

# Assignment 4: Dodo gets smarter

## Algorithmic Thinking and Structured Programming (in Greenfoot)

© 2017 Renske Smetsers-Weeda & Sjaak Smetsers<sup>1</sup>

### Contents

<b>Introduction</b>	<b>1</b>
<b>Learning objectives</b>	<b>1</b>
<b>Instructions</b>	<b>1</b>
<b>Theory</b>	<b>2</b>
4.1 Variable Plan . . . . .	2
4.2 Operators . . . . .	5
4.3 Tracing variables . . . . .	6
4.4 The Swap Plan . . . . .	7
4.5 Sentinel-controlled Loop Plan . . . . .	8
4.6 Generic strategies . . . . .	8
4.7 The Count Plan . . . . .	9
4.8 Counter-controlled Loop Plan . . . . .	10
<b>Challenges</b>	<b>12</b>
4.1 Tracing code . . . . .	12
4.2 Swapping egg names . . . . .	14
4.3 Turn facing East . . . . .	14
4.4 Turn facing East using a <code>while</code> . . . . .	14
4.5 Go to a location . . . . .	15
4.6 Counters and <code>while</code> . . . . .	16
4.7 Number of steps until the edge of the world . . . . .	17
4.8 Counting number of eggs in a row . . . . .	18
4.9 Lay a trail of 6 eggs . . . . .	18
4.10 Lay a trail of 6 eggs, moving only 6 steps . . . . .	19
4.11 Lay a trail of <code>n</code> eggs . . . . .	19
4.12 Lay a trail of <code>n</code> eggs, check for valid input . . . . .	20
<b>Reflection</b>	<b>21</b>

---

<sup>1</sup>Licensed under the Creative Commons Attribution 4.0 license: <https://creativecommons.org/licenses/by/4.0/>

**Saving and Handing in**

**22**

## Introduction

When Mimi runs into a difficult problem, her first step is to make a plan. Having a plan makes Mimi feel stronger. There are plans for all types of things, like using variables, counting, looping, swapping things ....

One of those plans is having a 'memory'. Obviously, for Mimi to count her eggs, she has to be able to remember a number. We're going to use variables to store information. We're going to make Mimi smarter! By making Mimi smarter, we can also get her do more complex (composite) tasks.

In the following challenges we will teach (i.e. program) Mimi to do all kinds of things. It is important to make Mimi as smart as possible. We will teach her to do flflnew things, but want her to be able to do those in any similar situation she is faced with in the future. So, we're focussing on generic solutions for the problems she runs into.

This assignment's goal are:

- **Learn to use variables**
- **Come up with generic algorithms which can be used as a solution to multiple problems**

## Learning objectives

After completing this assignment, you will be able to:

- explain what **variables** are used for;
- **declare** and **initialize** variables and reassign values (apply the **Variable Plan**);
- analyse code and trace variables using a **tracing table**.
- use the **assignment operator** '=';
- use the **comparison operators** '==', '!=', '<', '<=', '>' and '>=';
- use the **arithmetic operators** '+', '-', '\*', and '/';
- use the **incrementing operator** '++' (and decrementing operator '--');
- recognize and apply **plans** for counting, repeating tasks and swapping values;
- use a **counter** to control loop execution (apply the **counter-controlled loop plan**);
- use a conditional statement to control loop execution (apply the **sentinel-controlled loop plan**);
- understand how methods use **parameters**;
- guard for unexpected values.

## Instructions

In this assignment you will carry on with your code from the previous assignment. Make a copy of that scenario to continue working with. To make a copy follow the next steps:

- Open your scenario from the previous assignment.
- In the Greenfoot menu at the top of the screen, select 'Scenario' and then 'Save As ...'.
- Check that the window opens in the folder where you want to save your work.

- Choose a file name containing your own name(s) and the current assignment number, for example: `Asgmt4_John`.

Note: We recommend that you to continue working with your own code. If it is **absolutely** impossible to carry on working with your own code from the previous assignment, then you may download a new scenario from the course website<sup>2</sup>.

Throughout the assignment you will also need to answer some questions. The following must be handed in:

- All flowcharts: use pencil and paper, or go to <https://www.draw.io/>;
- Your code: the file `MyDodo.java` contains all your code and must be handed in;
- The reflection sheet: complete and hand it in.

You must discuss all other answers with a programming partner. Jot down a short answer on (the assignment) paper.

There are three types of challenges:

★	<b>Recommended.</b> Students who need more practice or with limited programming experience should complete all of these.
★★	<b>Mandatory.</b> Everyone must complete these.
★★★	<b>Excelling.</b> More inquisitive tasks, designed for students who completed 2 star tasks and are ready for a bigger challenge.

Students who skip 1-star challenges should complete all 3-star challenges.

**A note in advance:**

- In this assignment you may only make changes to the `MyDodo` class;
- You may use methods from the `MyDodo` or `Dodo` class, not from the `Actor` class;
- Teleportation is not permitted: if Mimi needs to get somewhere, she must walk there!

## Theory

### Theory 4.1: Variable Plan

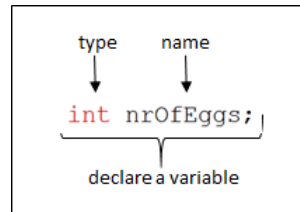
A variables is used to store information. In order to use variables, these must first be:

1. **Declared:** given a name and a type.
  - **Type:** such as `int`, `boolean`, `String`, or, as we shall see later on, a class or another object. The variable type describes what type of a value a variable can store.
  - **Name:** for example `Mimi`, or `nrOfEggs`.
2. **Initialized:** given an initial<sup>3</sup> value which can be changed later.

