: HelloWorld



```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

The class keyword. The Java Programs consist of class- and interface-definitions.

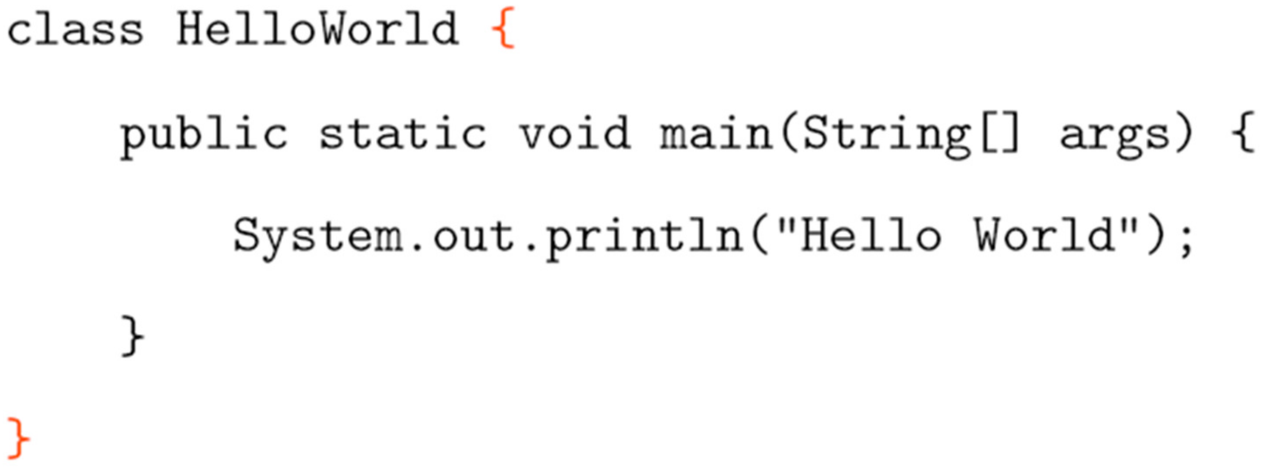
For Instance: HelloWorld



```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

The class names start with a capital letters. In case of more worlds: sepatatedByCapitalLetters (“camel case”).

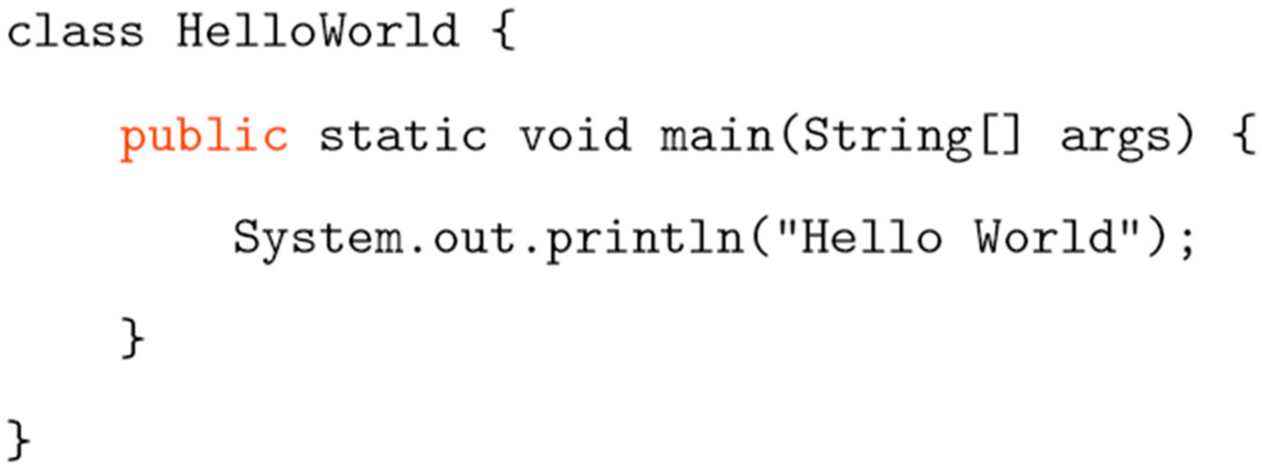
For Instance: HelloWorld



```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

Curly brackets is like in C (determine a block). The declarations of all attributes and methods are located between them.

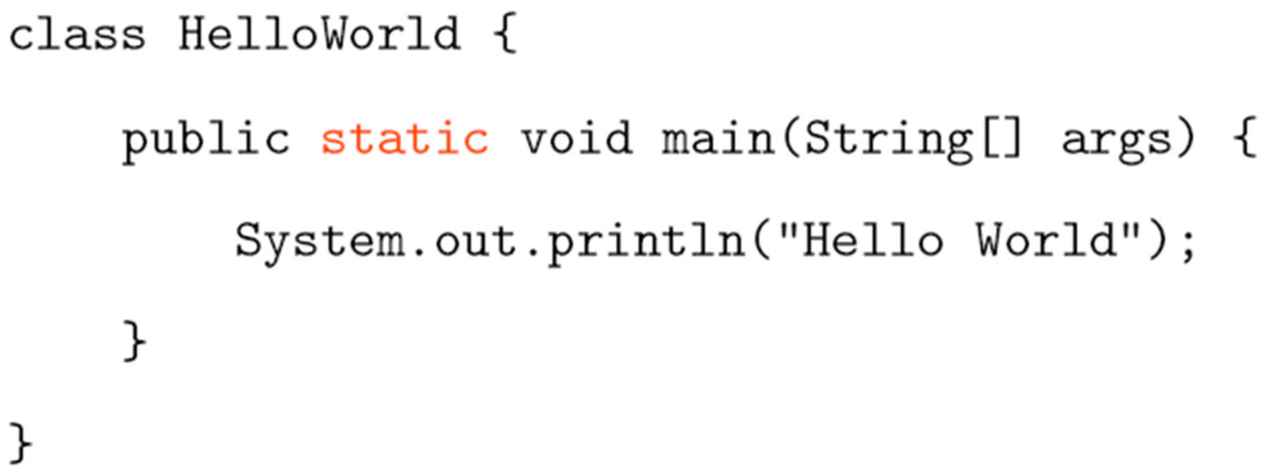
For Instance: HelloWorld



```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

The public keyword. Such a method can be called (available) from any other class.

