

2IP90

Lecture 2

Input/Output
Repetition
Arrays

Peach

- ④ pairs should submit only ONE program
- ④ rejected means: did not compile or execute
does NOT mean not registered
- ④ don't use packages

Three things

Syntax – Conventions – Semantics

1. Syntax: grammar
if (...)

2. Semantics: meaning
when guard is true, execute first part

3. Conventions: wise advise
indent after each {

Output

- ④ console output:
 - ④ `System.out.println("x is "+x);`
 - ④ without new line: `System.out.print("x is "+x);`
 - ④ numeric arguments are automatically converted to their textual decimal representation
- ④ dialog output: `JOptionPane.showMessageDialog(null, "boe!");`

Input

- console input: `scanner.next(); scanner.nextInt(); scanner.nextDouble();`

every next-command reads an input word and advances to the next word

- `import java.util.Scanner;`

- Declare variable: `Scanner scanner;` (or any other name)

- Create and connect object: `scanner = new Scanner(System.in);`

- `scanner.hasNext()` is true when there is input left (always true in interactive situation, not always true when reading from file (such as in Peach))

- `scanner.hasNextInt(), hasNextDouble()` etc. are true when such a number is on top of input

- dialog input: `JOptionPane.showInputDialog(null, "type wat");`

Scanner

`next()` reads next word (whitespace separated) and gives a `String`, without surrounding whitespace

`nextInt()` ditto, gives an `int`

`nextDouble()`, `next...` ditto, gives a `double` (or type ...)

`hasNext()` is true if there is still more than whitespace on input (never false when reading from keyboard, usually)

`hasNextInt()` is true if there is still a word on input and that word is an integer number

`hasNextDouble()`, `hasNext...` ditto, gives a `double` (or type ...)

- NB: what separates the words (tokens) can be defined by programmer

A Problem

- ④ Computer has a fixed set of elementary instructions
- ④ A program is a (long) sequence of these instructions



- ⦿ A fast computer can perform one instruction every nanosecond (1 GHz). Suppose a dedicated programmer writes one instruction every second. How long would it take him to program 1 second of computer time?
- ⦿ 30 years...

Solution? ... Repetition!

Loops



Repetition (while)

- ④ Aim: execute some statements more than once
- ④ Means: while-construct
- ④ Meaning: repeat loop-body while guard is true; inspection before each execution
- ④ (this is in fact the third trick)

Repetition example

```
int x = 0;
while (x<3) {
    System.out.println(x);
    x = x+1;
}
System.out.println(
    "ready!");
```

=

```
int x = 0;                                     output

if (x<3) continue else stop
    System.out.println(x);                    0
    x = x+1;

if (x<3) continue else stop
    System.out.println(x);                    1
    x = x+1;

if (x<3) continue else stop
    System.out.println(x);                    2
    x = x+1;

if (x<3) continue else stop

System.out.println(
    "ready!");                                ready!
```

Repetition - syntax

```
while ( <boolean expression> ) {           "guard"  
    <statements>                           "loop body"  
}
```

④ boolean expression as in if-construct: something with the value true or false

④ examples:

$x < 2$

$x == y$

$x > 0 \ \&\& \ x < 10$

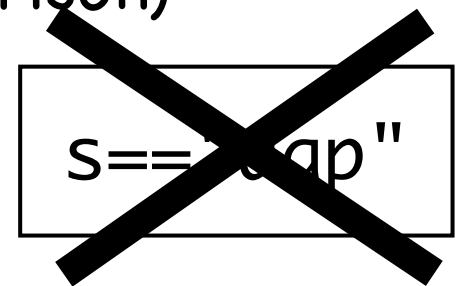
Boolean Operators

operator	description	example expression	result
<	less than	2 < 3	true
>	greater than	2 > 2	false
<=	less than or equal to	2 <= 2	true
>=	greater than or equal to	3 >= 5	false
==	equal to	1 == 1	true
!=	not equal to	1 != 1	false
&&	logical <u>and</u>	0 < 2 && 2 < 3 0 < 2 && 3 < 2	true false
	logical <u>or</u>	0 < 2 3 < 2 2 < 0 3 < 2	true false
!	negation, logical <u>not</u>	! true !(2 < 0)	false true

⦿ do not mix up = (assignment) and == (comparison)

⦿ do not use == to compare Strings
use instead:

s.equals("aap") or "aap".equals(s)



Data types

- ④ every expression and variable has a type
- ④ types must match
 - ④ equal or compatible
- ④ compatible types: “smaller” fits in “larger”
 - ④ e.g., `double x = 3;`

int

numeric types

- ④ integer types: int, long (byte, short, char)
- ④ fractional types: float, double

text types

④ char: 'a' 'b' '0'

④ String: "abc" "a" ""

"sir, I bear a rhyme excelling in mystic force and magic spelling"

String operations

- `+`: concatenation
`"good" + "bye" ➡ "goodbye"`
- `equals`: equality
`"good".equals("bad") ➡ false`
- `compareTo`: lexicographic ordering
`"bad".compareTo("good") ➡ 1`
`"good".compareTo("bad") ➡ -1`
`"equal;".compareTo("equal") ➡ 0`
- `length`: gives number of characters
`"five".length() ➡ 4`

String operations (ctd)