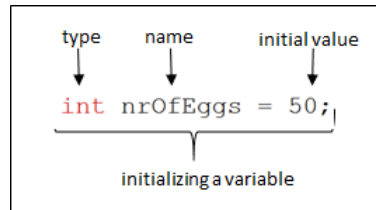
Before you can use a variable, it must be created. This is called a declaration:

<sup>2</sup><http://course.cs.ru.nl/greenfoot/>

<sup>3</sup>It is good programming practice to always give a variable an initial value. In Java this is not mandatory.



Usually, at the same time, you will assign it a value. This is called initialization:



### Variable Types and declaration

Type	Meaning	Example variable declaration	Example variable use
int	Integer: whole number	int nrOfEggsHatched = 2	if ( nrOfEggsHatched == 12) ...
boolean	True OR False	boolean doneLookingForEgg = false	if ( doneLookingForEgg ) ...
String	Text	String name = "Mimi"	showCompliment ( name );

### Assigning a value

To assign a value to a variable you must use '='. The '=' sign means 'becomes'.

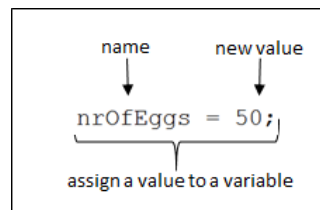


Figure 1: The variable nrOfEggs becomes 50

The variable and the value must be the same **type**. This means that the type of the variable on the left-hand-side of the '=' should be the same as the type of the value on the right-hand-side. The types in the example above are the same:

- the left-hand-side: nrOfEggs has been declared as an int,
- the right-hand-side: "50" is an int.

### Using a variable

What you can do with a variable (which operations you can use) depends on its type. For example, variables which store a text (or String) can be 'stuck together' (or concatenated). Numerical values (such as int) can be compared, multiplied, added, incremented ...

Examples are:

- `nrOfDozens = nrOfEggs / 12;`  
the variable nrOfDozens is assigned the value of nrOfEggs divided by 12. Note: the division that is used here is called an integer division, that is to say that the result is an integer (whole number) and any remainder will be thrown away.

- `nrOfEggs = nrOfEggs + 1;`  
the variable `nrOfEggs` is incremented by 1 (for example, when Mimi lays another egg);
- `nrOfEggs = nrOfEggs ++;`  
the variable `nrOfEggs` is incremented by 1 (same as above);
- `totalNrOfEggs += nrNewEggsFound ;`  
same as: `totalNrOfEggs = totalNrOfEggs + nrNewEggsFound;`

**Note:**

- When declaring a variable, indicate its type. When using (the value of) a variable, you don't refer to its type.<sup>4</sup>
- A variable can also be passed to a method as a parameter. For example in the following method call: `layNrOfEggs ( nrOfEggs )`; Here, in the method call of `layNrOfEggs`, the variable `nrOfEggs` is passed as a parameter. Based on this parameter, Mimi knows how many eggs she must lay.

**Naming conventions for a variable**

The name of a variable:

- is meaningful: it corresponds to what the variable means (one exception: the name of a counter variable in a loop may be one letter, such as `i`);
- consists of one or more nouns;
- is written in lowerCaseCamel: it starts with a lowercase letter, and each subsequent 'word' starts with a capital letter;
- consists of letters and numbers: it does not contain spaces, commas, or other 'strange' characters (one exception: `'_'` may be used);
- for example: `nrEggsFound`.

**Life span**

A variable that is created inside a method declaration exists only for that particular method. After the method is executed, the variable is destroyed. Thus, these variables have a limited life span (also called scope). A variable's life begins where it is declared and ends when the method stops.<sup>5</sup>

The Greenfoot editor helps you recognize the life span by using different background colors. For example, in the picture below you can see that the scope of the `int stepsTaken` variable is limited to the white area and all areas that are enclosed by the white area in which it is declared, in this case the pink area.

```
public int walkAndCountSteps() {
    int stepsTaken=0;
    while( canMove() ){
        stepsTaken++;
        move();
    }
    return stepsTaken;
}
```

<sup>4</sup>This is just like when you call a method, where you don't have to repeat the entire signature of the method. Moreover, Java does not even allow you to do so.

<sup>5</sup>Actually Java's scope rules are somewhat more complicated, but for the time being we'll keep it as simple as possible

Parameters also have a life span: this is limited to the body of a method (or constructor). For example, in the picture below you can see that the scope of the `int nrStepsToTake` parameter is limited to the white area of the method (and the enclosed pink area).

```
public void takeSteps (int nrStepsToTake){
    int stepsTaken = 0;

    while (stepsTaken < nrStepsToTake){
        stepsTaken++;
        move();
    }
}
```

Variables that are declared in a class (outside of a method and thus not within a particular method) exist as long as the object exists (you will learn more about this in the next assignment).

## Example

We write a method which squares a number (given as parameter):

```
/**
 * This method returns the square of a given number
 */
public int square (int number){

    int result = 0 ;           // declare and initialize the result variable

    result = number*number; // assign a value to result: the (parameter) number squared

    return result; // the method returns the result after squaring
}
```

### Note:

A variable that is declared in a method (and thus can only be used within that method) is called a local variable. In this example, `int result` is a local variable because it has been declared in the method `square`. In addition, the parameter `int number` can only be used within this method.