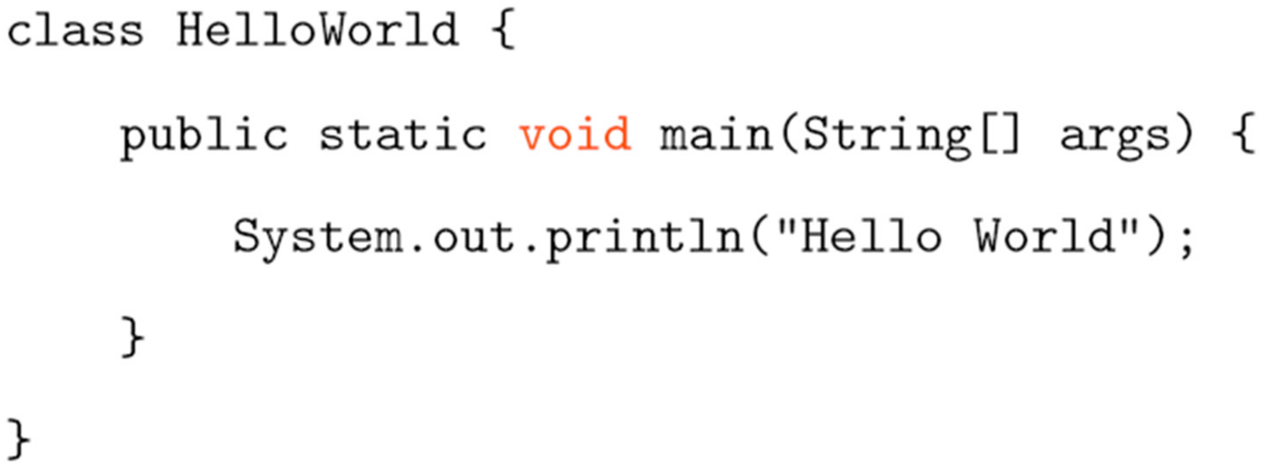
For Instance: HelloWorld



```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

The static keyword. Such a method is shared among all instances of a class.

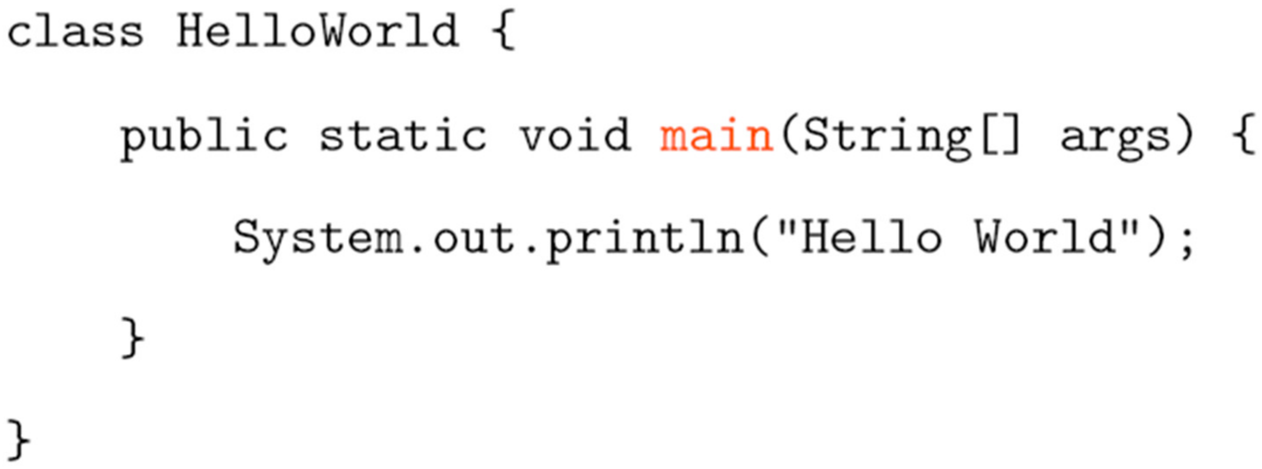
For Instance: HelloWorld



```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

The void is the “empty type”/”no type”. Such a method does not have a return value.

For Instance: HelloWorld



```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

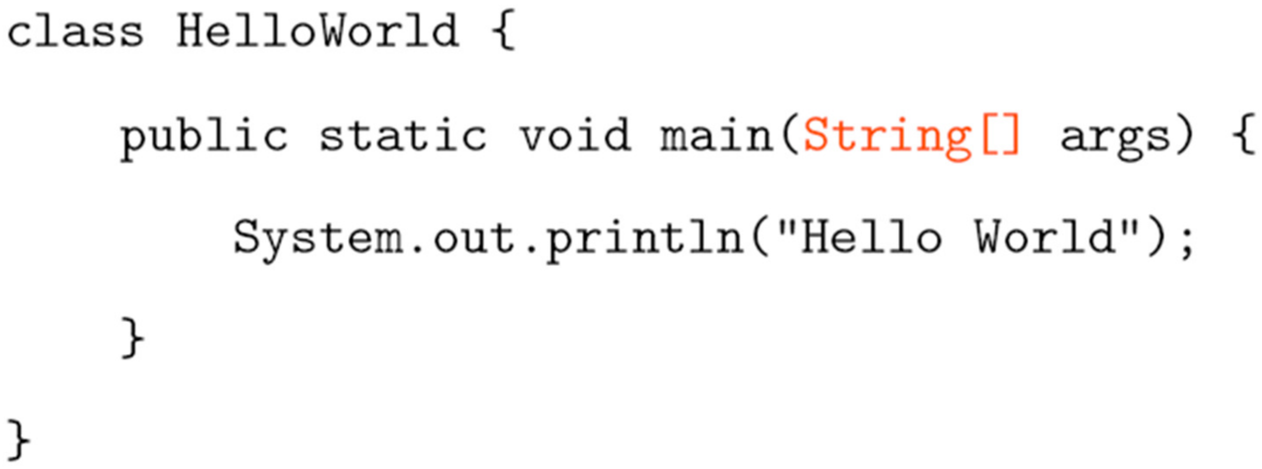
The name of the method. The method names starts with small letters. In case of more words “camel case” is used.

Method names called *main* can be called as a main program (they are always public and static).

java HelloWorld

- **Calls the HelloWorld.main(…)**

For Instance: HelloWorld

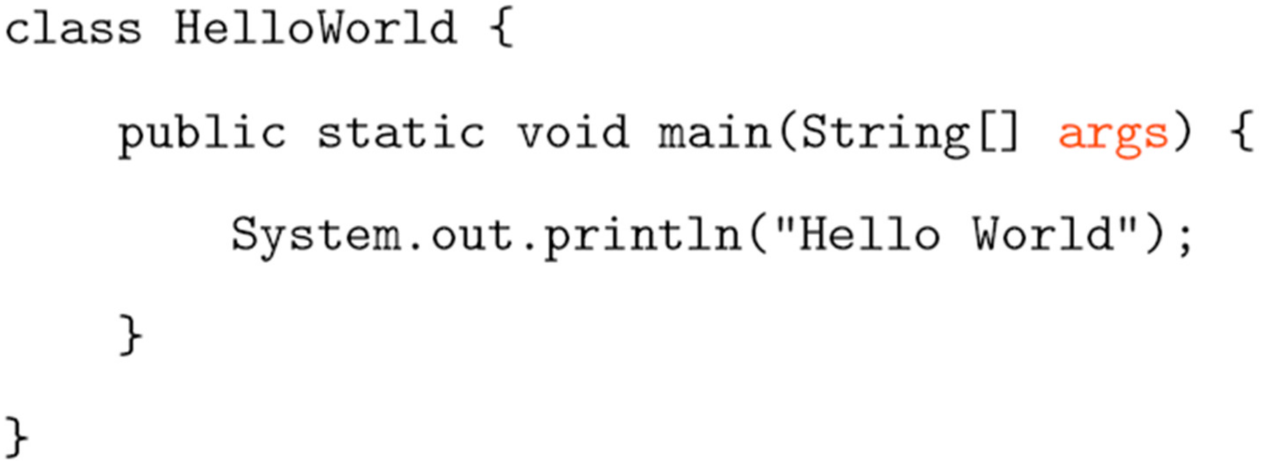


