Algorithmic Thinking
and
Structured Programming
(in Greenfoot)

Teachers:

Renske Smetsers-Weeda

Sjaak Smetsers

Ana Tanase

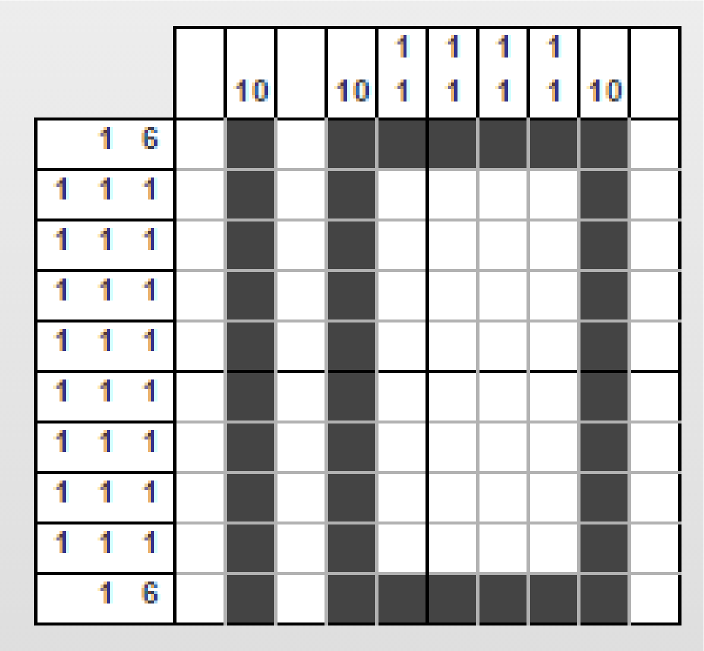
Today’s Lesson plan (8)

- Quiz
- Retrospective
  - Previous lesson
  - Task

- Blocks of theory and exercises/unplugged:

Nonogram puzzels

- Generic algorithm?
- How to represent the solution (for storing the picture)?
- Transfer: ideas for any real-world applications?

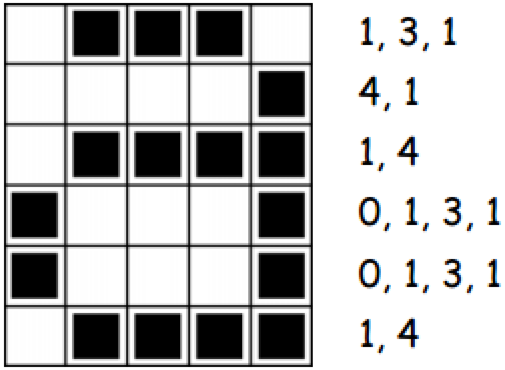


Real-world applications

- How does a fax work?
- Which computer applications store pictures?
- How can computers store pictures if they can only use digits 0 and 1 (=bits)?

Run length coding (Nonogram variation)

- Pixels (**PIC**ture **EL**ements)



1, 3, 1

4, 1

1, 4

0, 1, 3, 1

0, 1, 3, 1

1, 4

- Representation:
  - First number: white
  - Second number: black
  - Third number: white
  - ….

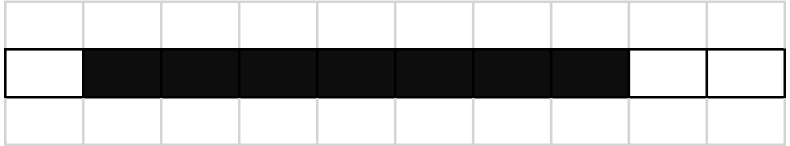
Run Length Coding

Decimal	Binary
1 2	1 01
5 7	101 111

- Representation as binary number

- More efficient way to represent?
  - choose an optimal number of bits
  - For example: max 3 bits (=seven white/black)

So 1 white, 7 black, 2 white



would be: 001 111 010

which is 1 shorter than: 0111111100

Just 1 bit,..but for big pics this does make a difference!

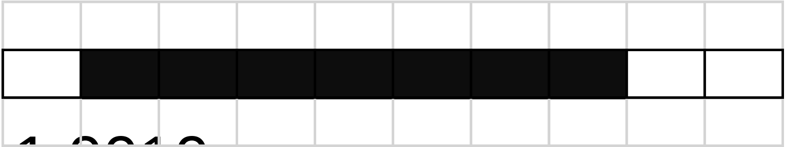
So, more bits to represent a chunk => smaller overall pic size

Run Length Coding

Decimal	Binary
1 2	1 01
5 7 12	101 111 1100

- But then (if you choose for 3 bits), how would you represent a run of 12 black pixels?
  - Use 4 bits so (12 binary): 1100
  - Which is definately shorter than: 0 111111111111

  - But what happens to:



would become: 0001 0111 0010

which is 2bits **longer** than: 0111111100 !!!!!!!