## Theory 4.2: Operators

### Assignment operator

The '=' operator assigns (gives) a value to a variable. For example:

Statement	Result values
<code>int nrOfEggsInNest = 4;</code>	<code>nrOfEggsInNest</code> has the value 4
<code>int nrOfEggsEatenBySnake = nrOfEggsInNest</code>	<code>nrOfEggsEatenBySnake</code> is 4 and <code>nrOfEggsInNest</code> is 4

### Note:

Using variables to assign values **copies** the value. In the example above `nrOfEggsEatenBySnake` gets the value 4, but the `nrOfEggsInNest` is also still 4.

## Comparison operators

The following operators compare numerical values:

Operator	Meaning	Example
<code>==</code>	is equal to	<code>number == 4</code>
<code>!=</code>	is NOT equal to	<code>number1 != number2</code>
<code>&gt;</code>	is larger than	<code>number &gt; 3</code>
<code>&gt;=</code>	is larger or equal to	<code>number1 &gt;= number2</code>
<code>&lt;</code>	is smaller than	<code>5 &lt; number</code>
<code>&lt;=</code>	is smaller or equal to	<code>number &lt;= 5</code>

The result of a comparison is always a boolean (thus, true OR false).

## Arithmetic operators

The following operators perform arithmetic operations on numerical values:

Operator	Meaning	Example
<code>+</code>	addition	<code>result = 4 + number;</code>
<code>-</code>	subtraction	<code>result = number - 4;</code>
<code>*</code>	multiplication	<code>result = 5 * number;</code>
<code>/</code>	division	<code>result = number / 5;</code>

## Increment and decrement operators

The following operators increase or decrease the value of a variable by one:

Operator	Meaning	Example
<code>++</code>	increase by one	<code>result ++;</code>
<code>--</code>	decrease by one	<code>result --;</code>

## Example

- Initialization: Assume Mimi has found 4 eggs, then the following statement declares and initializes the variable: `int nrEggsFound = 4;`
- Addition operator: If she then finds another two eggs, then the following statement increases the value of `nrEggsFound` by two: `nrEggsFound = nrEggsFound + 2;`
- Decrement: And if she loses an egg, the following statement decreases the value by one: `nrEggsFound --;`

The above operations change the value of a variable. To check whether Mimi now has five eggs, we use the following boolean expression: `nrEggsFound == 5`. This is indeed true.

## Theory 4.3: Tracing variables

Keeping track of variables and reviewing their values at certain points in a program is called tracing. Tracing is a useful skill for detecting errors in code.

## Example:

We will now trace the variables `number1`, `number2` and `number3` using the following code:



```

public void practiceTracingVariables( ){
    int a = 2;
    int b = 3;
    int c = a * b;
    System.out.println("Value of a is: " + a + ", b is: " + b + ", c is: " + c);

    b = c - a;
    System.out.println("Value of a is: " + a + ", b is: " + b + ", c is: " + c);

    a = a + b + c;
    System.out.println("Value of a is: " + a + ", b is: " + b + ", c is: " + c);

    c = b * a;
    System.out.println("Value of a is: " + a + ", b is: " + b + ", c is: " + c);
}

```

Figure 2: Code to practice tracing

A tracing table is used to keep track of each variable's value after executing each line of code:

Statement	Value after executing the statement		
	a	b	c
int a = 2;	2	-	-
int b = 3;	2	3	-
int c = a * b;	2	3	6
b = c - a;	2	4	6
a = a + b + c;	12	4	6
c = b * a;	12	4	48

Note: '-' indicates that the variable has not yet been declared or initialized. At the end of the program, the value of number1 is equal to 12, the value of number2 is equal to 4, and the value of number3 is equal to 48.

### Theory 4.4: The Swap Plan

To swap the values of two variables (called a triangular swap plan, you need an additional temporary 'helper' variable.

#### Example:

The goal is to swap the values of two eggs. A blue egg is worth 2 points, and a golden egg is worth 10 points. These values are stored in the variables `blueEggValue` and `goldenEggValue`, respectively. Both variables are of type `int`. This is how the swapping is done:

1. The `temporaryEggValue` gets the value of the `blueEggValue`;
2. The `blueEggValue` gets the value of the `goldenEggValue`;
3. The `goldenEggValue` gets the value of the `temporaryEggValue`;

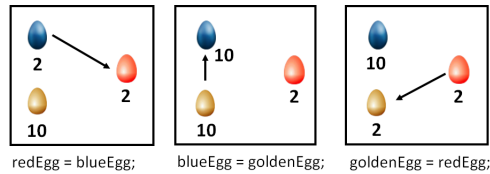


Figure 3: Plan for swapping values of two variables

Statement	Value after executing the statement		
	blueEggValue	goldenEggValue	temporaryEggValue
initial values	2	10	-
temporaryEggValue = blueEggValue ;	2	10	<b>2</b>
blueEggValue = goldenEggValue ;	<b>10</b>	10	2
goldenEggValue = temporaryEggValue ;	10	<b>2</b>	2

**Note:**

You may want to try this with files. Open your file explorer. Make a file "A.txt" and a file "B.txt" each containing a different text, for example by using Notepad. Now, by means of only copying files, try to switch the files around.

**Theory 4.5: Sentinel-controlled Loop Plan**

A sentinel can be used to determine when a loop is to stop being executed. As long as a particular condition is not met, the loop is repeated. This is called a sentinel-controlled loop plan.