```
class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World");
    }
}
```

The class String is class of Unicode character chain.

The type String[] designates an array of Strings.

For Instance: HelloWorld

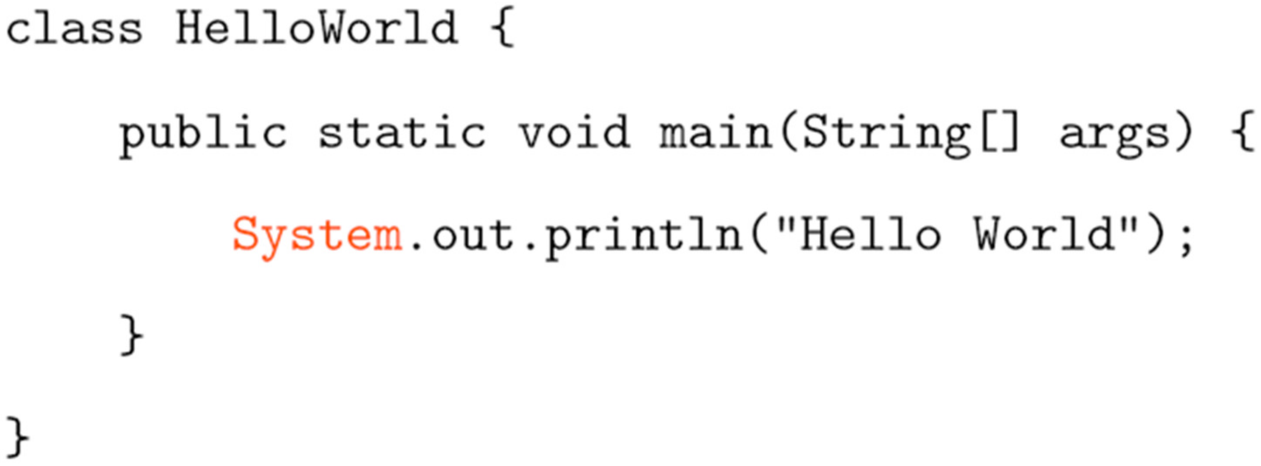


```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

The name of the arguments. The arguments, attributes and variables are written with small letter and “camel case”.

The arguments of a main program are taken from the command line.

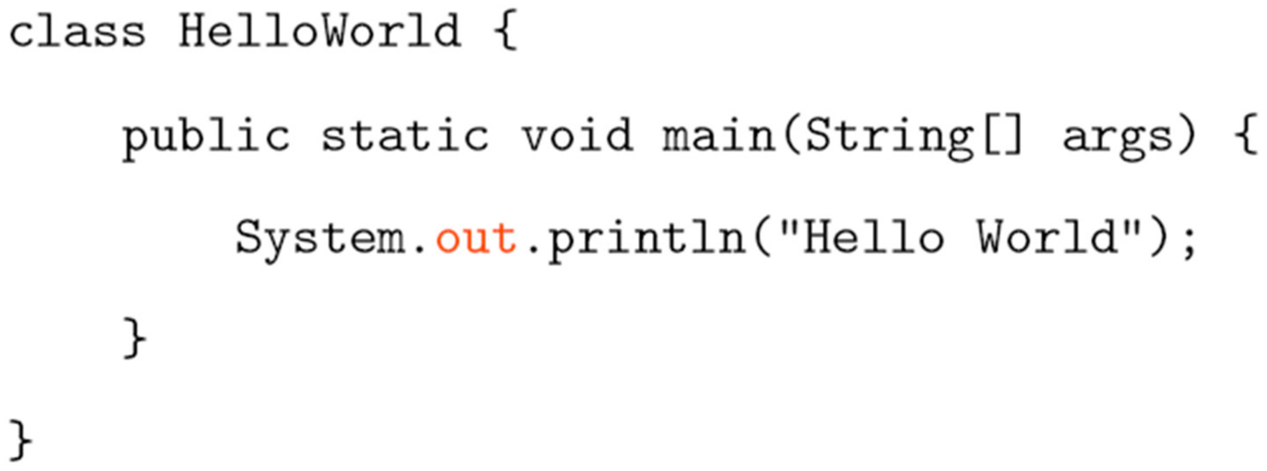
For Instance: HelloWorld



```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

The class *System* contains methods for accessing to the runtime environment: I/O, etc.

For Instance: HelloWorld

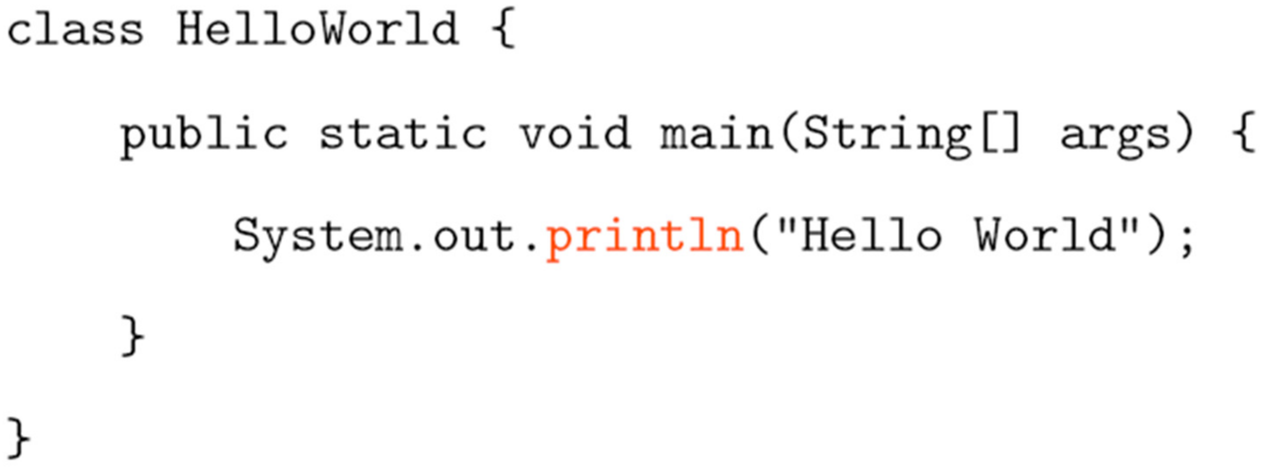


```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

out is a static attribute of the class System.

