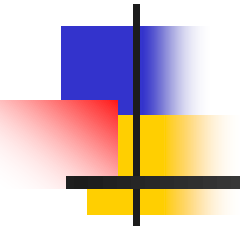
Algorithmic Thinking and Structured Programming (in Greenfoot)



Teachers:

Renske Smetsers-Weeda

Sjaak Smetsers

Today's Lesson plan (4)

- 20 min Quiz
- 20 min Looking back
  - What did we learn last week?
  - Discuss problems / homework (and handing-in)
- Assignment 4
- (10 min Wrapping up)

Discuss problems / homework

- Only hand in (via email):
  - MyDodo.java
  - Document with answers to **only** (IN) questions

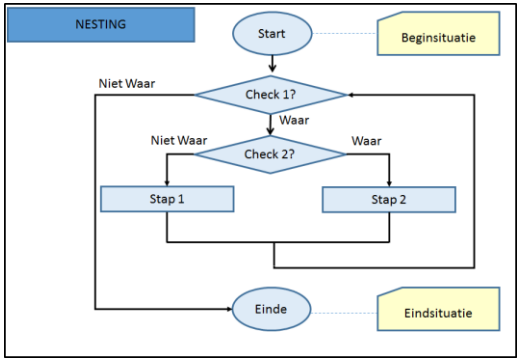
- Any problems? Please email!

Challenge & problem

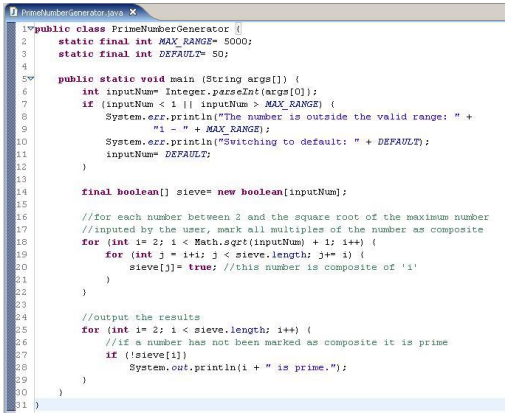
You must perform two aspects well:

1) Create a *problem-solving algorithm* (a disciplined and creative process)

We use a systematic approach



2) *Formulate* that algorithm *in terms of a programming language* (a disciplined and very precise process)



```
PrimeNumberGenerator.java
public class PrimeNumberGenerator {
    static final int MAX_RANGE= 5000;
    static final int DEFAULT= 50;

    public static void main (String args[]) {
        int inputNum= Integer.parseInt(args[0]);
        if (inputNum < 1 || inputNum > MAX_RANGE) {
            System.err.println("The number is outside the valid range: " +
                    "1 - " + MAX_RANGE);
            System.err.println("Switching to default: " + DEFAULT);
            inputNum= DEFAULT;
        }

        final boolean[] sieve= new boolean[inputNum];

        //for each number between 2 and the square root of the maximum number
        //inputed by the user, mark all multiples of the number as composite
        for (int i= 2; i < Math.sqrt(inputNum) + 1; i++) {
            for (int j = i+i; j < sieve.length; j+= i) {
                sieve[j]= true; //this number is composite of 'i'
            }
        }

        //output the results
        for (int i= 2; i < sieve.length; i++) {
            //if a number has not been marked as composite it is prime
            if (!sieve[i])
                System.out.println(i + " is prime.");
        }
    }
}
```

Always check that your algorithm is correct by running/testing the implementation!