Works well when large parts of picture are completely black or completely white

Run Length Coding

So, your algorithm can result in

- Awesome compression

OR

- Horrible expansion

This depends on:

- Representation: choice of #bits to store
- Data: bit-lengths in the picture

Compression Algorithms

Fax machine: Compression algorithm: run length coding

- When:
  - Image characteristics:
    - Large blocks of white (margins)
    - Large blacks of black (horizontal line)
- Why:
  - Save space
  - transmission time / bandwith
  - Easy bit-parity (error checking and correcting)

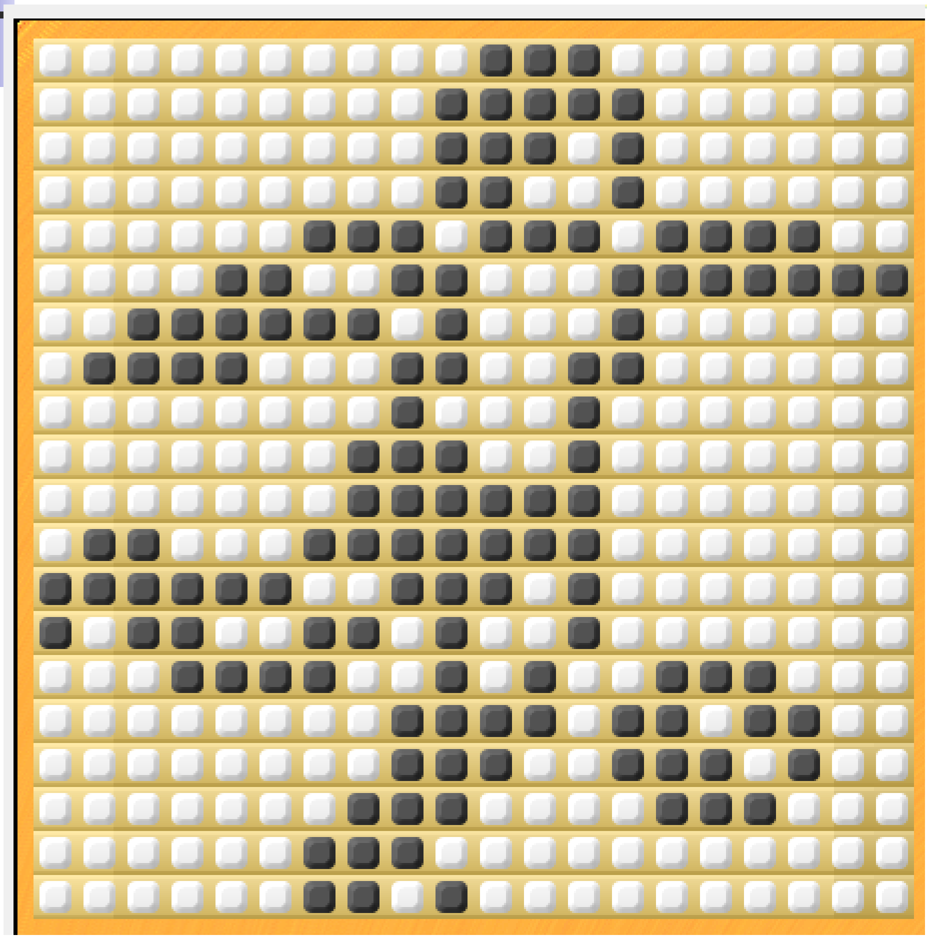
Compression Algorithms

Many more different compression techniques:

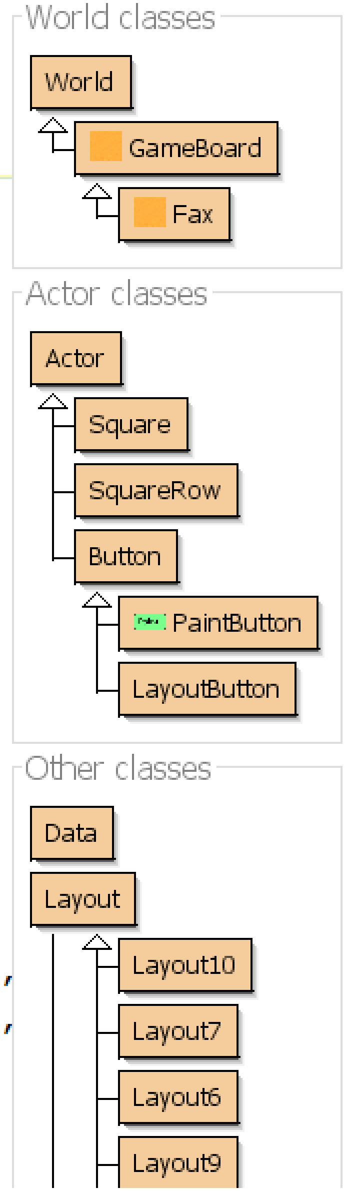
- Photographs / Pictures (JPG)
- Music (MP3)
- Text (ZIP)