It denotes the standard output and it has a type *java.io.PrintStream*

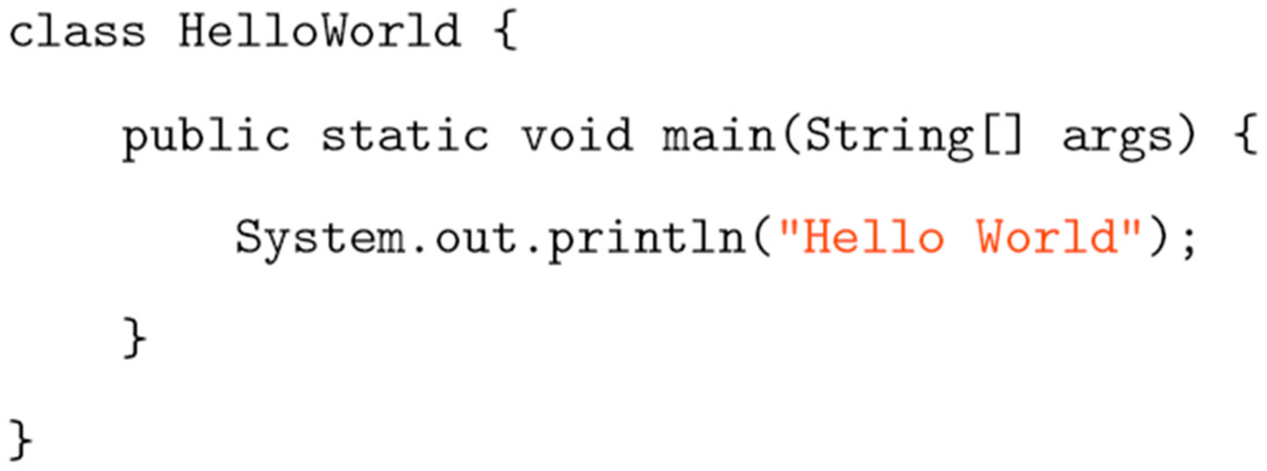
For Instance: HelloWorld



```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

The *println* is a method of the class PrintStream. It writes a String into the Stream, which will be followed by a new line character.

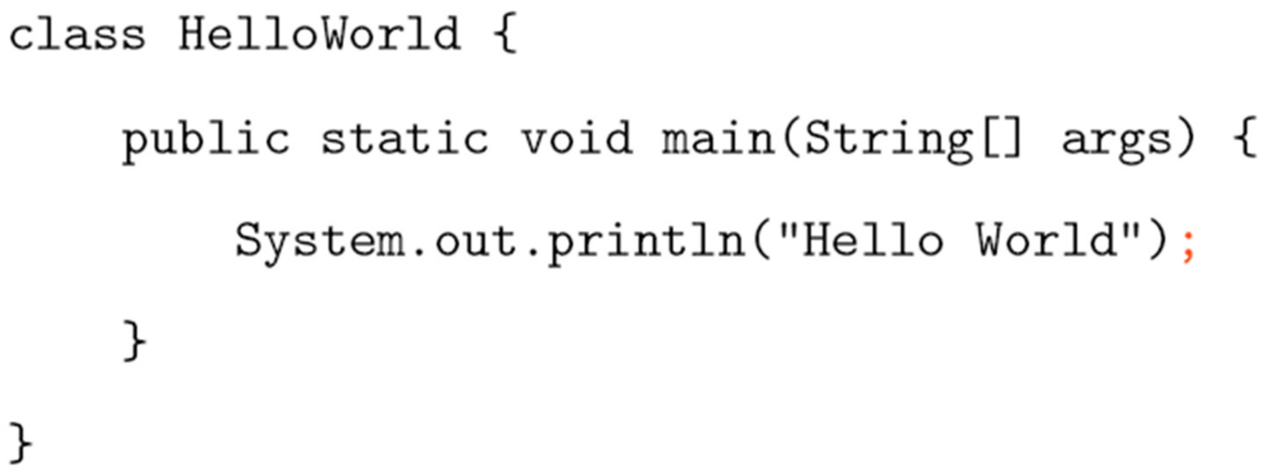
For Instance: HelloWorld



```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

It is a string literal

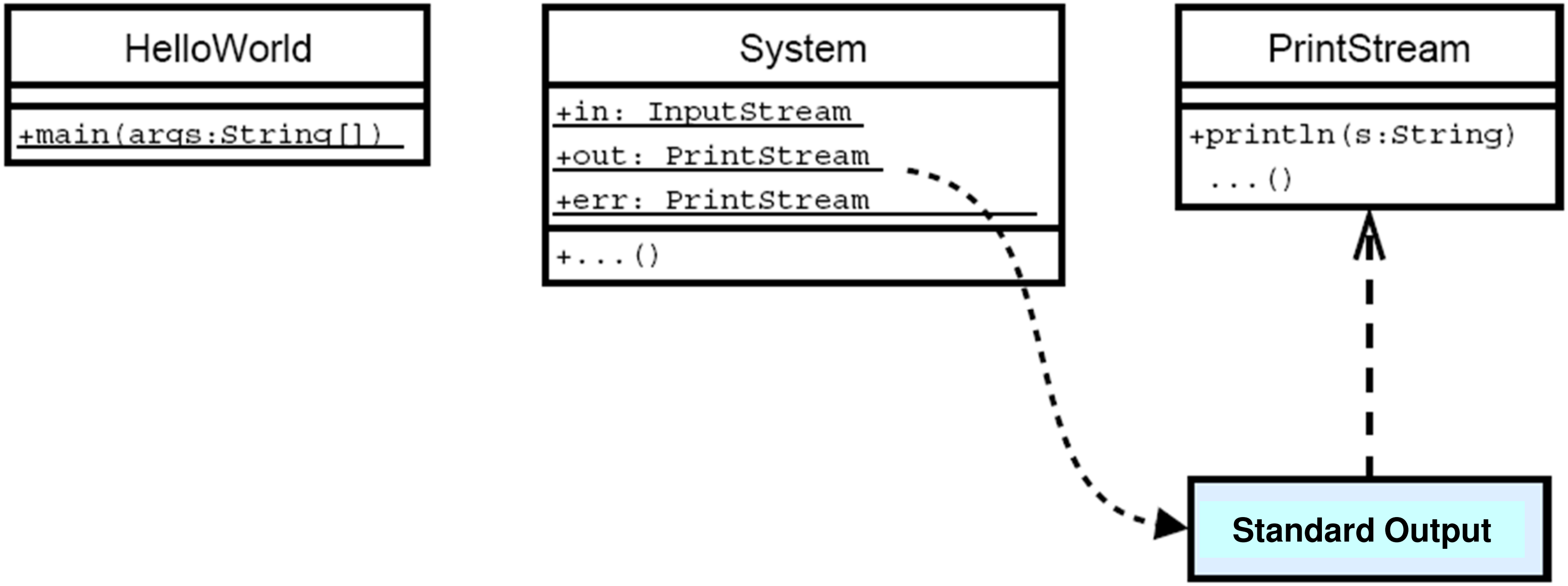
For Instance: HelloWorld



```
class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello World");

    }

}
```

Every statement/command ends with semicolon.

