

# Lecture 3

Methods

# Lecture 3

## Methods

---

### Control structuring – methods

- A method is a sequence of statements with a name
- **Calling** (aka invoking) the name leads to execution of the sequence of statements
- After execution, control **returns** to the place after the call

# Defining and using a method

```
Define a class    class Publiek {  
define the name  void scheld() {  
of the method   System.out.println("Hij is een hondelul");  
define the body }  
  
                void spreekkoor() {  
call the method scheld();  
                again scheld();  
and again      scheld();  
                }  
  
                public static void main(String..  
new Publiek().spreekkoor();  
                }  
}
```

output

```
Hij is een hondelul  
Hij is een hondelul  
Hij is een hondelul
```

# method - syntax

## 1. definition (declaration):

```
void <name> () {  
    <statements>           "method body"  
}
```

## 2. use (call):

```
<name>();
```

- "void" is **only** used at the definition
- will be explained later

# Why use methods?

- saves time with recurring tasks
- makes your program more readable
- gives a better organization of your program:
  - subdivision of larger tasks
  - group tasks that belong together
- makes your program more flexible and reusable

# Recurring tasks



```
class Emoticon {  
    void smile() {  
        System.out.println( ":-)" );  
    }  
}
```

```
void sad() {  
    System.out.println( ":-(" );  
}
```



```
void angry() {  
    System.out.println( ">:-|" );  
}
```

```
void message() {  
    System.out.println("Hi, Plien");  
    smile();  
    System.out.println("Gisteren zag ik Annechien");  
    angry();  
    angry();  
    angry();  
}  
}
```

Output:

Hi, Plien

:-)

Gisteren zag ik Annechien

>:-|

>:-|

>:-|

# Reusability: parameters

```
class Emoticon {  
    void smile() {  
        System.out.println( ":-)" );  
    }  
}
```

```
void angry(int n) {  
    for (int i=0; i<n; i++) {  
        System.out.println( ">:-|" );  
    }  
}
```

```
void demoticon {  
    System.out.println("Hi, Plien");  
    smile();  
    System.out.println("Gisteren zag ik Annechien");  
    angry(3);  
}  
}
```

Output:

Hi, Plien

:-)

Gisteren zag ik Annechien

>:-|

>:-|

>:-|

[www.milenniumbattle.nl](http://www.milenniumbattle.nl)



# Reusability: parameters

```
class Blah {  
    void angry( int n ) {  
        for (int i=0; i<n; i++) {  
            System.out.println( ">:-|" );  
        }  
    }  
  
    void happy( int h1 ) {  
        System.out.println("Happiness level "+h1);  
    }  
  
    void printSquare( int n ) {  
        System.out.println( n*n );  
    }  
}
```

parameter declaration works as a variable in the body

working area (scope) of n

scope of h1

scope of another n

# special methods – functions

- known from mathematics: square root, formulas, functions

- math:  $\text{squareIt}(w) = w^2$

parameter

```
class Example {  
    double squareIt(double w) {  
        return w*w;  
    }  
}
```

return type

return value

- examples of use:

```
a = squareIt(10);
```

```
b = squareIt(5) + squareIt(q/2);
```

```
System.out.println("kwadraat: "+squareIt(x));
```

# Local variables

- a variable that is only relevant for a single method execution can be declared inside the method body:

```
void elections() {  
    String answer;  
    System.out.println("Who won?");  
    answer = Scanner.next();  
    if (answer.equals("AZ") {  
        System.out.println("pheww");  
    }  
}
```

- variable answer is only needed for elections
- answer is not available outside elections
- value is discarded when call to elections terminates (returns)

# Variables

```
class Blah {  
    String madsmile ;
```

instance variable

```
void angry( int n ) {  
    for (int i = 0; i < n; i++) {  
        System.out.println( madsmile );  
    }  
}
```

parameter

local variable

```
void doIt() {  
    madsmile = ">:-|";  
    angry( 3 );  
}
```

# three kinds of variables

- instance variable:

- scope is all methods in class
- automatic initialization with 0 or false (booleans)

- parameter:

- scope is one method only
- initialization by call

- local variable:

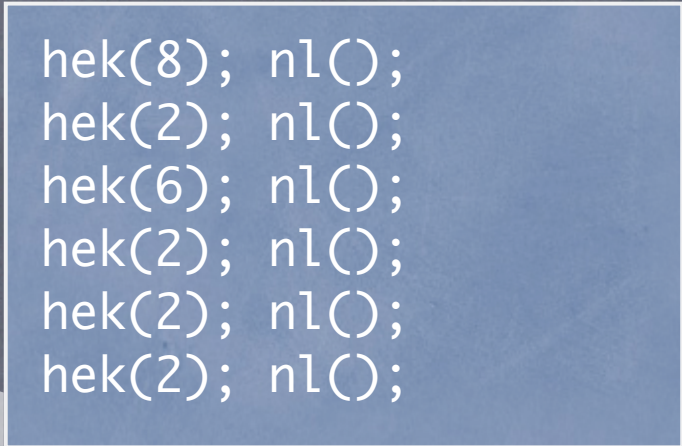
- scope is one method or less
- initialization required

# What does the box do?

```
class Printer {  
    void hek(int n) {  
        String s = "";  
        for (int i=0; i<n; i++) {  
            s = s+"#";  
        }  
        System.out.println(s);  
    }  
  
    void box() {  
        hek(1);  
        hek(2);  
        hek(3);  
        hek(4);  
    }  
  
    public static void main(String[] a) { new Printer().box(); }  
}
```

# What does the box do?

```
class Printer {  
    void hek(int n) {  
        String s = "";  
        for (int i=0; i<n; i++) {  
            s = s+"#";  
        }  
        System.out.print(s);  
    }  
  
    void nl() {  
        System.out.println();  
    }  
  
    void box() { }  
}
```



```
hek(8); nl();  
hek(2); nl();  
hek(6); nl();  
hek(2); nl();  
hek(2); nl();  
hek(2); nl();
```



# Method types

## I. simple

```
void <name>() {  
    <body>  
}
```

## II. parameters

```
void <name>(<type> <var>, ..., <type> <var>) {  
    <body>  
}
```

## III. return

```
<type> <name>() {  
    <body>  
    return <val>;  
}
```

## IV. combination of II and III



# Method examples

## I. simple

```
void hello() {  
    System.out.println("Hellooooo!");  
}
```

## II. parameters

```
void repeat(String s, int n) {  
    for (int i=0;i<n;i++){ System.out.println(s); }  
}
```

## III. return

```
double pi() {  
    return 3.14159265359;  
}
```

## IV. combination of II and III

```
int max (int a, int b) {  
    if (a>b){return a;}else{return b;}  
}
```

# Scope

- scope is the working area of a variable or parameter
- it begins at the declaration and ends at the closing brace
- outside its scope, a variable cannot be referred to
- Example:

```
class Zuip {  
    void drunk() {  
        if (n > 10) { // error!  
            ...  
        }  
    }  
  
    void drink(int n) { ... }  
}
```

- scope of an **instance variable** is the **whole class** (except the static methods)

## Scope 2a

- many variables and parameters with the same name may occur in different parts of your program, without conflict
- Example:

```
void angry(int n) {  
    for (int i=0; i<n; i++) {  
        System.out.println(">:-|");  
    }  
}  
...  
void halve(int n) {  
    double i = n/2.0;  
    System.out.println("heft is "+i);  
}
```

different variables i



## Scope 2b

- many variables and parameters with the same name may occur, without conflict
- Example:

```
void angry(int n) {  
    for (int i=0; i<n; i++) {  
        System.out.println(">:-|");  
    }  
}  
...  
void halve(int n) {  
    double i = n/2.0;  
    System.out.println("heft is "+i);  
}
```

different variables n



## Scope 3

```
class ScopeExample {
```

```
    int n;
```

```
    void blah( int p ) {
```

```
        System.out.println( p );
```

```
        int q = 5;
```

```
        n = q * p;
```

```
        // etc.
```

```
    }
```

```
    void rhubarb( ) {
```

```
        String n = "twee";
```

```
        System.out.println( n + n );
```

```
        // etc.
```

```
    }
```

```
    void plauder( ) {
```

```
        System.out.println( n*n );
```

```
    }
```

```
}
```

n

p

q

n

# Notes

- changes in local variables are lost as soon as method returns
- use instance variables for “permanent” changes
- do not change parameters

# More notes

- group code that belongs together in method
- break long methods up ( > ±15 lines )
- try to find methods with simple interface:
  - few and clear parameters
  - few and clear uses/changes of instance variables

# Notes on functions

- functions are methods with return value
- reserve functions for “mathematical” calculations
- preferably not in functions:
  - change of instance variables
  - input or output