Programming Basics

Digital Urban Visualization. People as Flows.

28.09.2015

iA
zuend@arch.ethz.ch
treyer@arch.ethz.ch

Programming?

Programming is the interaction between the programmer and the computer. The computer evaluates the formal text you write and executes the instructions of the text.

Programming?

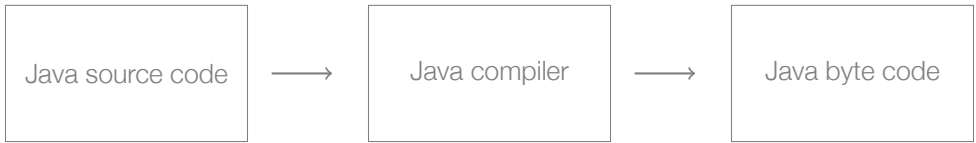
Programming means to solve problems and to give the instructions on how to solve them to the computer.

Sorting: [9,2,8,1,7] → [1,2,7,8,9]

From Text Files to an Application

simplified explanation

Java is a **high-level programming** language. This means, we can write a **human readable** set of instructions. But an interpreter/compiler is needed to translate the text (a.k.a. code) to a computer readable format.



Object Oriented Programming

In the real world, you'll often find many individual objects all of the same kind. There may be thousands of other bicycles in existence, all of the same make and model. Each bicycle was built from the same set of blueprints and therefore contains the same components. In object-oriented terms, we say that your bicycle is an instance of the class of objects known as bicycles. A class is the blueprint from which individual objects are created.

Where it all Begins

the main function

The compiler needs to know, where to begin a program. This is defined by a function called **main**, i.e.

public static void main(String[] args) { …}

Last week we added this function automatically when we created the *HelloWorld* program.

The compiler knows the main keyword and will start from there.

Values and Primitive Data Types

Values are for example: *3.1415*, *23*, or *Hello World!*.
Each value has a **type** and we can use variables of specific primitive data types to store values. The three main data types we will be using are:

– *int*: int (or integers) are whole numbers.
– *double*: doubles are real numbers.
– *char*: is any character. We will normally use a more advanced data structure called *string* which can be understood as a list of characters.
– *boolean*: has a value of either *true* or *false*.

Values and Data Types

assign values to variables

A **variable** is a name that refers to a value. In Java, you first have to define the variable and its type. When you use an assignment statement (=), Java will set the variable to the corresponding value. This accounts for any type (primitive data types and instances of classes).

```
int a;
a = 23;
System.out.println(a);
```

```
double b;
b = 3.1415;
System.out.println(b);
```

```
boolean c;
c = true;
System.out.println(c);
```

```
string e;
e = "Hello World!";
System.out.println(e);
```

```
char d;
d = 'a';
System.out.println(d);
```

Operators

Operators represent a computation you want to do with values or variables. For example the operators “+, *, -, /” perform addition, multiplication, subtraction and division, respectively. It is important to ensure that the data type is correct when you use such an operation.

```
double a, b, c;
a = 1;
b = 2;
c = a / b;
System.out.println(c);
```

Output: ?

```
int a, b, c;
a = 1;
b = 2;
c = a / b;
System.out.println(c);
```

Output: ?

Operators

Operators represent a computation you want to do with values or variables. For example the operators “+, *, -, /” perform addition, multiplication, subtraction and division, respectively. It is important to ensure that the data type is correct when you use such an operation.

```
double a, b, c;
a = 1;
b = 2;
c = a / b;
System.out.println(c);
```

Output: 0.5

```
int a, b, c;
a = 1;
b = 2;
c = a / b;
System.out.println(c);
```

Output: ?

Operators

Operators represent a computation you want to do with values or variables. For example the operators “+, *, -, /” perform addition, multiplication, subtraction and division, respectively. It is important to ensure that the data type is correct when you use such an operation.

```
double a, b, c;
a = 1;
b = 2;
c = a / b;
System.out.println(c);
```

Output: 0.5

```
int a, b, c;
a = 1;
b = 2;
c = a / b;
System.out.println(c);
```

Output: 0

Lists

There exist many types of lists in Java. We will use the *ArrayList*. To work with a list, first initialize a variable of the *ArrayList* type.

ArrayList<double> myList;

The list can only contain one type of object/data types, thus we have to define it at initialisation, e.g. *double* above. To add elements, we use the add function of *ArrayLists*.

myList.add(2.3);

This appends 2.3 to the end of the list.

Lists

To add elements, we use the add function of *ArrayLists*.

```
myList.add(2.3);
```

This appends 2.3 to the end of the list. To access an element in the list, we use the *get* functionality.

```
double a;
a = myList.get(0);
```

a has stored the value 2.3 now. **Be aware** that counting normally starts at zero in programming, thus the first element in the list has the index **0**.

Lists

To access an element in the list, we use the *get* functionality.

```
double a;
a = myList.get(0);
```

a has stored the value 2.3 now. **Be aware** that counting normally starts at zero in programming, thus the first element in the list has the index **0**. It is also possible to delete elements for this, *ArrayList* has the *remove* functionality.

```
myList.remove(0);
```

Now the list is empty again. There are many other functionalities for *ArrayList*, check the internet, e.g. [1].

