2IP65-2IP70

Lecture 2

Repetition

Arrays

Let op

- zalen 2IP70 gewijzigd
- ga in de goede cluster zitten (zit je in "loslopers" of in "wiskunde" , maar doet TIW)
- register with Studyweb (mainly for informing you)

Three things

Syntax - Conventions - Semantics

1. Syntax: if (...)
2. Conventions: indent after each {
3. Semantics: when guard is true, execute first part

Input & Output

- console output: System.out.println("x is "+x);
- 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);
- dialog output: JOptionPane.showMessageDialog(null, "boe!");
- dialog input: JOptionPane.showInputDialog(null, "type wat");

- A fast computer can perform one instruction every nanosecond (1 GHz). Suppose an incredibly fast programmer writes one instruction every second.

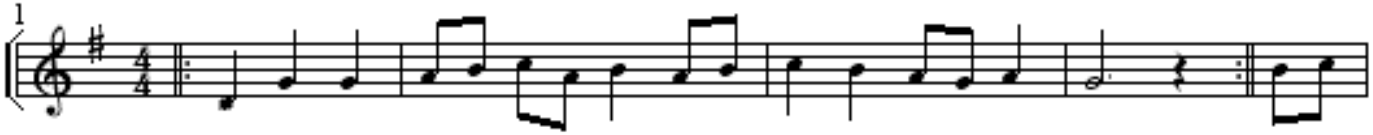
- It will take him 30 years to program one second of computer time...

Solution? ... Repetition!

Repetition (while)

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

Loops



Repetition example

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

=

```
int x = 0;

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

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

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

if (x<3) continue else stop

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

output

```
0
1
2
ready!
```

Repetition - syntax

```
while ( <boolean expression> ) {          "guard"

   <statements>                           "loop body"

}
```

- boolean expression as in if: something with the value true or false
- examples:
  x < 2
  x == y
  x > 0 && x < 10

Boolean Operators

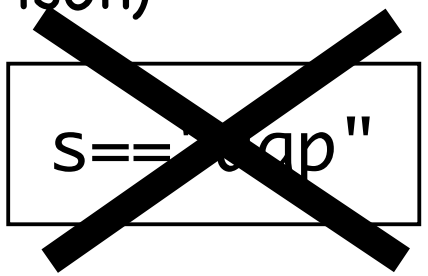
operator	description	example expression	result
< >	less than greater than	2 < 3 2 > 2	true false
<= >=	less than or equal to greater than or equal to	2 <= 2 3 >= 5	true false
== !=	equal to not equal to	1 == 1 1 != 1	true false
&&	logical and	0 < 2 && 2 < 3 0 < 2 && 3 < 2	true false
||	logical or	0 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



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"
  );
}
```

for-loop - syntax

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

      <statements>

}
```

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

do-while loop

- 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);
```

Remarks

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

- i++ and i+=1 are shorthand for i=i+1
- don't mix them! i=i++ is wrong!
- 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.

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?

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

- Number between [] is called index
- In Java, all arrays start with 0. Last index is number of elements - 1
- Declaration and creation are separate; often, you don't know directly how many variables you need, e.g., asked as input from the user
- 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 studensts; 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

   e.g., 7/2 = 3

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