- Lossy: compression losing some info (eg. MP3)
- Lossless: compression allowing full recovery (eg. ZIP)

Greenfoot fax

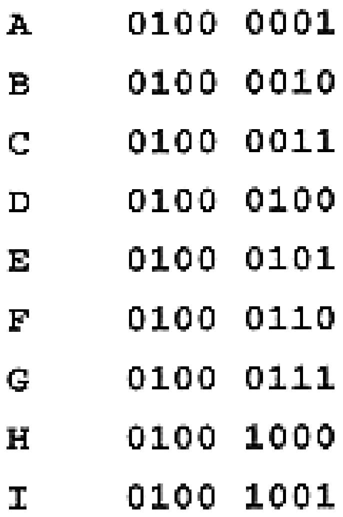


```
super( new int[][] {
        { 10, 3 },
        { 9, 5 },
        { 9, 3, 1, 1 },
        { 9, 2, 2, 1 },
        { 6, 3, 1, 3, 1, 4 },

        { 4, 2, 2, 2, 3, 7 },
        { 2, 6, 1, 1, 3, 1 },
        { 1, 4, 3, 2, 2, 2 },
        { 8, 1, 3, 1 },
        { 7, 3, 2, 1 },

        { 7, 6 },
        { 1, 2, 3, 7 },
        { 0, 6, 2, 3, 1, 1 },
        { 0, 1, 1, 2, 2, 2, 1,
        { 3, 4, 2, 1, 1, 1, 2,

        { 8, 4, 1, 2, 1, 2 },
        { 8, 3, 2, 3, 1, 1 },
        { 7, 3, 4, 3 },
        { 6, 3 },
        { 6, 2, 1, 1 }
} );
```



Text compression

- Characters encoded using ASCII: 8 bits per char



A	0100 0001
B	0100 0010
C	0100 0011
D	0100 0100
E	0100 0101
F	0100 0110
G	0100 0111
H	0100 1000
I	0100 1001

- Text often has repeating letters
- ‘EFFICIENCY’ 10 chars

  10* 8 = 80 bits for representation

- Compression idea:
  - Highest freq letters ⇨ shortest representation
  - Lowest freq letters ⇨ longer representation

Hufmann Coding

Coding table:

E	I	F	C	N	Y
00	01	10	110	1100	1101

Unique coding for ‘EFFICIENCY’:

00 10 10 01 110 01 00 1100 110 1101

26 bits (vs. 80 with ASCII) ⇨ 80% reduction!!

Of course you must still send decoding table

But definately viable for large texts!

Topics for assignment 6

- Constructors, instance variables
- Access modifiers: private, public (protected): information hiding
- Getter/setter methods