[1]: docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

Boolean Expressions

Boolean expressions are either *true* or *false*. The result is of type Boolean. They are most often used to check for conditions in **conditional statements**. Most of the time, relational operators are used.

Relational operators: == , *!=*, <, >, <=, >=
They stand for: equal, not equal, smaller, greater, smaller-equal, greater-equal.
Do not confuse = with == , the first is the assignment operator, the other one the relational operator!

Logical Operators

Logical operators are *or (||)*, *and (&&)* and *not (!)*. They are used to construct logical expressions. The following examples make the concept very understandable, if you read them with an *is* at the beginning and formulate them as a question.

```
x < 2 && x >= 0
x == 0 || x != 0
!(x == 0 || x > 100)
```

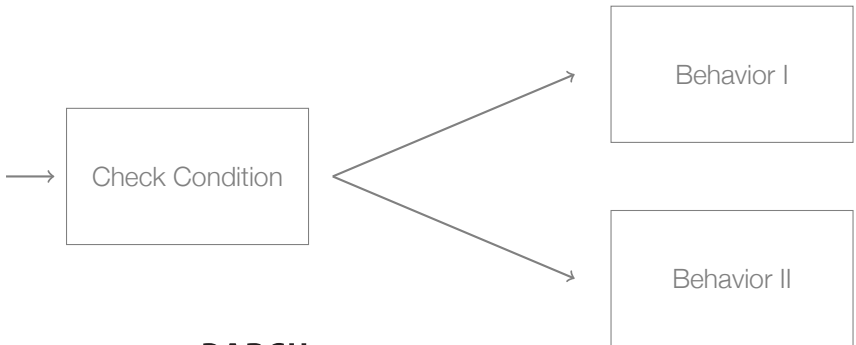
Logical Operators

Logical operators are *or (||)*, *and (&&)* and *not (!)*. They are used to construct logical expressions. The following examples make the concept very understandable, if you read them with an *is* at the beginning and formulate them as a question.

x < 2 && x >= 0	true if x between 0 and 2, excluding 2.
x == 0 || x != 0	always true
!(x == 0 || x > 100)	true if x is smaller then 0 or if it is between 0 and 100, excluding 0.

Conditional Statements

Conditional Statements are an important construct in programming, since we always need to check some condition and let the program changing its behavior accordingly.



Conditional Statements

The simplest form of a **conditional statement** is the if statement. It checks if some condition is fulfilled and executes the code if it is *true*.

```
if (aValue < 2) {
System.out.println("Hello World!");
}
```

Conditional Statements

When you want the program to execute an alternative, if a condition is not met you can add the *else* statement.

```
 if (aValue < 2) {
System.out.println("Hello World!");
} else {
System.out.println("Goodbye World!");
}
```

Conditional Statements

It is also possible to chain conditional statements using the *else if* statement.

```
 if (aValue < 2) {
System.out.println("Hello World!");
} else if (aValue > 99) {
System.out.println("Wow!");
} else {
System.out.println("Goodbye World!");
}
```

Loops

The second important construct in programming is **loops**. This makes it possible to repeat a statement multiple times.

There are two different kinds of loops, *for* loops and *while* loops.

while Loop

A *while* loop executes the corresponding sequence of instructions until some condition is not fulfilled anymore.

```
 int a = 1;                    2
while (a < 16) {               4
a = a + a;                     8
System.out.println(a);         16
}
```

for Loop

A *for* loop allows you to repeat a task a specific number of times.