HelloWorld **Diagram**



Data Types

There are 2 kinds of data types

- ***Primitive types:*** int, char, float, etc (like the corresponding types in C)

Data Types

There are 2 kinds of data types

- ***Primitive types:*** int, char, float, etc (like the corresponding types in C)
- ***Reference types:*** references of object (similar to the pointer of struct in C)

Data Types

There are 2 kinds of data types

- ***Primitive types:*** int, char, float, etc (like the corresponding types in C)
- ***Reference types:*** references of object (similar to the pointer of struct in C)

Arrays, String, … are directly supported by the language, however they are object types ultimately.

Primitive Types

- `byte`: $-2^7 \ldots +2^7 - 1$
- `short`: $-2^{15} \ldots +2^{15} - 1$
- `int`: $-2^{31} \ldots +2^{31} - 1$
- `long`: $-2^{63} \ldots +2^{63} - 1$
- `float`: 32-bit IEEE 754 **Floating Point Number**
- `double`: 64-bit IEEE 754 **Floating Point Number**
- `boolean`: **true or false**
- `char`: **a 16-bit Unicode character**

Primitive Types

- `byte`: $-2^7 \ldots +2^7-1$
- `short`: $-2^{15} \ldots +2^{15}-1$
- `int`: $-2^{31} \ldots +2^{31}-1$
- `long`: $-2^{63} \ldots +2^{63}-1$
- `float`: 32-bit IEEE 754 **Floating Point Number**
- `double`: 64-bit IEEE 754 **Floating Point Number**
- `boolean`: **true or false**
- `char`: **a 16-bit Unicode character**

- Machine independent

Primitive Types

- byte: $-2^7 \ldots +2^7 - 1$
- short: $-2^{15} \ldots +2^{15} - 1$
- int: $-2^{31} \ldots +2^{31} - 1$
- long: $-2^{63} \ldots +2^{63} - 1$
- float: 32-bit IEEE 754 **Floating Point Number**
- double: 64-bit IEEE 754 **Floating Point Number**
- boolean: **true or false**
- char: **a 16-bit Unicode character**

- No unsigned type

Primitive Types

- byte: $-2^7 \ldots + 2^7 - 1$
- short: $-2^{15} \ldots + 2^{15} - 1$
- int: $-2^{31} \ldots + 2^{31} - 1$
- long: $-2^{63} \ldots + 2^{63} - 1$
- float: 32-bit IEEE 754 **Floating Point Number**
- double: 64-bit IEEE 754 **Floating Point Number**
- boolean: **true or false**
- char: **a 16-bit Unicode character**

- Default values: 0, false, …

Literals

- int: 23, 027 (oktal), 0x17 (hex)
- long: 9223372036854775807L
- float: 12.34f, 1.234e1f
- double: 12.34, 1.234e1, 12.34d
- boolean: true, false
- char: ’A’, ’Ä’, ’\n’, ’\’’, ’\"’, ’\\’,

Variable Declarations

- **Initialized with default value:**

```
int i;
```

- **With initialization:**

```
int i = 23;
```

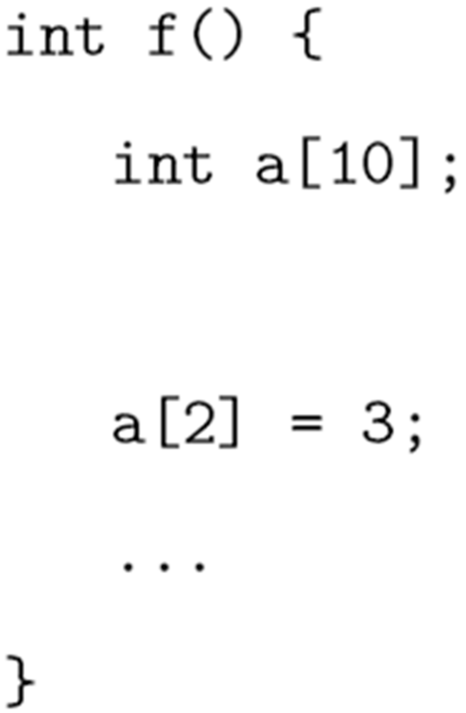
- **In the middle of a block as well:**

```
int f(int i) {
    int j;
    …do something with i and j…
    boolean jPositiv = (j >0);
    …
}
```

Arrays

Similar as in C, but:

Always allocated dynamically!



```
int f() {
    int a[10];


    a[2] = 3;

    ...

}
```

Such as in C, it does not work!

Arrays 2.

In Java:

```
int f() {
    int[] a;  //this is a reference to an array whose elements are int

    a = new int[10]; //place for 10 integer value are allocated

    a[2] = 3;
    …
}
```

Arrays are realized like object → int[] is a reference type

Arrays 3.

Number of elements in an Array:

a.length

a[0] is the first, a[a.length-1] is the last element

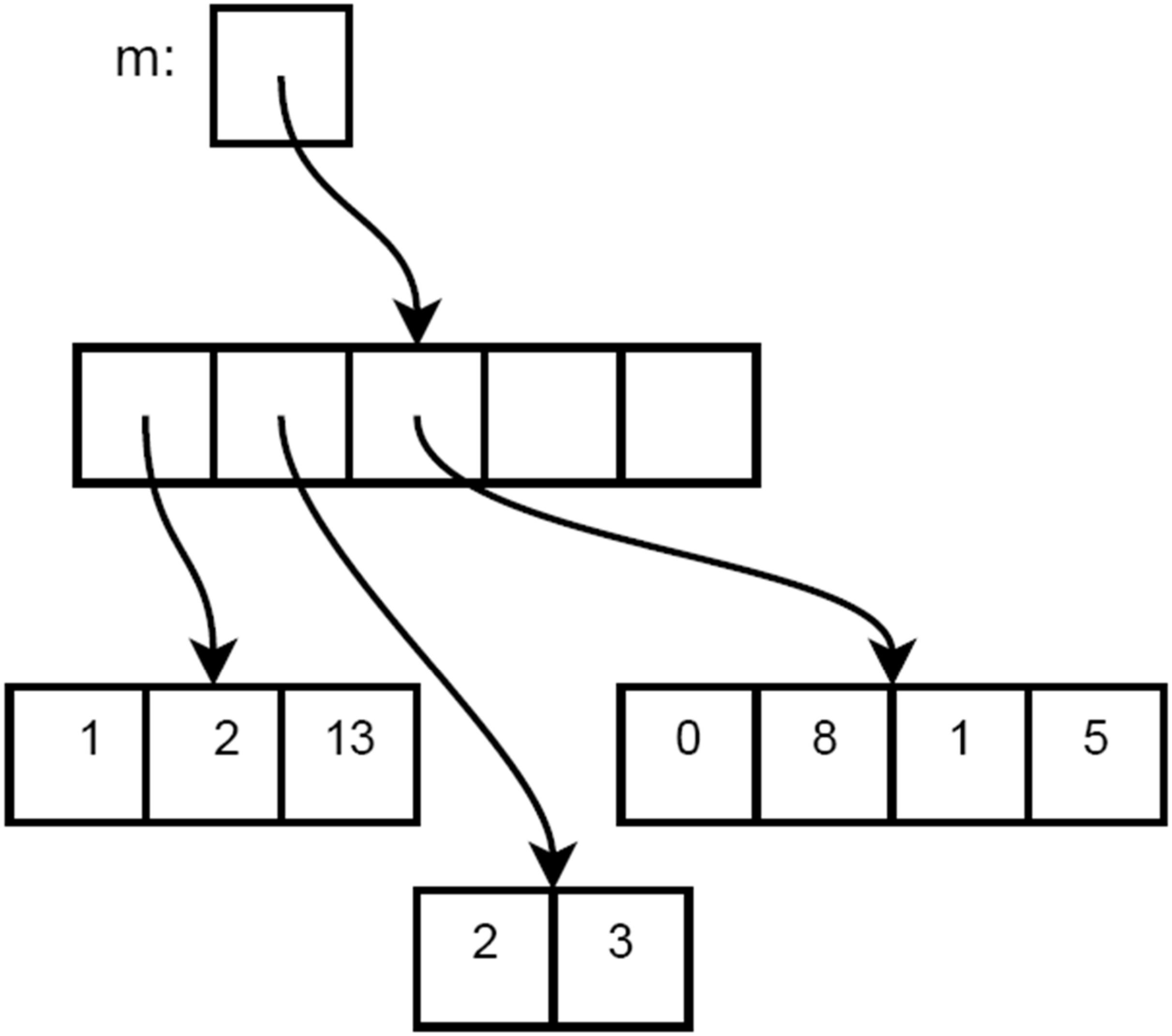
There is not Pointer-Aritmethic:

in C: a+1 is a pointer to the array from its 2. element

In Java: an independent reference to the array and index are needed

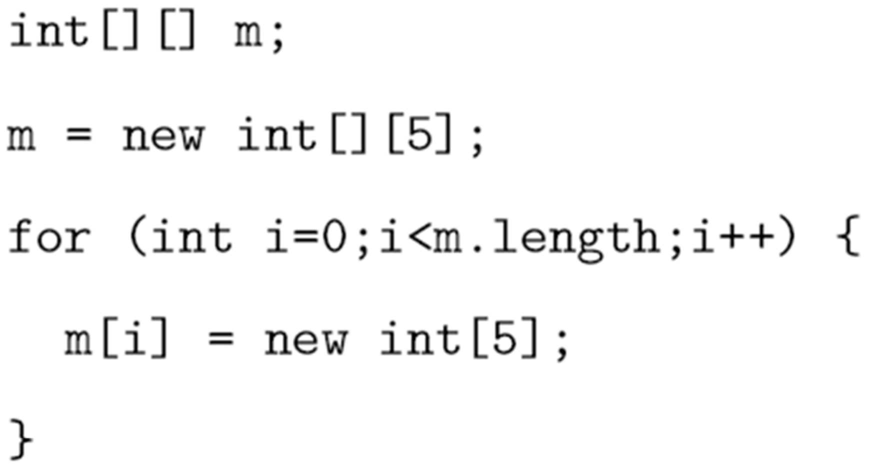
Matrices/Multidimensional Arrays

```
int[][] m;
```



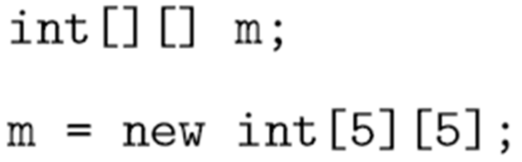
Allocation of Multidimensional Arrays

A 5 times 5 Array/Matrix:



```
int[][] m;

m = new int[][5];

for (int i=0;i<m.length;i++) {

  m[i] = new int[5];

}
```

Or, as a shortcut:



```
int[][] m;

m = new int[5][5];
```

Strings

Unicode Strings

Literal: “This is a row\nThis is another row”

Concatenation of Strings:

String a = “This is a row”;

String b = “This is another row”;

String twoRows = a + “\n” + b;

Addition of other types:

String s = “The answer is: “ + 42;

→ *The outcome will be:* “The answer is: 42”

More about Strings

Strings are also reference types:

```
String a = “World”;

String b = a;
```

Here only the reference was copied (not the value)

The String objects never change after their creation:

```
a = “Hello ” + a;
```

More about Strings

Strings are also reference types:

```
String a = “World”;

String b = a;
```

Here only the reference was copied (not the value)

The String objects never change after their creation:

```
a = “Hello ” + a;
```

- **It creates a new string:** “Hello World”

More about Strings

Strings are also reference types:

```
String a = “World”;

String b = a;
```

Here only the reference was copied (not the value)

The String objects never change after their creation:

```
a = “Hello ” + a;
```

- **It creates a new string:** “Hello World”
- **The reference of the new String is stored in** a

More about Strings

Strings are also reference types:

```
String a = “World”;

String b = a;
```

Here only the reference was copied (not the value)

The String objects never change after their creation:

```
a = “Hello ” + a;
```

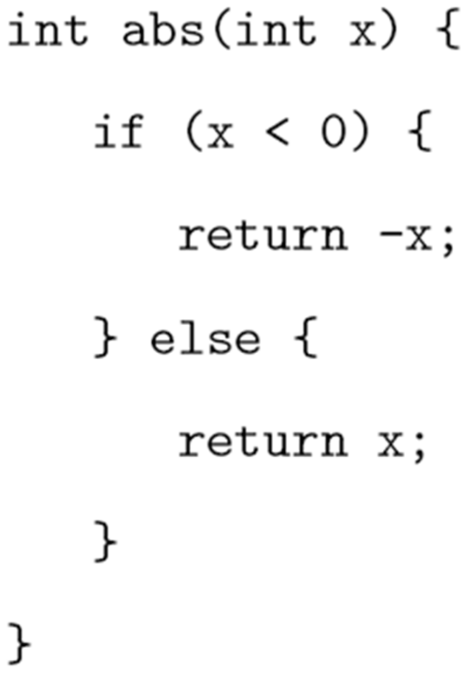
- **It creates a new string:** “Hello World”
- **The reference of the new string is stored in** a
- b **still refers to the old string**

Operators

Operators are similar as in C:

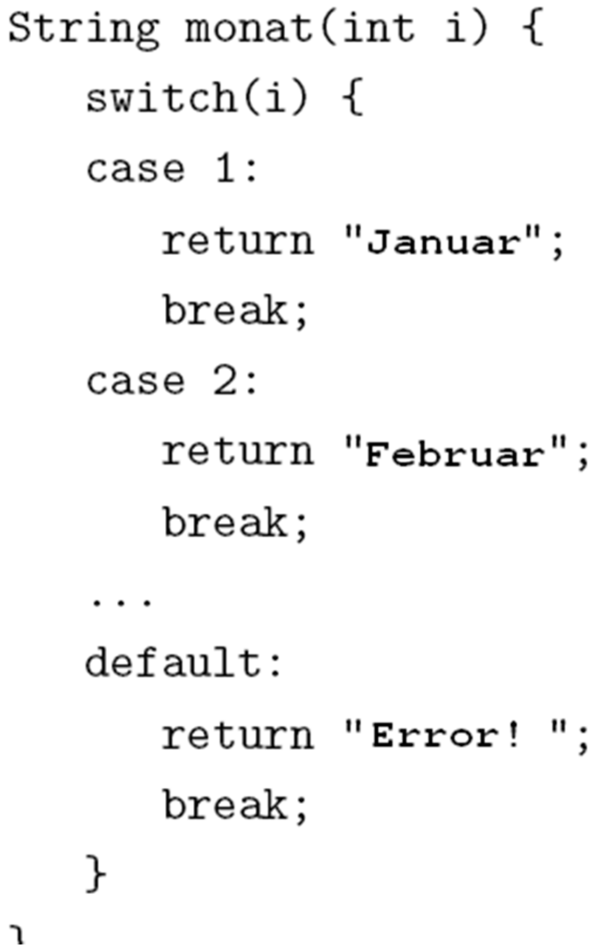
- **Arithmetik:** +, -, *, /, %
- **Bind of Variables:** =, +=, -=, …
- **Comparison:** ==, !=, <, >, <=, >=
- **Incrementing/Decrementing:** ++, --
- **Logical Operations:** &&, ||, !
- **Logical Operations on Bits:** &, |, ^
- **Conditional Structures:** ? :
- **Object Operators:** new, instanceof

Control Structures: if-then-else



```
int abs(int x) {
   if (x < 0) {
      return -x;
   } else {
      return x;
   }
}
```

Control Structures: switch

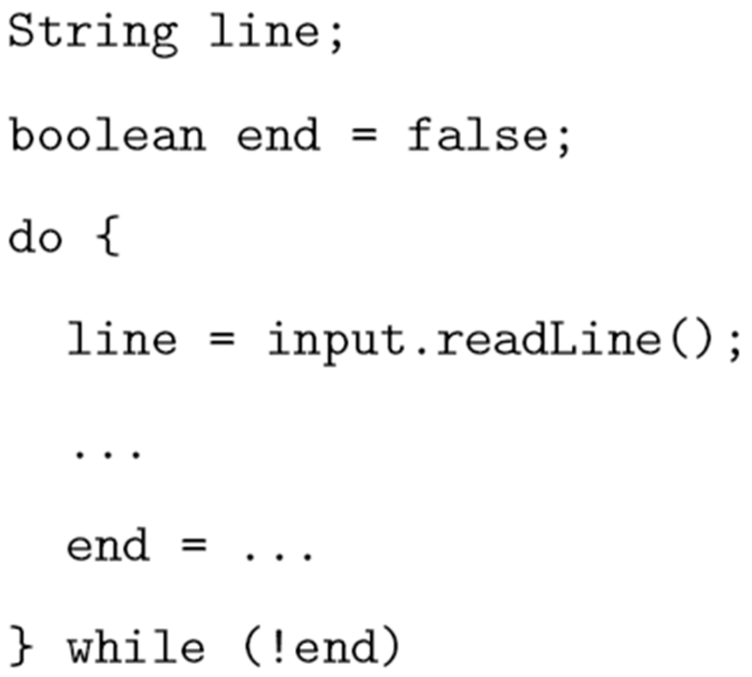


```
String monat(int i) {
   switch(i) {
   case 1:
      return "Januar";
      break;
   case 2:
      return "Februar";
      break;
   ...
   default:
      return "Error! ";
      break;
   }
}
```

Control Structures: while

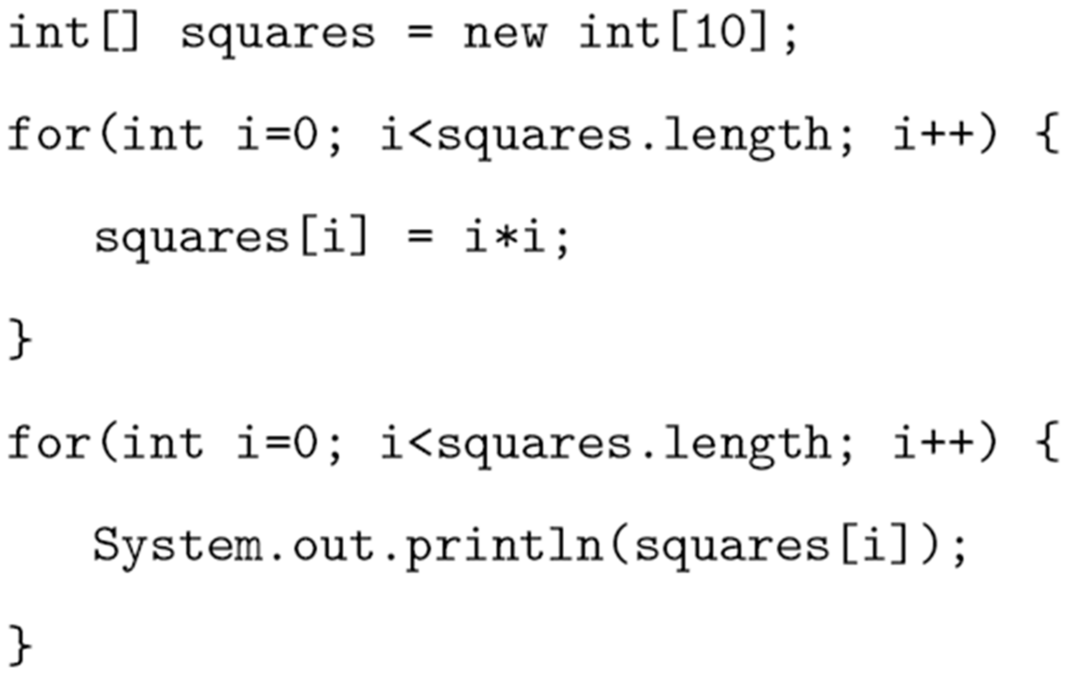
```
int   digitsum (int i) {
   int q = 0;
   while (i != 0) {
      q += i % 10;
      i /= 10;
   }
   return q;
}
```

Control Structures: do-while



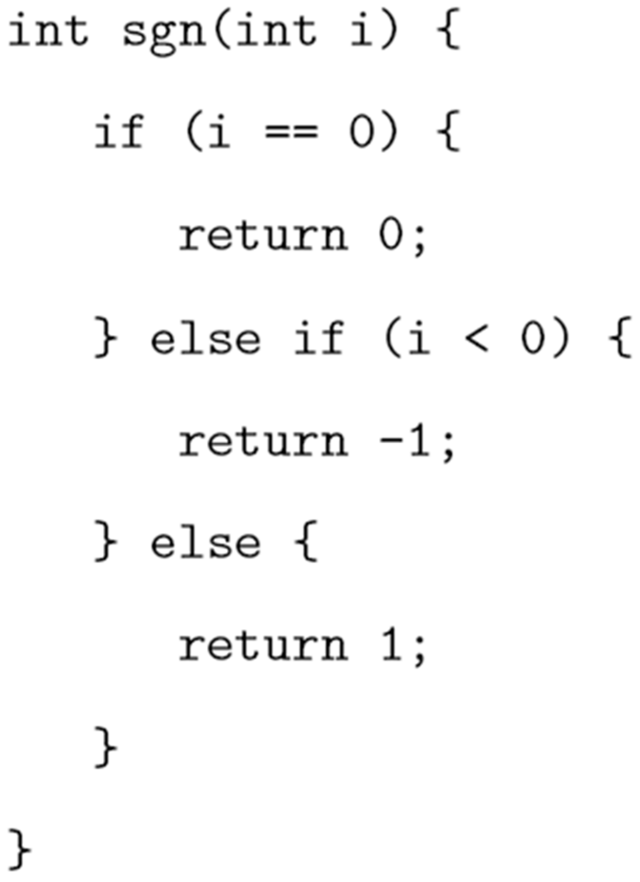
```
String line;
boolean end = false;
do {
  line = input.readLine();
  ...
  end = ...
} while (!end)
```