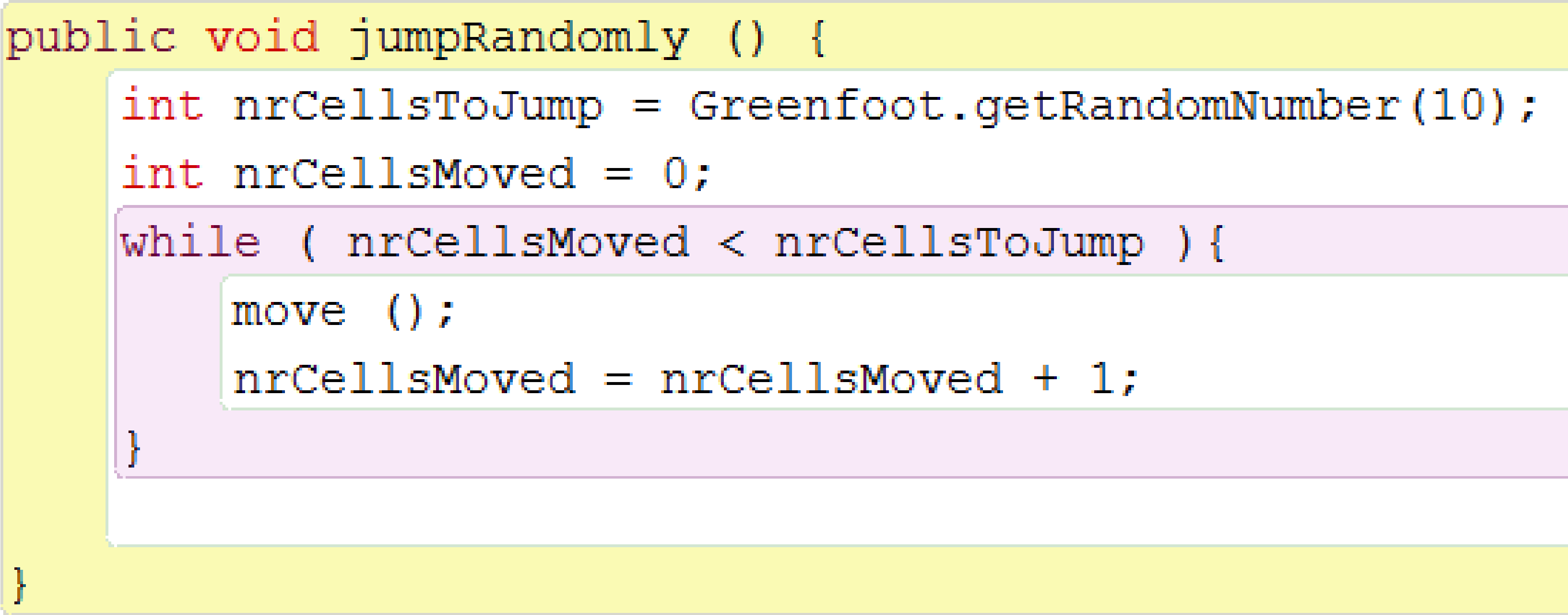
Instance variables vs. Local variables

Demo

- Create an object using new (drag)
- Explain effect on method variables

Variable Scope (lifetime)

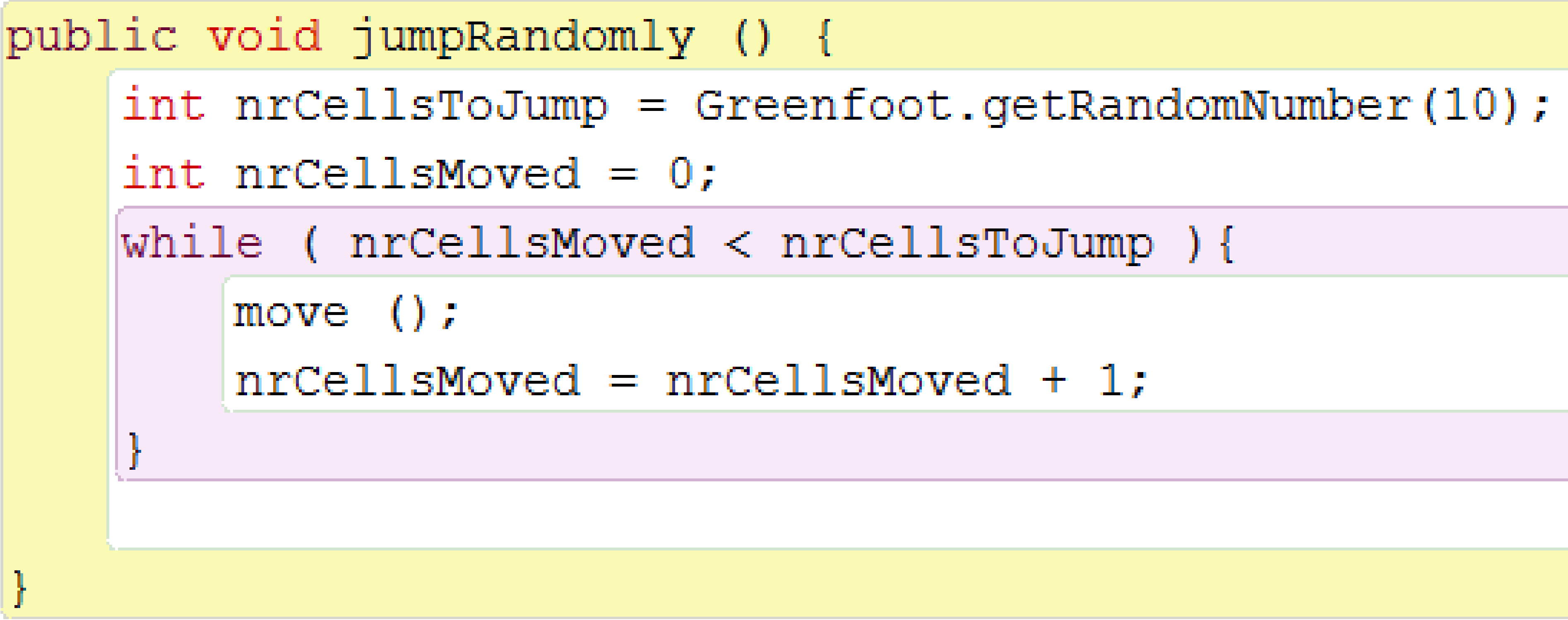
- What happens to variable **nrCellsMoved** after this method?



```
public void jumpRandomly () {
    int nrCellsToJump = Greenfoot.getRandomNumber(10);
    int nrCellsMoved = 0;
    while ( nrCellsMoved < nrCellsToJump ){
        move ();
        nrCellsMoved = nrCellsMoved + 1;
    }

}
```

Variable Scope (lifetime)

- After the method, **nrCellsMoved** is destroyed!
- So we can’t use **nrCellsMoved** in another method….