**Example:**

As a concrete example, let's have a look at MyDodo's code for `void walkToWorldEdge ( )`.

As long as the sentinel `! borderAhead ( )` is true, the loop is repeated and Mimi takes a step.

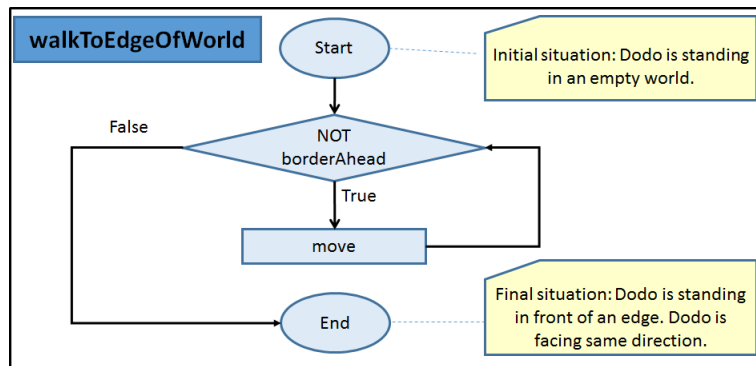


Figure 4: Flowchart for sentinel controlled loop plan

When the sentinel value is reached, the loop is not executed again.

**Theory 4.6: Generic strategies**

Programs that are generic can be used as a solution to more problems. Choosing appropriate and general boundaries can make your solution applicable to more solutions. For example, in a world with a different size.

Avoid using hard-coded values such as '12'. Instead, determine the boundary values by:

- using a sentinel such as `borderAhead()` or
- calculating the number of rows `nrOfRowsInWorld` as follows:

```
World world = getWorld();           // gets world to determine world height
int nrOfRowsInWorld = world.getHeight(); // stores world height in nrOfRowsInWorld
```

- using parameters as input for a method. A method which depends on parameters is more flexible than one that doesn't. The parameter is used to determine specific values, such as how often (part of) an algorithm should be repeated. One method can be used to solve multiple similar problems. For example, Mimi can use the method `void jump( int distance )` to jump several distances, depending on the parameter it is given. On the other hand, using `void move( )`, Mimi can merely take one step.

```
public void jump( int distance ) {
    int nrStepsTaken = 0;           // set counter to 0
    while ( nrStepsTaken < distance ) { // check if more steps must be taken
        move();                     // take a step
        nrStepsTaken++;             // increment the counter
    }
}
```

Figure 5: Using a parameter `distance` for a generic method

### Theory 4.7: The Count Plan

A counter is a variable that is used to keep track of how often something happens. For example, a variable can be used to count how often the code in a `while` loop is executed. A Count Plan has the following structure:

- Initialize a counter variable (usually to 0);
- Inside the loop, modify the counter variable (usually increment);
- At the end of the method, return the counter variable.

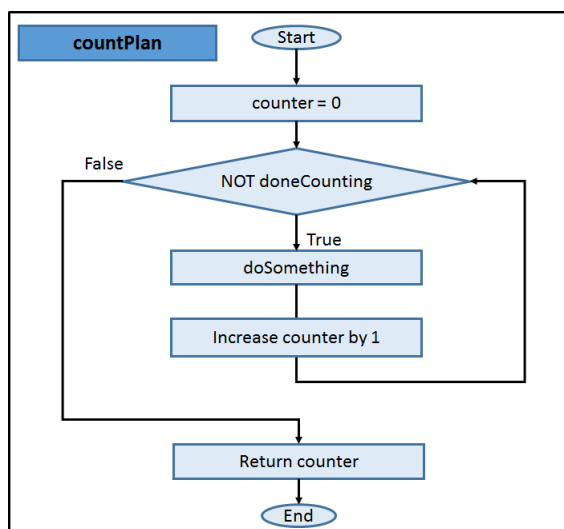


Figure 6: Flowchart framework for Count Plan

Figure 7: Code framework for the Count Plan

```

/**
 * Example of a count plan: counting steps till the end of the world
 */
public int countSteps( ) {
    int nrStepsTaken = 0;           // set counter to 0
    while ( !borderAhead() ) {      // check if more steps can be taken
        move();                     // take a step
        nrStepsTaken++;             // increment the counter
    }
    return nrStepsTaken++;
}

```

### Theory 4.8: Counter-controlled Loop Plan

The counter can also be used in the loop's condition to determine how often the loop should be executed. That way, you know when you're done looping. Even during the last repetition, all the code in the loop will be executed (it will never stop half-way through a loop). This is called a counter-controlled loop plan.

### Example:

As a concrete example, let's have a look at MyDodo's code for `void jump (int distance)` (see figures 8 and 9). This method makes Mimi take steps repeatedly until the required distance has been reached. A counter (`nrStepsTaken`) is used to keep track of how many steps Mimi has taken. With each step, the counter is incremented (increased by 1). As long as the counter hasn't reached the required distance, the loop is executed again. When the counter (`nrStepsTaken`) is equal to the required number of steps (`distance`), the method ends.