Control Structures: for



```
int[] squares = new int[10];

for(int i=0; i<squares.length; i++) {

   squares[i] = i*i;

}

for(int i=0; i<squares.length; i++) {

   System.out.println(squares[i]);

}
```

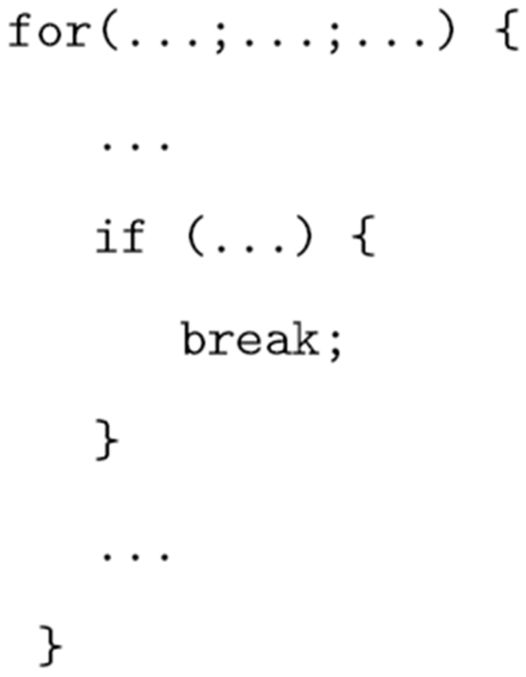
Control Structures: return



```
int sgn(int i) {
   if (i == 0) {
      return 0;
   } else if (i < 0) {
      return -1;
   } else {
      return 1;
   }
}
```

Control Structures: break/continue

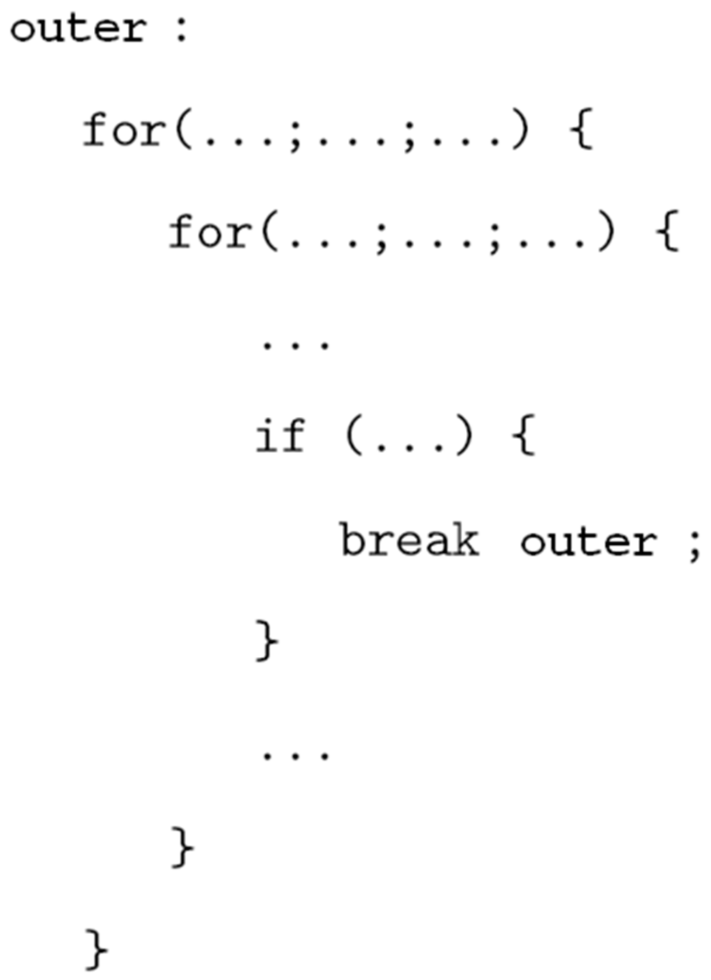
Without label:



```
for(...;...;...) {
   ...
   if (...) {
      break;
   }
   ...
 }
```

Control Structures: break/continue

With label:



```
outer :
  for(...;...;...) {
    for(...;...;...) {
      ...
      if (...) {
        break outer ;
      }
      ...
    }
  }
```

Static Methods

So far there is not any object that was created by ourselves.

➔ There is not any method belonging to such an object

Static Methods

So far there is not any object that was created by ourselves.

➔ There is not any method belonging to such an object

<u>The main program:</u>

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

Static Methods

So far there is not any object that was created by ourselves.

➔ There is not any method belonging to an object

The main program:

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

Our own static methods:

```
public static int myMethod(int i)
```

Static Methods

So far there is not any object that was created by ourselves.

➔ There is not any method belonging to such an object

<u>The main program:</u>

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

<u>Our own static methods:</u>

```
public static int myMethod(int i)
```

<u>Calling from the main:</u>

```
result = myMethod(23);
```

Static Methods

So far there is not any object that was created by ourselves.

➔ There is not any method belonging to such an object

The main program:

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

Our own static methods:

```
public static int myMethod(int i)
```

Calling from the main:

```
result = myMethod(23);
```

Global variables are static as well:

```
static int[] qu;
```

.java files

- **Generally every class is defined in a** .java **file.**
- **The name of the file has to correspond with the name of the class. For instance, the content of the file** Exercise.java**:**

```
class Exercise {
    static int counter;
    …
    static double f(int i) {
        …
    }
    …
    public static void main(String args[]) {

        …
    }

}
```

Comments

There are 3 kinds of the comments:

- Comment in one line: //
- Comment in more lines: /* … */
- .JavaDoc comment: /** … */

Recommended to Read

Reading and completing the course material from the online Java Tutorial:

http://download.oracle.com/javase/tutorial/java/index.html

- Object Oriented Concept
- Language Basics

Exercise 2 Deadline: 02.04.2014

Hallo World – Advanced Version

java Hallo

➔Who is there?

java Hallo Tom

➔Hallo Tom!

java Hallo Tom Tim

➔Hallo Tom and Tim!

java Hallo Tracy Tom Tim **(Attention: Arbitrary many arguments)**

➔Hallo Tracy, Tom and Tim!

Exercise 3 Deadline: 02.04.2014

Matrix Product of two matrices (4x5 and 5x4 at least)

- Matrices can be initialized from the source code.
- Output should be printed out in a “nice” matrix format on the screen.