Algorithmic Thinking and Structured Programming (in Greenfoot)

Teachers:

Renske Smetsers-Weeda

Sjaak Smetsers

Today’s Lesson plan (8)

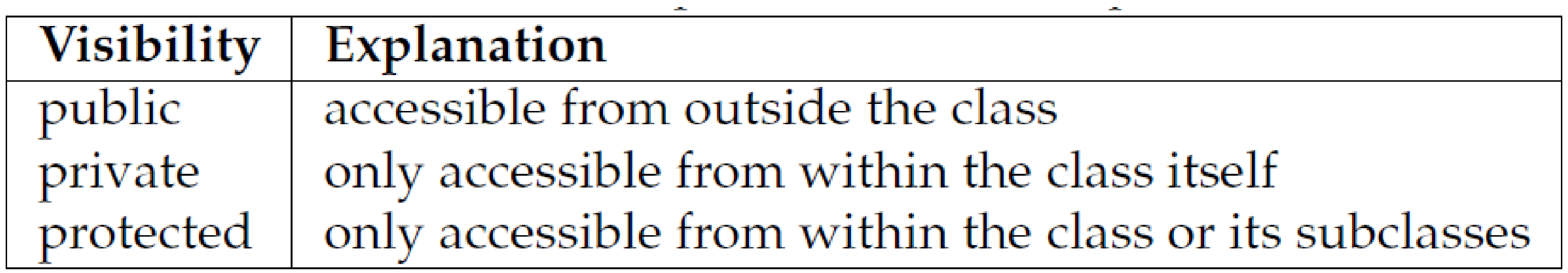
- Retrospective
  - Previous lesson
  - Discuss Quiz and Task
- Exercises

Retrospective

- Constructors, instance variables

Information hiding

- Rule: make instance variables **private**



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

- This means: other objects can’t reach it!
- Solution: create (if really needed)
  - **public getter** method
  - **public setter** method

Setter and getter methods (examples)

```
public void setOneEggLessToHatch( ) {
    myEggsToHatch--; // decrease value by one
}
```

```
public int getNrOfEggsHatched( ) {
    return myNrOfEggsHatched( );
}
```

Calling a method from another class

Example:

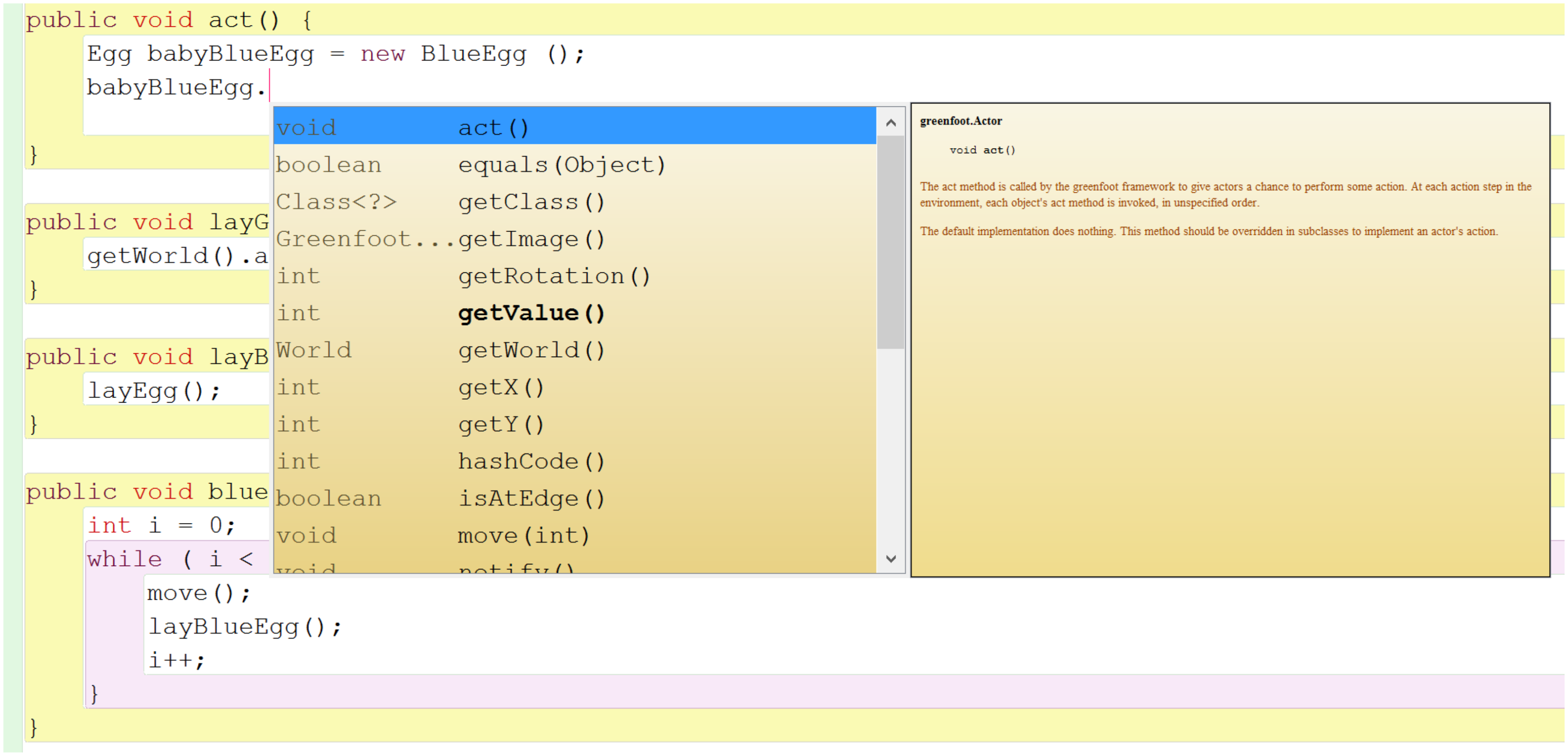
Chicken object called `clara` with method:

```
public void setOneEggLessToHatch( ) {
    myEggsToHatch--; // decrease value by one
}
```

then Farmer can call:

```
clara.setOneEggLessToHatch ( ) ;
```

Tip: type ‘.’ and then <Ctrl>+<Space>



```
public void act() {
    Egg babyBlueEgg = new BlueEgg ();
    babyBlueEgg.

}

public void layG
    getWorld().a
}

public void layB
    layEgg();
}

public void blue
    int i = 0;
    while ( i <
        move();
        layBlueEgg();
        i++;
    }
}
```

```
void          act()
boolean       equals(Object)
Class<?>      getClass()
Greenfoot...getImage()
int           getRotation()
int           getValue()
World         getWorld()
int           getX()
int           getY()
int           hashCode()
boolean       isAtEdge()
void          move(int)
void          notify()
```

greenfoot.Actor

`void act()`

The act method is called by the greenfoot framework to give actors a chance to perform some action. At each action step in the environment, each object's act method is invoked, in unspecified order.

The default implementation does nothing. This method should be overridden in subclasses to implement an actor's action.

Steps for using instance variables

1. **Declare** instance variable in top of **class**:

```
private int myNrOfEggs;
```

2. **Initialize** (set intial value) in **constructor**:

```
myNrOfEggs = 10;
```

3. Write **public getter accessor** method

```
public int getNrOfEggs (){
    return myNrOfEggs;
}
```

4. Write **public setter mutator** method:

```
public void setNrOfEggs( int newNrEggs ){
    myNrOfEggs = newNrEggs;
}
```

instance variables: life-long memory

- Now that you know how to use instance variables
- You can write complex algorithms
- Dodo has life-long memory!