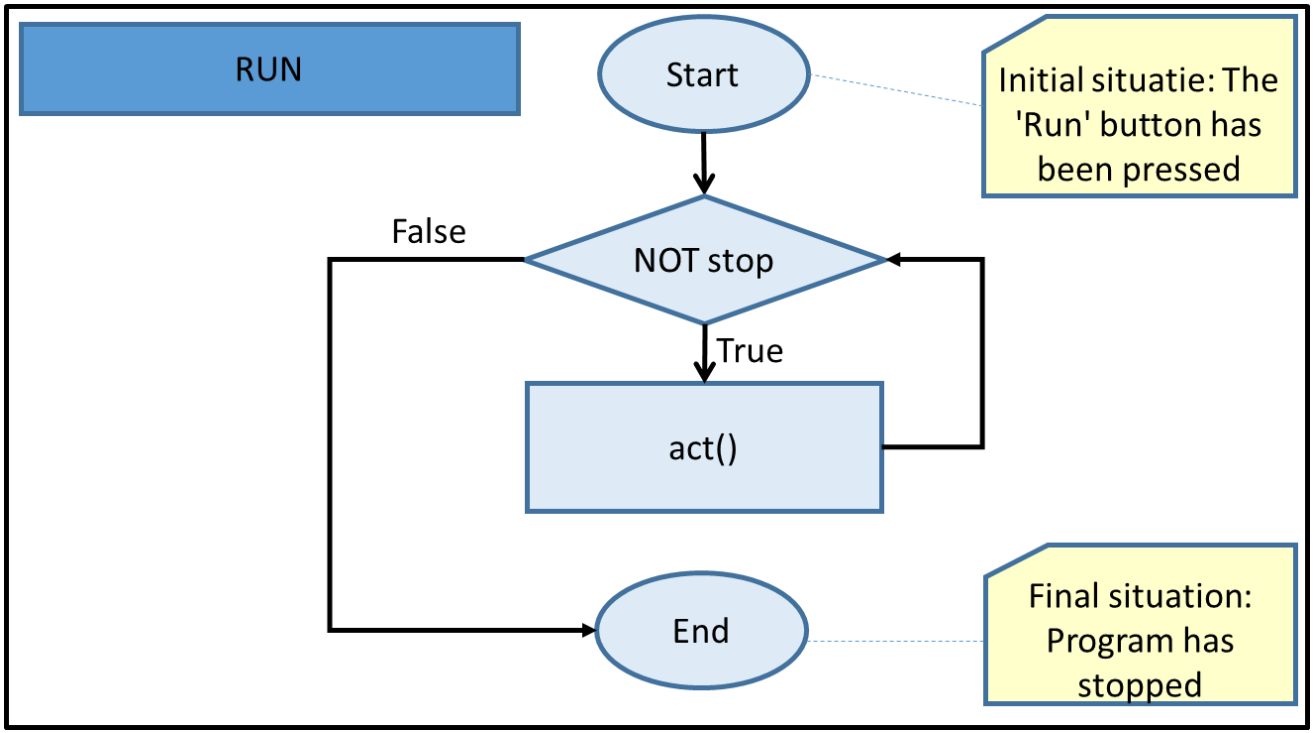
Today:

- Greenfoot Run: ‘Act’ in a while loop
  - Greenfoot.stop()
- Parameters
- Submethods: a method call in a method
- Boolean expressions (NOT, OR, AND)

Greenfoot Run

- Run is a special Greenfoot feature
- Run: Act called repeatedly
  - Act in a while loop

>Run: built –in iteration



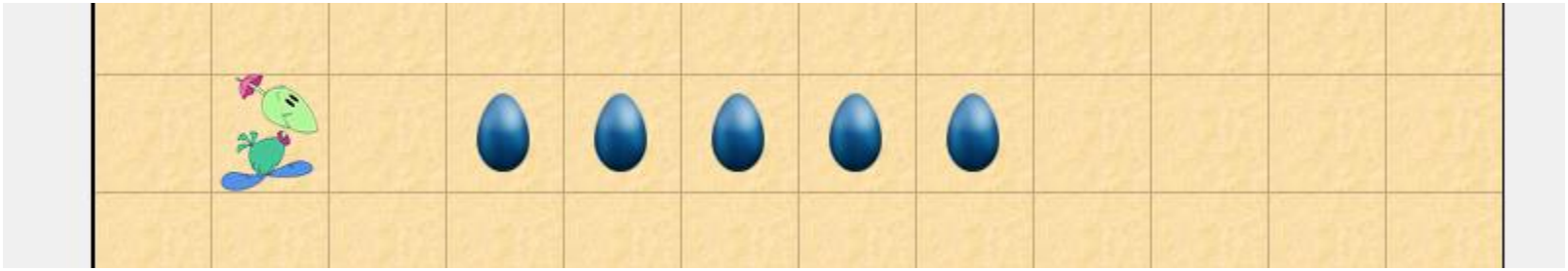
Can only be interrupted by:

- Pressing ‘Pause’
- Calling Greenfoot.stop()

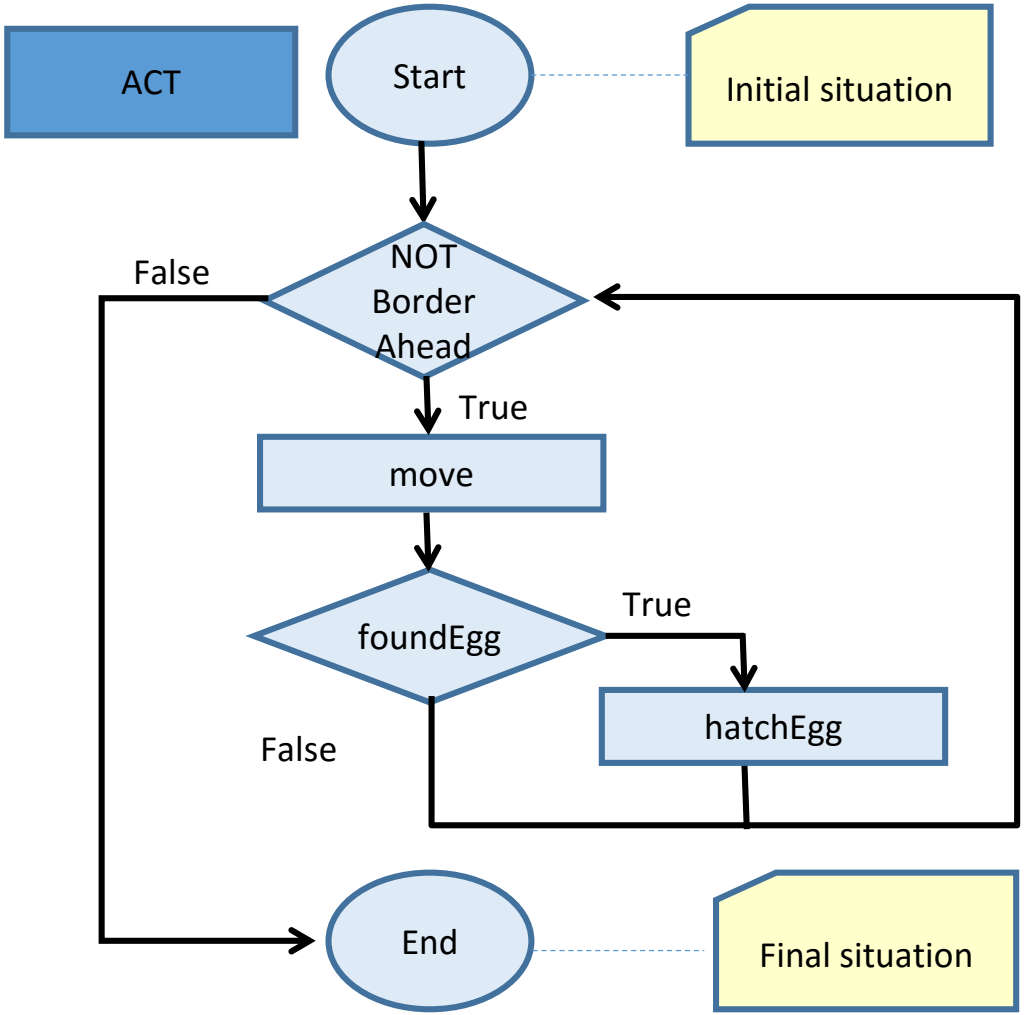
>Run: calling `act` repeatedly

What if your **`void`** `act()` contains a while-loop?