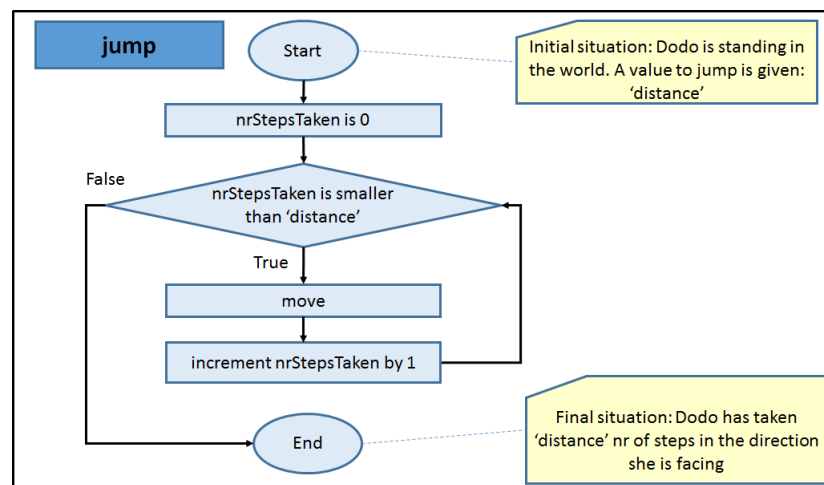


Figure 8: Flowchart for jump method

```

public void jump( int distance ) {
    int nrStepsTaken = 0;           // set counter to 0
    while ( nrStepsTaken < distance ) { // check if more steps must be taken
        move();                     // take a step
        nrStepsTaken++;             // increment the counter
    }
}

```

Figure 9: Code for `jump` method**Note:**

Three common mistakes or misconceptions are:

- to forget to increment the counter in the loop. Then the program will never come out of the loop and will repeat for ever.
- to use the incorrect comparison operator in the condition. If counter is set to 0, the '`<`' comparison operator must be used instead of '`<=`'.
- to think that the method ends as soon as the counter is incremented to the desired value. On the last pass, all the code in the while loop will be executed. Then, when the conditional expression is checked again, the method will end.

## Challenges



Please read **Theory 4.1: Variable Plan.**



Please read **Theory 4.2: Operators.**



Please read **Theory 4.3: Tracing variables.**



Please read **Theory 4.4: The Swap Plan.**



### Challenge 4.1: Tracing code

In the following exercises we will practice using variables and operators. You can use Greenfoot to check your answers, see the example code in figure 10

```
public void practiceTracingCode ( ){
    int nrOfEggsFound = 3;

    nrOfEggsFound ++;

    System.out.println("Value of nrOfEggsFound is: " +nrOfEggsFound );
}
```

Figure 10: Code to practice tracing

- a) What does the value of `int nrOfEggsFound` become? Match the code for A,B,C,D to the value of `intNrEggsFound` after code execution:

Question	Code
A	<code>int nrOfEggsFound = 3;</code> <code>nrOfEggsFound ++;</code>
B	<code>int nrOfEggsFound = 2;</code> <code>nrOfEggsFound = nrOfEggsFound + 4;</code>
C	<code>int nrOfEggsFound = 1;</code> <code>nrOfEggsFound --;</code>
D	<code>int nrOfEggsFound = 1;</code> <code>nrOfEggsFound +=2;</code>

nrOfEggsFound
0
2
3
4
6

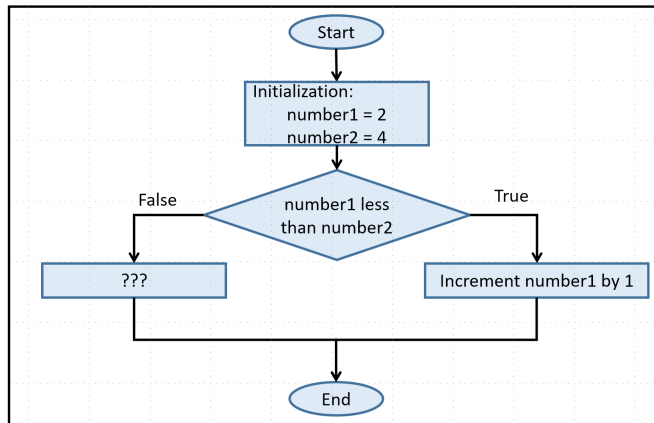
- b) What do the values of `int number1` and `int number2` become?

```
int number1 = 2;
int number2 = 4;

number1 = number1 + number2;
number2 = number1 + number2;
```

statement	number1	number2
initialization	2	4
<code>number1 = number1 + number2;</code>		
<code>number2 = number1 + number2;</code>		

- c) After execution, the values of `number1` and `number2` should be equal to each other. Fill in the missing pieces of code and flowchart, and fix the code.



```

int number1 = 2;
int number2 = 4;

if ( ??? ){
    number1++;
} else {
    number2 = number1;
}
  
```

- d) What do the values of `int number1` and `int number2` become? After execution, the values of `number2` should be 36. If this is not the case, fix the mistake.

```

int number1 = 2;
int number2 = number1 * 3;

number1 = number2;
number2 = number1 + number1;
  
```

- e) Fill in the tracing table for the following code snippet.