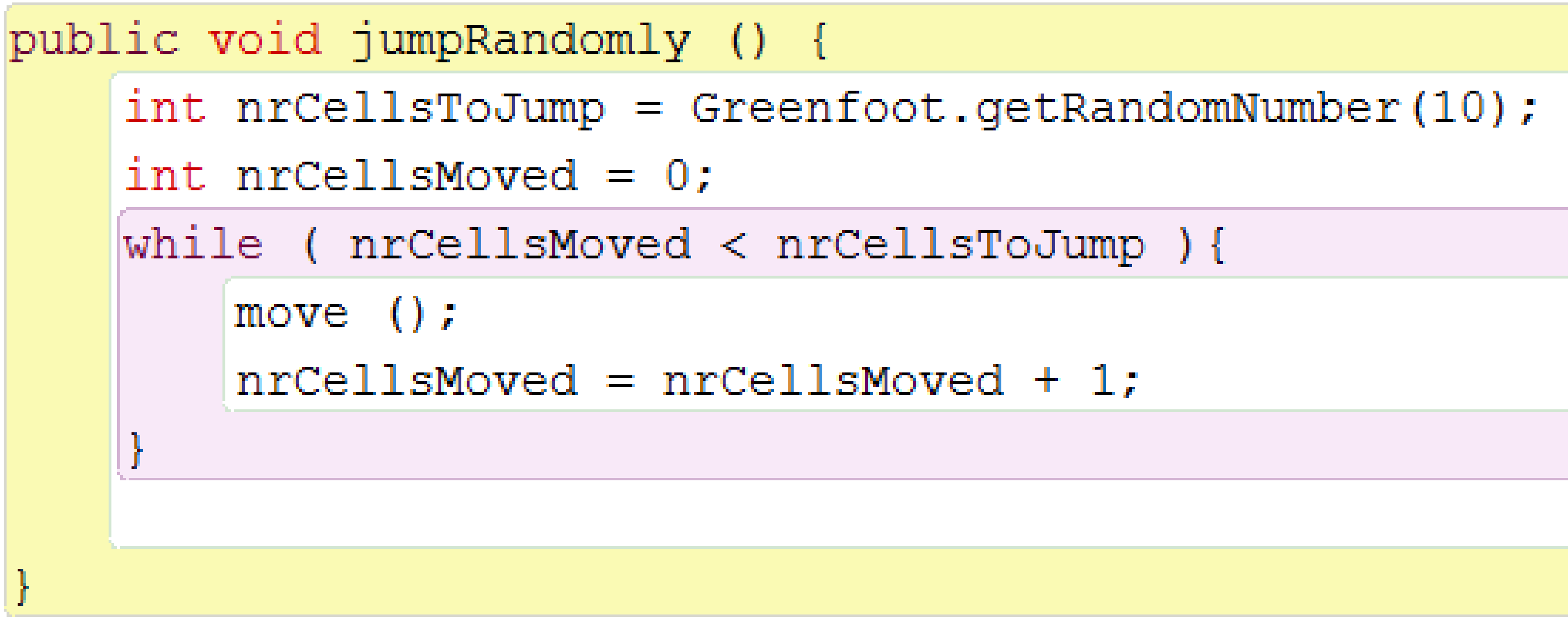
- How:
  - NO **while** in the **act ()**
  - Transform methods used in act() from ‘while’ into ‘if’
  - Use **instance variables** instead of local variables

    local variables: variables in (sub)methods

(last exercises in assignment 6)

Variable Scope (lifetime)