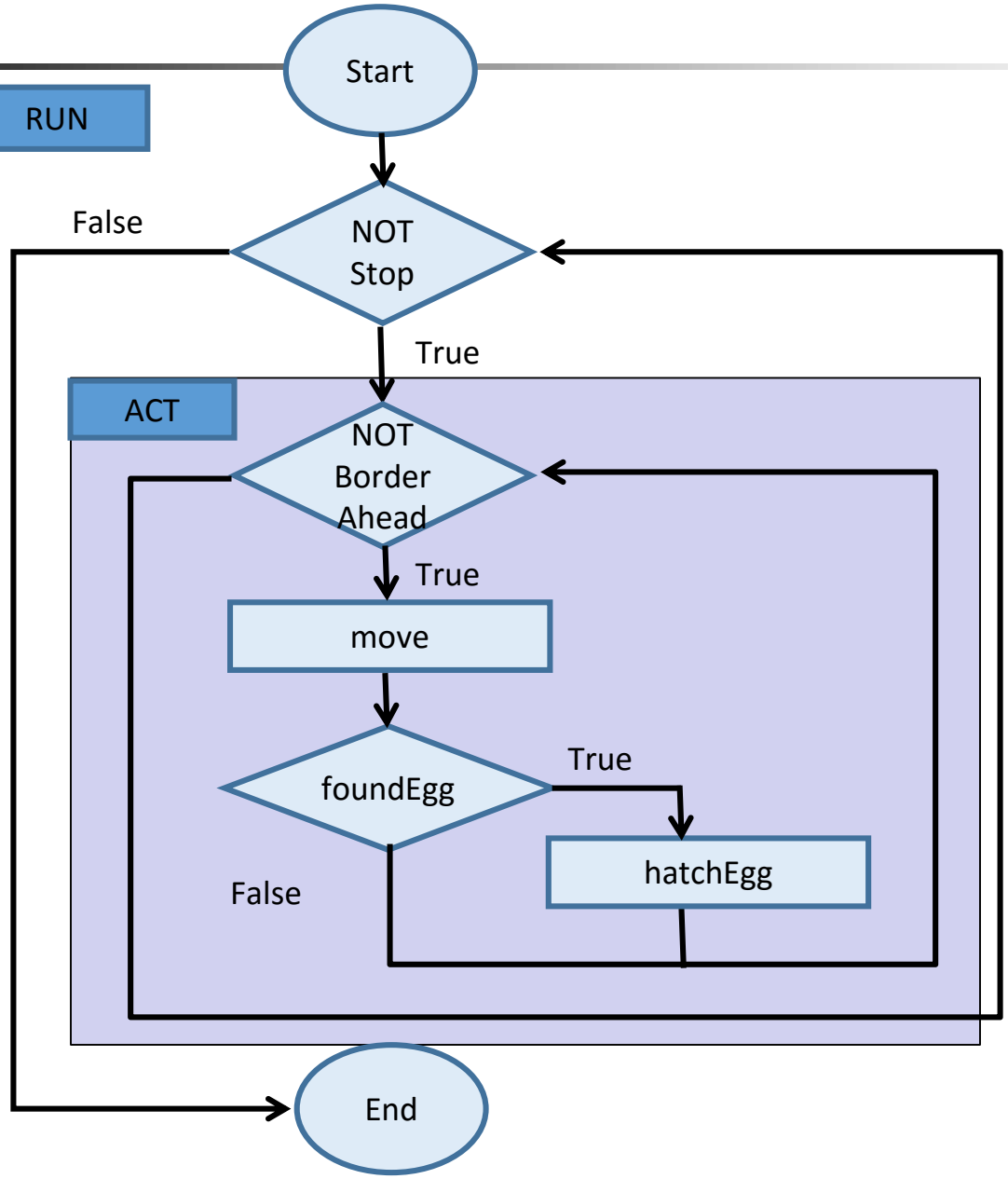
Example: hatching a row of eggs



Iteration in act

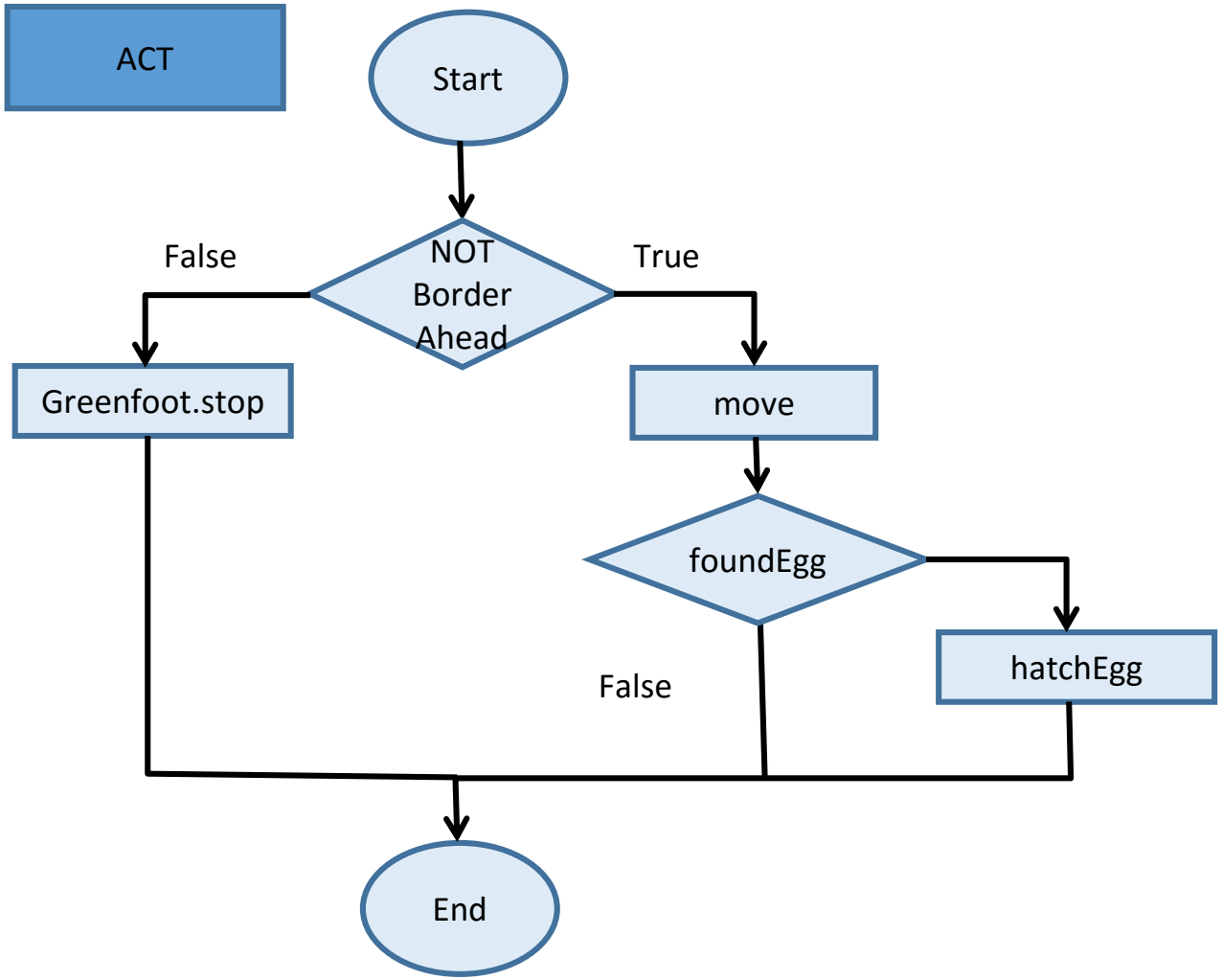


Act in Run



- One loop is superfluous
- Try to eliminate the act loop: Keep your act as simple/small as possible.

No iteration in act



The golden-promise:

- Don’t put too much work in the act method.
  - Avoid time-consuming while-loops or while-loops with ‘visible effects’.

Unplugged Songwriting

- Parameters
- Submethods

Songwriting: Parameters & Submethods

Old MACDONALD had a farm
E-I-E-I-O
And on his farm he had a cow
E-I-E-I-O
With a moo moo here
And a moo moo there
Here a moo, there a moo
Everywhere a moo moo
Old MacDonald had a farm
E-I-E-I-O

….