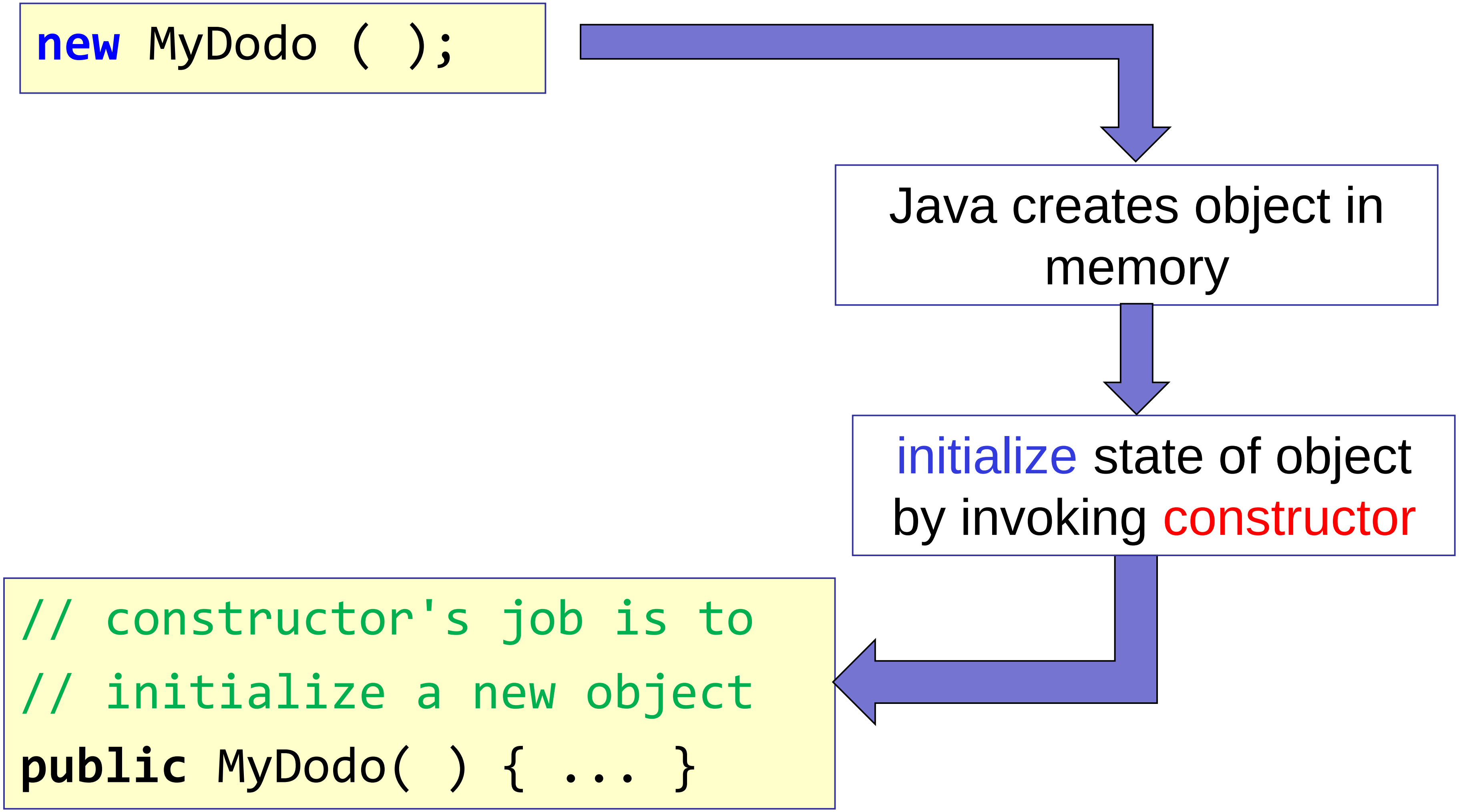
```
public void jumpRandomly () {
    int nrCellsToJump = Greenfoot.getRandomNumber(10);
    int nrCellsMoved = 0;
    while ( nrCellsMoved < nrCellsToJump ){
        move ();
        nrCellsMoved = nrCellsMoved + 1;
    }

}
```

- Unless, we use **instance variables.**

Instance variables

- To store (remember) values for longer periods of time
  - Outside of method:
    - ‘normal’ method variables loose their values
    - Use instance variables when using same variable by two different methods
  - When act is called again:
    - Only instance variables are stored
    - All other values are lost
  - You can even ‘inspect’ object value at all times