for(initialization; condition; update) {
statements
}

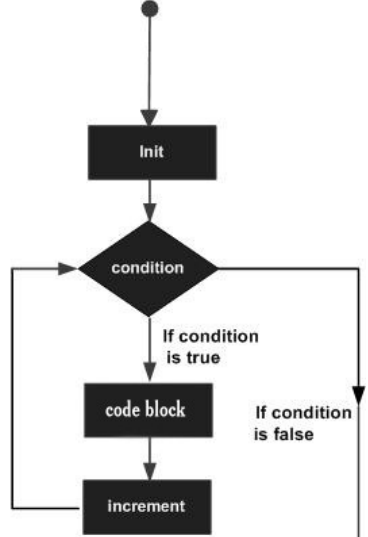


Image source: www.tutorialspoint.com

for Loop

```
for (int x = 0; x < 5; x = x + 1) {
System.out.println(x);
}
```

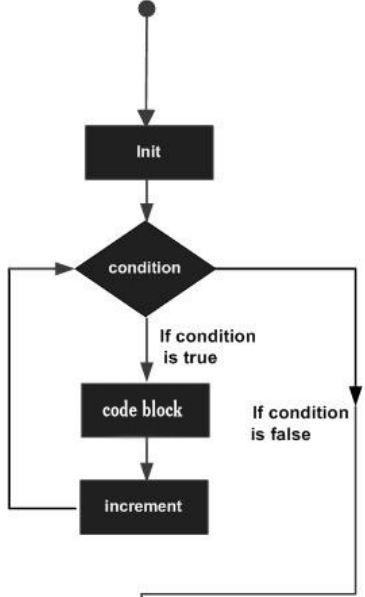


Image source: www.tutorialspoint.com

for Loop

```
for (int x = 0; x < 5; x = x + 1) {
System.out.println(x);
}
```

0
1
2
3
4

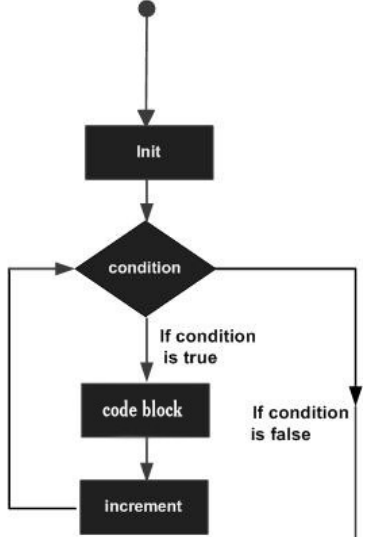


Image source: www.tutorialspoint.com

What is the difference?

```
for (int x = 0; x < 4; x = x + 1) {
System.out.println(x);
}
```

```
int x = 0;
while (x < 4) {
x = x + 1;
System.out.println(x);
}
```

What is the difference?

```
for (int x = 0; x < 4; x = x + 1) {
System.out.println(x);
}
```

```
0
1
2
3
```

```
int x = 0;
while (x < 4) {
x = x + 1;
System.out.println(x);
}
```

What is the difference?

```
for (int x = 0; x < 4; x = x + 1) {
System.out.println(x);
}
```

0
1
2
3

```
int x = 0;
while (x < 4) {
x = x + 1;
System.out.println(x);
}
```

1
2
3
4

Loop Keywords

Very useful keywords when working with loops are *break* and *continue*.

– **break** terminates the loop and resumes after the loop construct.
– **continue** terminates the current iteration and returns to the beginning of the loop.

```
for (int a = 0; a < 10; a = a + 1) {
    if (a < 3) {
        System.out.println(a * 100);
    } else if (a ==  5) {
        continue;
    } else if (a > 7) {
        break;
    } else {
        System.out.println(a);
    }
}
```

```
for (int a = 0; a < 10; a = a + 1) {
    if (a < 3) {
        System.out.println(a * 100);
    } else if (a ==  5) {
        continue;
    } else if (a > 7) {
        break;
    } else {
        System.out.println(a);
    }
}
```

```
0
100
200
3
4
6
7
```

Exercise I

In the first exercise, you will need to use conditional statements as well as loops.

Please send the code you programmed to Dani (zuend@arch.ethz.ch) by next Sunday.

This means to send the *.java* file located in "yourWorkspace → yourProjectName → src" with all the code to Dani. You can check if it the right file with every text editor.

Put all your code in the main method, the same location we put the *System.out.println("Hello World!");* last week.

Exercise I

###
00
#

information

It is not allowed to hardcode the solutions, you should print only one character per time and use loops and conditional statements!

The **modulo** operator (%) can be very useful for this exercise. It calculates the remainder of a division. This can be used, e.g., to find out very fast if a number is odd or even.

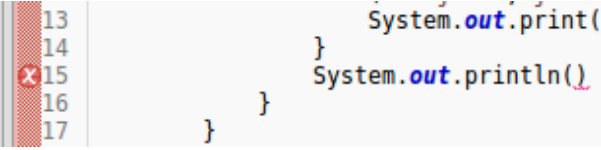
For example *7%4 == 3*; *8%2 == 0*; *8%3 == 2*

```
public static void main(String[] args) {
    for (int i = 3; i >= 0; i = i-1) {
        char symbol;
        if (i%2 == 1) {
            symbol = '#';
        } else {
            symbol = '0';
        }
        for (int j = 0; j < i; j = j + 1) {
            System.out.print(symbol);
        }
        System.out.println();
    }
}
```

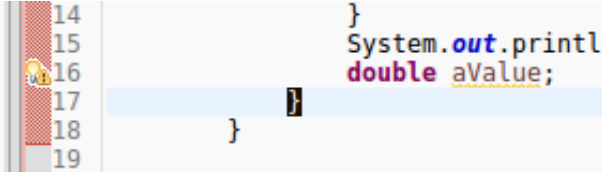
```
###
00
#
```

Editor

errors and warnings



The editor helps to find errors in your code. To find out what it is, hover with the mouse over the error symbol and it will give you a hint what could be wrong.



The same accounts for warnings, with the yellow symbol.

Other Helpers

debugging & resources

The most important helpers for larger projects are two, the **internet** and the **debugging mode**.

When looking for something specific for Java programming, search in your favourite search engine, beginning with *Java* and then your question.

When you cannot figure out what is wrong with your code, check the Internet for *Eclipse Java Debugging* to find for example this tutorial:
`http://www.vogella.com/tutorials/EclipseDebugging/article.html`