- `substring`: gives subsequence by position
`"Building".substring(1,4) ➡ "uil"`

first character has position 0

- `indexOf`: gives position of substring
`"Building".indexOf("uil") ➡ 1`
`"Building".indexOf("valk") ➡ -1`

codingbat.com

- ④ site with exercises
- ④ make an account
- ④ do the designated exercises

codingbat

calculates
an int

name of the
function

- ④ write a functions that calculates the square of a given number"

the
given number

```
int square(int x) {  
    return x*x;  
}
```

here comes the
calculation

- ④ calculation is $x * x$

- ④ how to say that this is the result?

return

codingbat

🌐 "write a functions that calculates the square of a given number"

```
int square(int x) {  
    return x*x;  
}
```

for-loop - syntax

```
for ( <initialisation>; <guard>; <increment> ) {  
    <statements>  
}
```

- ① <initialisation>: starting assignment to counter
- ② <increment>: statement that "increases" counter
($i=i+1$, $j=j-1$, $k=k+x$, etc.)
- ③ <guard>: boolean expression as in while

do-while loop

- ③ when you have to do something at least once

```
double balance;  
String choice;  
// init balance  
do {  
    System.out.println("Type amount for deposit.");  
    deposit = sc.nextDouble();  
    balance = balance + deposit;  
    System.out.println("New balance is " + balance + ".");  
  
    System.out.println(  
        "Do you want to make a deposit? Type y/n");  
    choice = sc.next();  
} while (choice.equals("y"));
```

do-while loop

- ⌚ when you have to do something at least once

```
int som = 0;
int cijfer = 0;

do {
    som = som + cijfer;
    System.out.println("type cijfer, -1 om te stoppen");
    cijfer = scanner.nextInt();
} while (cijfer > 0);
```


for-loop

- ⊗ when you know the number of iterations (number of times you execute the loop-body) on beforehand
- ⊗ when you want to do something for each member of a set (range) of values

```
for (int i = 0; i<100; i=i+1) {  
    System.out.println(  
        "Ik mag de accolades om loop-body's niet vergeten"  
    );  
}
```

Control and data

- Two fundamental concepts of programming are control and data. Data are values stored in variables. Information is data with meaning. Control determines what you do with the data.
- A program executes the statements (lines or parts of it) one after the other. The order (called control) is determined by:
 1. the order in the text
 2. control structuring
- We have seen some ways to structure control: put statements after each other, make while-loops, make if-structures
- Can we structure data too?

Remarks

④ `i++` and `i+=1` are shorthand for `i=i+1`

idea of `+=` works with many operators, such as `*=`, `-=`, and `%=`

④ don't mix them! `i=i++` is wrong!

④ It is in some cases allowed to write the body of a loop or the branch of an if without braces (`{ }`).

Don't do this. It easily leads to errors.

Simple structuring of data: arrays

- An array is a sequence of numbered variables

- Instead of

```
int student1;  
int student2;  
int student3;  
student1 = 7;  
student2 = 5;  
student3 = 3;
```

- you can do:

```
int[] students;  
students = new int[3];  
students[0] = 7;  
students[1] = 5;  
students[2] = 3;
```

declaration
creation: gives 3 elements
last one is students[2]

Arrays

- ④ Number between [] is called index
- ④ In Java, all arrays start with 0. Last index is number of elements - 1
- ④ Declaration and creation are separate; often, you don't know directly how many variables you need, e.g., asked as input from the user
- ④ declaration and creation can be combined, though:
`int[] students = new int[3];`

Arrays - example

```
int[] students = new int[3];
students[0] = 7;
students[1] = 5;
students[2] = 6;

// bereken som
double som = 0.0;
for (int i=0; i<3; i=i+1) {
    som = som + students[i];
}
double gemiddelde = som/3;
```

Flexibility - example

Using the number 3 is inflexible

Make the program work for any number of students

```
System.out.println("Geef aantal studenten: ");  
int n = scanner.nextInt();
```

```
int[] students = new int[n];  
for (int i=0; i<n; i=i+1) {  
    students[i] = scanner.nextInt();  
}
```

```
students[0] = 7; students[1] = 5; students[2] = 6;  
// bereken som  
double som = 0.0;  
for (int i=0; i<n; i=i+1) {  
    som = som + students[i];  
}  
double gemiddelde = som/n;
```



beware: division does not work
when som is an int

using length of arrays

- following program works for students array of any length not depend on

```
System.out.println("De volgende studenten zitten onder het gemiddelde:");
for (int i=0; i<students.length; i=i+1) {
    if (students[i] < gemiddelde) {
        System.out.println(i);
    }
}
```

- Advise: use .length whenever possible

Improvement - example

students.length gives you the number of elements in the array
works safer than the variable n

Note: trying to access an array element that doesn't exist, gives nasty errors

```
System.out.println("Geef aantal studenten: ");
int n = scanner.nextInt();

int[] students = new int[n];
for (int i=0; i<n; i=i+1) {
    students[i] = scanner.nextInt();
}

students[0] = 7; students[1] = 5; students[2] = 6;
// bereken som
double som = 0.0;
for (int i=0; i< students.length; i=i+1) {
    som = som + students[i];
}
double gemiddelde = som/students.length;
```

This works for any number of students; only the input has to be adapted

The devious division

- ⊗ som/3 doesn't work when som is an int
Why?

There are two versions of /

1. when one of the arguments is a double:
ordinary division
2. when both arguments are int:
integer division

$$\text{e.g., } 7/2 = 3$$

- ⊗ integer division can be handy in programming;
it is confusing to have one symbol for two
operations