How Objects are Created



```
new MyDodo ( );
```

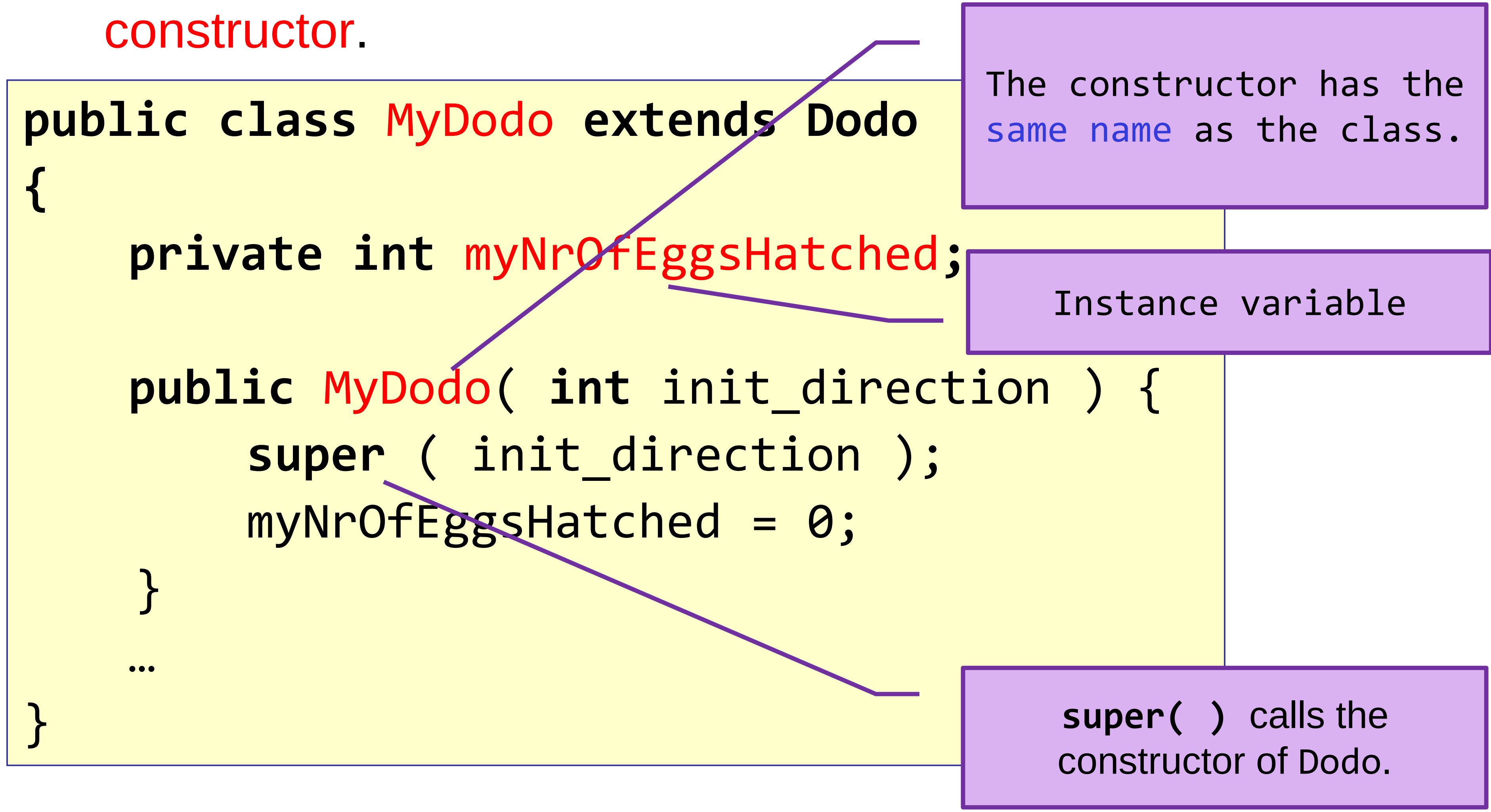
Java creates object in memory

initialize state of object by invoking constructor

```
// constructor's job is to
// initialize a new object
public MyDodo( ) { ... }
```

The Constructor

- When Java creates a new object, it calls the class's constructor.



```
public class MyDodo extends Dodo
{
    private int myNrOfEggsHatched;

    public MyDodo( int init_direction ) {
        super ( init_direction );
        myNrOfEggsHatched = 0;
    }
    …
}
```

The constructor has the same name as the class.

Instance variable

super( ) calls the constructor of Dodo.