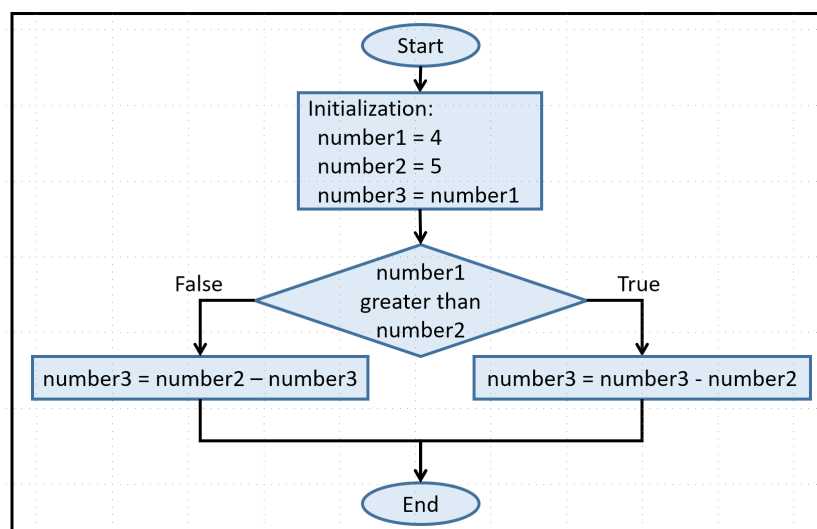
```

int number1 = 6;
int number2 = 3;

number1 = number2;
number2 = number1;
if( number1 == number2 ){
    number1 = number1 + number2;
}
  
```

statement	number1	number2
initialization	6	3
number1 = number2;		

- f) What does the value of `int number3` become?



g) What does the value of `int number3` become?

```
int number1 = 10;
int number2 = 8;
int number3 = 4;

number3 = doSomethingWithTwoNrs( number1, number2 );
```

where:

```
public int doSomethingWithTwoNrs ( int a, int b ) {
    int result = (a + b)/2;
    return result;
}
```



### Challenge 4.2: Swapping egg names

Mimi accidentally gave her eggs the wrong names! Can you help her swap the names?



Figure 11: Dodo mixed up the names of these eggs

- Determine a strategy to swap the names around.
- Translate your strategy code. Tip: Having trouble getting started? Have a look at figure 12.
- Print the values to the console to test that it works.
- Note the final values of any variables you used.

```
public void swapEggNames() {
    String nameOfBlueEgg = "Gold";
    String nameOfGoldenEgg = "Blue";

    System.out.println ("Initial values before the swap:");
    System.out.println ("Name of blue egg is: " + nameOfBlueEgg);
}
```

Figure 12: First steps for writing swap plan code



### Challenge 4.3: Turn facing East

Write a method `void faceEast( )` which turns Mimi so that she faces East. Tip: You can use `getDirection( )==1` to check if Mimi is facing East. You have now learned how to compare a variable and a number.



Please read **Theory 4.5: Sentinel-controlled Loop Plan.**





### Challenge 4.4: Turn facing East using a `while`

Write a method `void faceEast( )` which turns Mimi so that she faces East. Make use of a `while` (not an `if .. then .. else`).

You have now learned how to compare a variable and a number. Furthermore you used a `while`-loop to repeat steps.



Please read **Theory 4.6: Generic strategies.**



### Challenge 4.5: Go to a location

We will now write a method `void goToLocation( int coordX, int coordY )` that sends Mimi to a location with the following coordinates ( `coordX`, `coordY` ).

- a) Come up with a suitable high-level algorithm and draw the flowchart. Determine which sub-methods you will need. Tips:
  - Determine Mimi's coordinates, and then determine how many steps in each direction she must take to get to her destination.
  - It may be useful to consider different cases individually, depending on which direction she needs to head. Consider describing the cases using the terms 'WEST', 'EAST', 'NORTH', and 'SOUTH'.
  - It may be useful to have a sub-method such as `boolean locationReached( int coordX, int coordY )` to check if the destinations has been reached.
  - Check if you can optimize your flowchart using a `while`.
- b) Write the corresponding code. Don't forget to add comments. Tips:
  - You can use `setDirection( WEST );` to make Mimi face to the West (and likewise any other direction).
  - You can use `getX( )` and `getY( )` to get Mimi's coordinates.
- c) Test your method by right-clicking and filling in several different values. Try boundary values such as (0,0) and (11,11)? Test for invalid values such as (14,14) and (-1,-1)?
- d) Adjust your program so that it deals with invalid coordinates (so called unexpected values) correctly:
  - (a) Write a separate sub-method `boolean validCoordinates( int coordX, int coordY )` which checks if the given coordinates (the arguments) are valid (in other words: if they exist in Mimi's world).
  - (b) If invalid coordinates are given, show an error message using:
 

```
showError( "Invalid coordinates" );
```
  - (c) If the input is invalid, Mimi should stay in her place (do nothing);
  - (d) Your program should work for any world size, not only a world with 12 by 12 cells. Tip: To get the world's width, first call: `World world = getWorld( );` and then ask its width: `world.getWidth( )` (see Theory 4.6).
- e) Test your code modifications.

We have now seen how to use and compare variables. We also saw how to use a combination of operators with different parameters, variables, and numbers to build more complex conditional expressions.



Please read **Theory 4.7: The Count Plan.**



Please read **Theory 4.8: Counter-controlled Loop Plan.**



### Challenge 4.6: Counters and `while`

It's time to practice using the `while` loop. Just like in the first challenge (4.1), we're going to review code and trace how the variables change as the code runs. This is a good way to really understand how a `while` loop works. That way, you'll make less (hard to find) mistakes while coding. Have a look at the following pieces of code and try to figure out what happens to the variables. Use Greenfoot to check your answers. Use your `void practiceTracingCode ( )` method from challenge 4.1 to run the code.