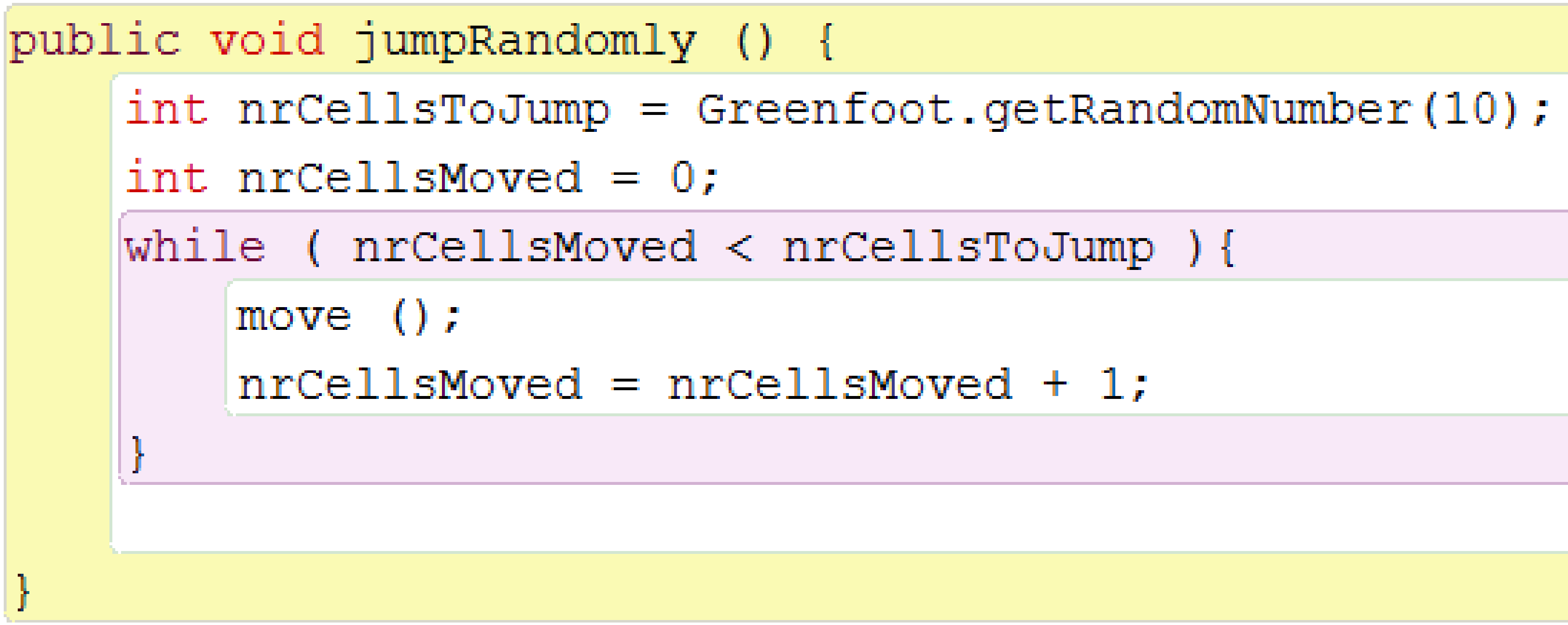
- What happens to variable **nrCellsMoved** after this method?



```
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