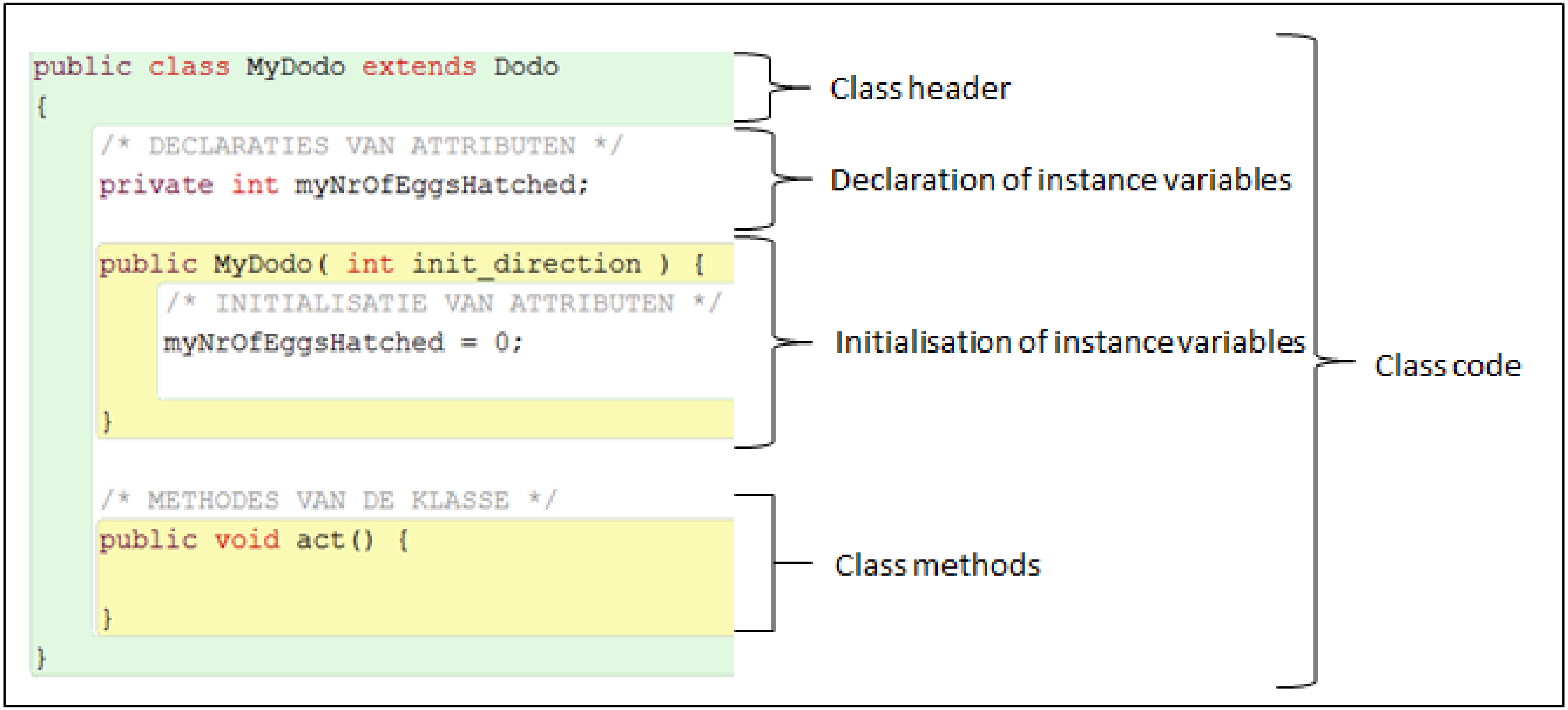
Constructor (2)

- The purpose of a Constructor is to initialize the state of a new object... Prepare the object to start work.

- A class may have several constructors, ONLY ONE is called, and object prepared accordingly.

```
public class MyDodo extends Dodo
{
   private int myNrOfEggsHatched;

   public MyDodo() {
      super ( EAST );
      myNrOfEggsHatched = 0;
   }

   public MyDodo( int init_direction ) {
      super ( init_direction );
      myNrOfEggsHatched = 0;
   }
```

Class code



```
public class MyDodo extends Dodo
{
    /* DECLARATIES VAN ATTRIBUTEN */
    private int myNrOfEggsHatched;

    public MyDodo( int init_direction ) {
        /* INITIALISATIE VAN ATTRIBUTEN */
        myNrOfEggsHatched = 0;

    }

    /* METHODES VAN DE KLASSE */
    public void act() {

    }
}
```