- a) After running the following code, what are the values of `counter` and `nrOfStepsToTake`? How often does the method `move ( )` get called? Does the code work correctly?

```
int nrOfStepsToTake = 6;
int counter = 0;

while( counter <= nrOfStepsToTake ){
    move();
    counter++;
}
```

- b) Complete the following table for each time that the code in the `while` is executed.

```
int number1 = 2;
int number2 = 5;
int counter = 0;

while( number1 <= number2){
    move();
    number1 ++;
    counter ++;
}
```

Nr of while-loops executed	Value of number1 after while	Value of number2 after while
0		
1		
2		

- c) Modify the following **conditional expression** in the `while` so that `move ( )` is called three times.

```
int number1 = 10;
int number2 = 8;

while( number1 > number2 ){
    move();
    number1--;
}
```

d) Have a look at the method `jump` as described in 4.8.

- (a) Add (i.e. copy-paste) the method `void jump(int distance)` to the `MyDodo` class.
- (b) For each of the following, indicate how often the code in the `while` (of `void jump(int distance)`) is executed:
  - i. if `distance` is equal to 5,
  - ii. if `distance` is equal to 1?
  - iii. if `distance` is equal to 0?
  - iv. if `nrStepsTaken` is equal to 3 and `distance` is equal to 5?

Tip: You may want to print some values in the console.

We have now seen how to use a counter variable to determine how often code (a `while` loop) should be executed in order to get correct results.

### ★ Challenge 4.7: Number of steps until the edge of the world

In this challenge you will use a Count Plan to write a method which returns how many steps Mimi has to take to get to the edge of the world (see Theory 4.7 for more information about the Count Plan). The following flowchart and code shows a framework for the method `walkToWorldEdgeAndCountSteps ( )` based on the Count Plan:

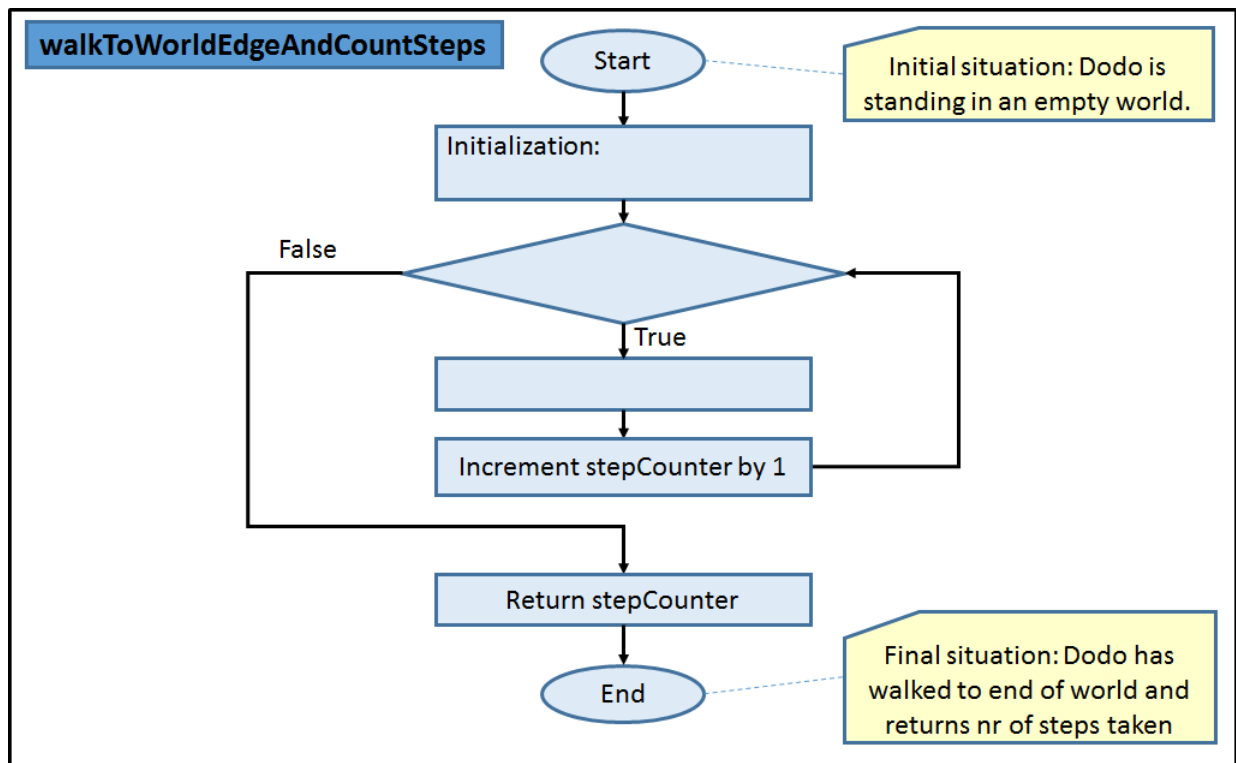


Figure 13: Flowchart for counting steps to the edge of the world

```

public int walkToWorldEdgeAndCountSteps() {
    int stepCounter = 0;

    while ( ) {
        stepCounter++;
    }

    return stepCounter;
}

```

Figure 14: Code for counting steps to the edge of the world

- Initialize a counter variable;
- Inside the loop, modify the counter variable (usually increment).
- At the end of the method, return the counter variable.
- Test your method.
- What is the smallest value that your method can return as a result? And the largest?

We have now used a counter variable to keep track of how often a `while` loop has been executed. We also wrote a method that returns the variable so that it can be used in other parts of the code.