.

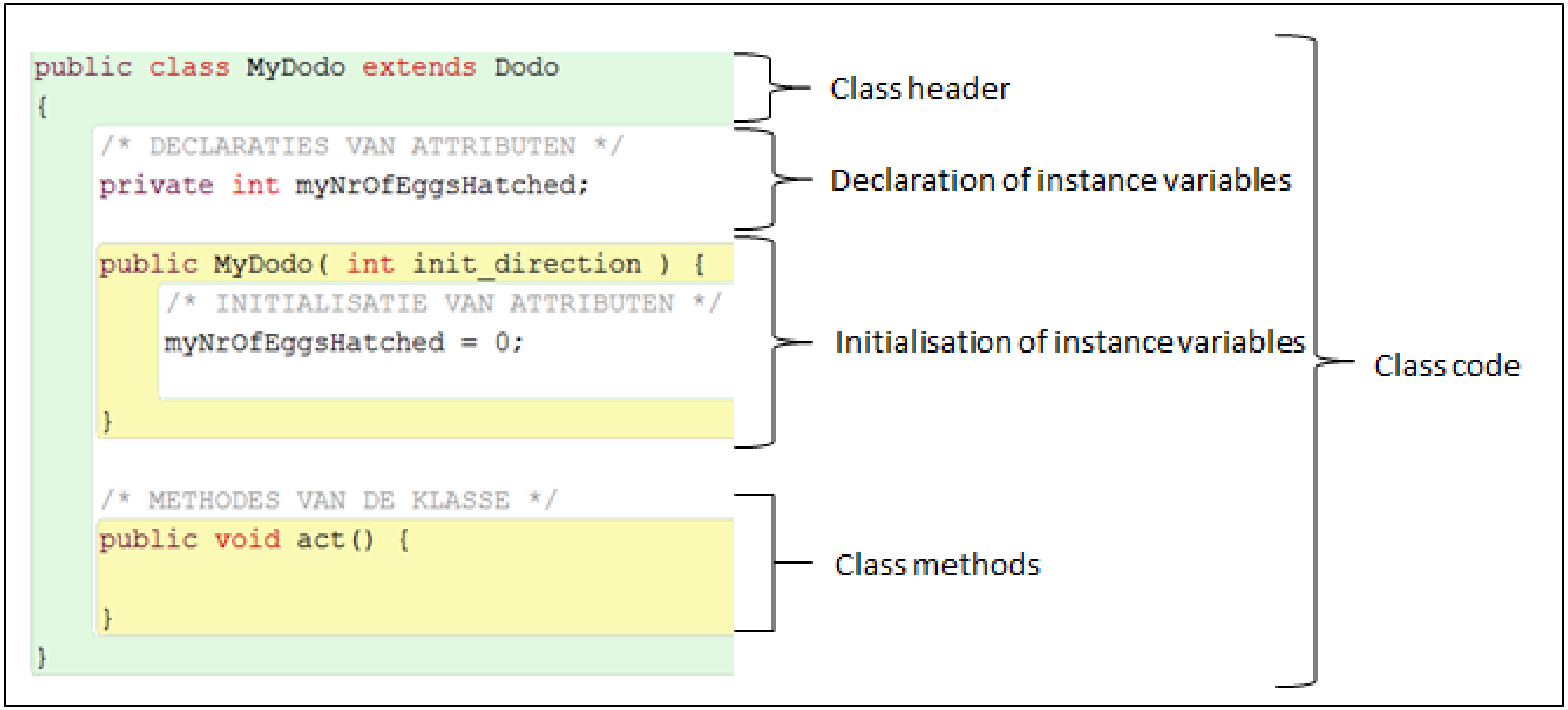
```
public class MyDodo extends Dodo
{
    private int myNrOfEggsHatched;