Class header

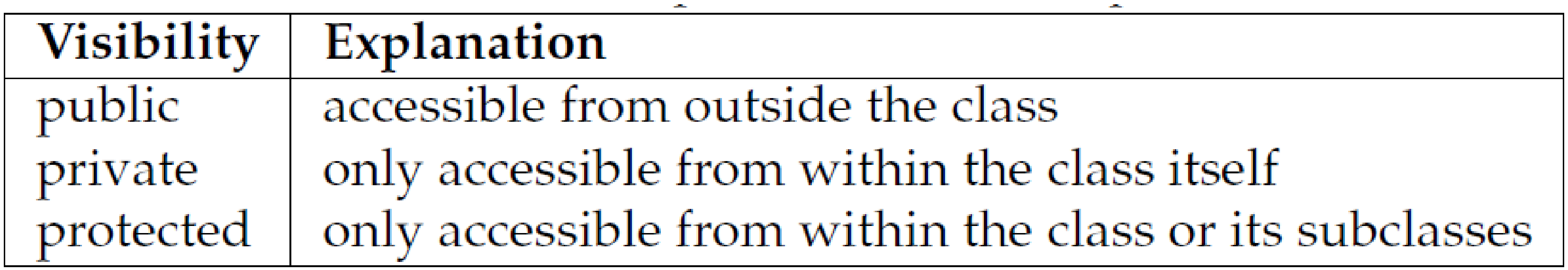
Declaration of instance variables

Initialisation of instance variables

Class methods

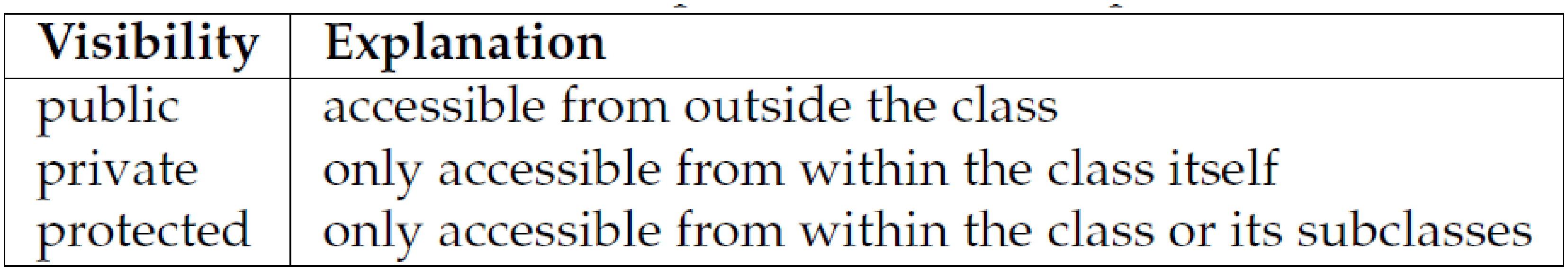
Class code

Visibility of variables / methods



Visibility	Explanation
public	accessible from outside the class
private	only accessible from within the class itself
protected	only accessible from within the class or its subclasses

Getter method



Visibility	Explanation
public	accessible from outside the class
private	only accessible from within the class itself
protected	only accessible from within the class or its subclasses

int myAge is private, no one needs to know… so…

```
private int myAge;
```

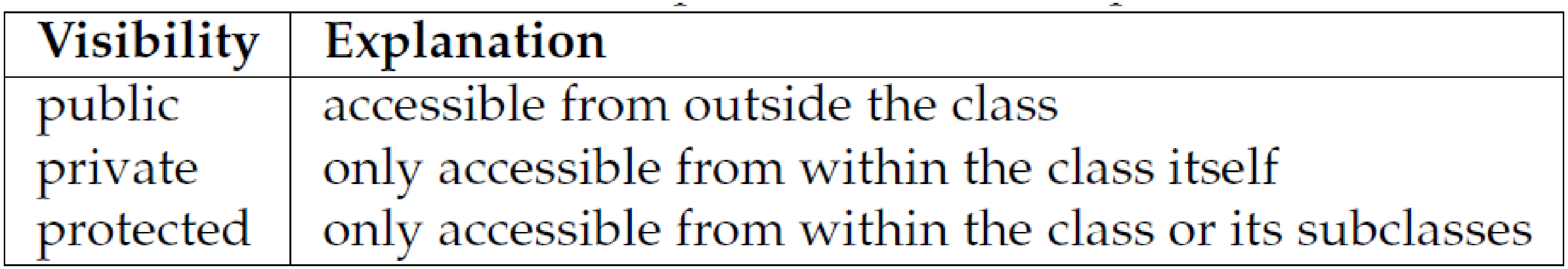
But… if myAge needs to **asked** for a (real) reason:

```
public int getMyAge( ) {
     if ( youHavePermissionToKnow ( )  ){
          return myAge( ) ;
      } else {
            return 0;
   }
}
```

To call (object Teacher) from another method, use:

```
Teacher.getMyAge()
```

Setter method



Visibility	Explanation
public	accessible from outside the class
private	only accessible from within the class itself
protected	only accessible from within the class or its subclasses

String myPassword is private, so:

```
private string myPassword;
```

But… if myPassword needs to be **changed** for a (real) reason:

```
public void setMyPassword ( string newPassword ) {
        myPassword = newPassword;
}
```

How to call (object Teacher) from another method, call:

```
Teacher.setMyPassword ( “doorbell” );
```

Questions?

Wrapping up

Homework for Wednesday 8:30 Feb 17th:

- Assignment 6:
  - **FINISH assignment 6 up to and incl 5.3**

    (you may advance if you wish

    -> less homework next time)
  - ZIP code and ‘IN’ and **email** to **Renske.weeda@gmail.com**