Song goes on for (just about) ever

More generic: Finding parameters

Old MACDONALD had a farm
E-I-E-I-O
And on his farm he had a **cow**
E-I-E-I-O
With a **moo moo** here
And a **moo moo** there
Here a **moo**, there a **moo**
Everywhere a **moo moo**
Old MacDonald had a farm
E-I-E-I-O

Old MACDONALD had a farm
E-I-E-I-O
And on his farm he had a **<ANIMAL>**
E-I-E-I-O
With a **<SOUND> <SOUND>** here
And a **<SOUND> <SOUND>** there
Here a **<SOUND>** , there a **<SOUND>**
Everywhere a **<SOUND> <SOUND>**
Old MacDonald had a farm
E-I-E-I-O

More generic: Using parameters

Old MACDONALD had a farm
E-I-E-I-O
And on his farm he had a **<ANIMAL>**
E-I-E-I-O
With a **<SOUND> <SOUND>** here
And a **<SOUND> <SOUND>** there
..
Old MacDonald had a farm
E-I-E-I-O

```
System.out.println("Old MACDONALD had a farm");
System.out.println("E-I-E-I-O");
System.out.println ("And on his farm he had a " + animal );
System.out.println("E-I-E-I-O");
System.out.println("With a " + sound + " " + sound+ "here" );
System.out.println("And a " + sound + " " + sound + "there" )
…
System.out.println("Old MACDONALD had a farm");
System.out.println("E-I-E-I-O");
```

Introducing parameters

```
public void singOldMcDonaldChorusWithParameters ( String animal, String sound ) {
    System.out.println( "Old MACDONALD had a farm" );
    System.out.println( "E-I-E-I-O" );
    System.out.println( "And on his farm he had a " + animal );
    System.out.println( "E-I-E-I-O" );
    System.out.println( "With a " + sound + " " + sound+ " here" );
    System.out.println( "And a " + sound + " " + sound + " there" );
    System.out.println( "Here a " + sound + ", there a " + sound );
    System.out.println( "Everywhere a " + sound + " " + sound );
    System.out.println( "Old MACDONALD had a farm" );
    System.out.println( "E-I-E-I-O" );
}
```

Generic: Using parameters

```
public void singOldMcDonaldSongWithParameters ( ) {
    singOldMcDonaldChorusWithParameters ( "cow", "moo" );
    singOldMcDonaldChorusWithParameters ( "pig", "oink" );
    singOldMcDonaldChorusWithParameters ( "duck", "quack" );
    singOldMcDonaldChorusWithParameters ( "lam", "baa" );
}
```

More generic: finding repetition

Old MACDONALD had a farm
E-I-E-I-O
And on his farm he had a cow
E-I-E-I-O
With a moo moo here
And a moo moo there
Here a moo, there a moo
Everywhere a moo moo
Old MacDonald had a farm
E-I-E-I-O

Defining submethods [1]

```
public void singOldMcHadFarm ( ) {
        System.out.println("Old MACDONALD had a farm");
}

public void singEIEIO ( ) {
        System.out.println("E-I-E-I-O");
}

public void singOldMcDonaldChorus ( String animal, String sound ) {
        singOldMcHadFarm ( );
        singEIEIO ();
        System.out.println( "And on his farm he had a " + animal );
        singEIEIO();
        …
}
```

Why submethods [1]: easy to change

- Change in 1 place
- From:

```
public void singOldMcHadFarm ( ) {
        System.out.println("Old MACDONALD had a farm");
}
```

- Into:

```
public void singOldMcHadFarm ( ) {
        System.out.println("Old McDonald had a farm");
}
```

Defining submethod with arguments

```
public void singHadAnimal ( String animal ) {
        System.out.println("And on his farm he had a " + animal );
}

public void singOldMcDonaldChoruss ( String animal, String sound ) {
        singOldMcHadFarm();
        singEIEIO();
        singHadAnimal ( animal );
        singEIEIO ();
        …
        singOldMcHadFarm();
        singEIEIO ();
}
```

Why submethods and arguments

- More generic:
  - Less code
    - Less mistakes
    - Easier to read / understand
  - Code can be used for more (… animals)
  - Easier to change
  - Easier to reuse

Wrapping up

Save your work!

Discuss how/when to finish off and who will turn it in.

Homework:

- Course downloads can be found at: http://www.cs.ru.nl/~S.Smetsers/Greenfoot/Dominicus/
- Finish Assignment 4
- **Hand via email to sjaaksm@live.com**