    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.

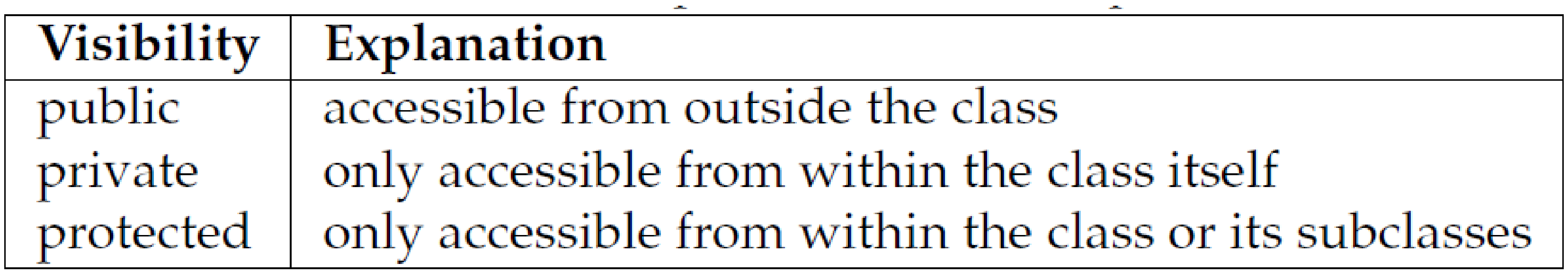
Class code



```
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

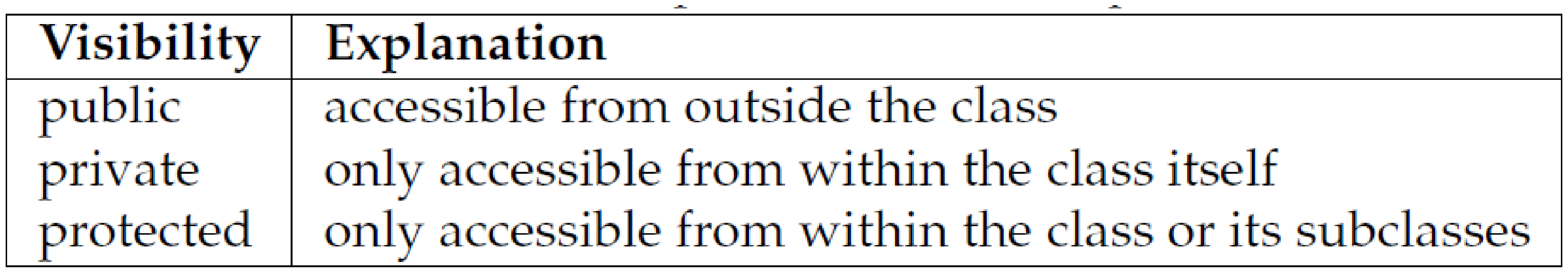
Visibility of variables / methods



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

Information hiding

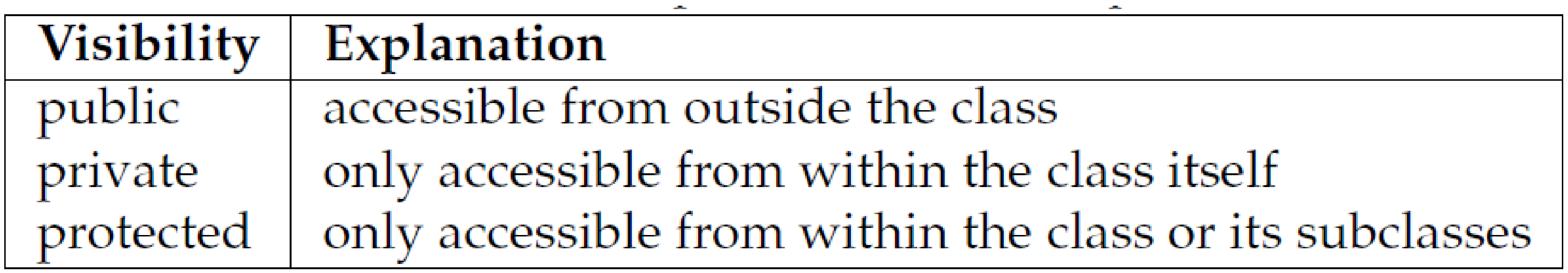
- Rule: make instance variables **private**



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

- This means: other objects can’t reach it!
- Solution: create (if needed)
  - **public getter** method
  - **public setter** method

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;
```

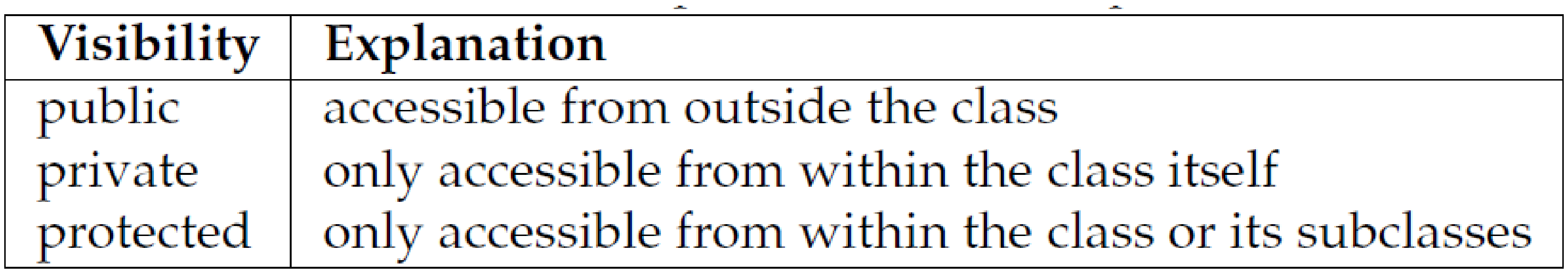
But… if myAge needs to **asked** for a (real) reason:

```
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”);

Wrapping up

Homework for Wednesday 8:30 May 18th:

- Assignment 6 and 7:
  - **FINISH assignment 6 and 7 up to and incl 4.1**

    (you may advance if you wish

    -> less homework next time)
  - email Java code and ‘IN’-answers to **sjaaksm@live.com**