### Challenge 4.8: Counting number of eggs in a row

Write an accessor method `int countEggsInRow` which makes Mimi walk to the edge of the world, count how many eggs she finds in that row (or column) and then go back to the beginning of the row (accessor method).

- Choose an appropriate variable to store the number of eggs found.
- Draw the corresponding flowchart. Return the number of eggs found as a result. Tip: The `return` is the very last statement that can be called. Mimi must thus first go back to the beginning of the row, before returning the value. It may be useful to write a sub-method which makes Dodo return to her initial position.
- Write the code. Don't forget to add JavaDoc comments.
- Test your method by right-clicking. Also test with an egg in the front and the end of the row.

We have now used a variable to store information which was gathered throughout the execution of a `while` loop (The Count Plan).



### Challenge 4.9: Lay a trail of 6 eggs

We want to teach Mimi to leave a trail of 6 eggs. The initial and final situations are as follows:

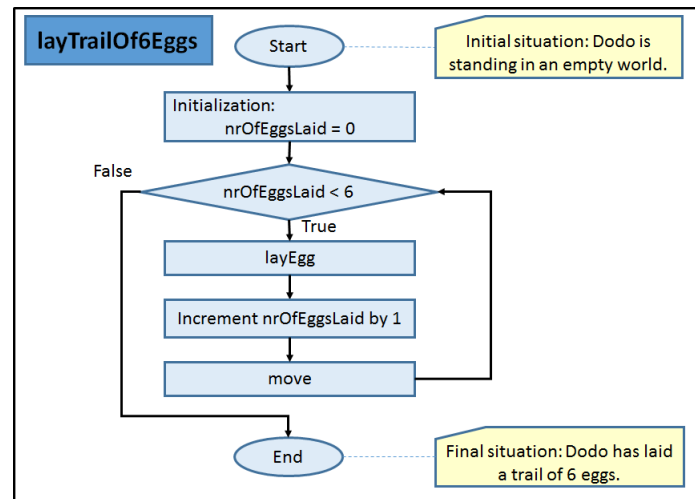


Figure 15: Initial situation



Figure 16: Final situation: Dodo has laid 6 eggs and is standing behind trail

Write a method `void layTrailOf6EggsAndMove ( )` which corresponds to the flowchart in figure 20.

Figure 17: Flowchart for method `layTrailOfEggsAndMove ( )`

We have now used a counter variable (`nrOfEggsLaid`) to determine how often a while loop is executed.

### ★ Challenge 4.10: Lay a trail of 6 eggs, moving only 6 steps

Again (as in Challenge 4.9), we want to teach Mimi to leave a trail of 6 eggs. This time, she must end up sitting on her final egg. The initial and final situations are as follows:

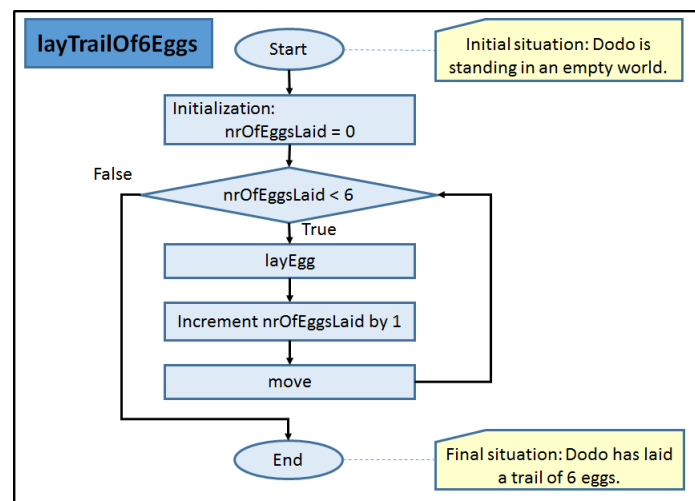


Figure 18: Initial situation



Figure 19: Final situation: Dodo has laid 6 eggs and is sitting on her final egg

This way, Mimi can lay a trail of eggs all the way until the border of the world. Write a method `void layTrailOf6Eggs ( )` which corresponds to the flowchart in figure 20.

Figure 20: Flowchart for method `layTrailOfEggs ( )`

We have now used a counter variable (`nrOfEggsLaid`) to determine how often a while loop is executed.

**Challenge 4.11: Lay a trail of n eggs**

We want to teach Mimi to leave a trail of n (a certain number of) eggs. This is similar to `layTrailOf6Eggs ( )` (challenge 4.10), however, this time for any number of eggs, not just 6. Write a method called `void layTrailOfEggs ( int nrOfEggsToLay )` with which Mimi leaves a trail of `nrOfEggsToLay` eggs. In the end, she should be standing on her final egg. Add comments to your code and test your work. Don't forget to guard for unexpected input.

**Challenge 4.12: Lay a trail of n eggs, check for valid input**

We want to teach Mimi to leave a trail of n (a certain number of) eggs. This is similar to `layTrailOfnEggs ()` (challenge 4.11), however, this time for any number of eggs, not just 6. Additionally, you must guard for invalid input.

Add comments to your code and test your work. Check that your solution works no matter what direction Mimi is facing.

## Reflection

In this assignment you practiced using variables to store values. You also learned about loops and how methods use parameters. One of the most important steps in becoming good at anything is to evaluate and reflect on what you did and how it went:




### Result






I know my solution works because ...  
 I am proud of my solution because ...  
 I could improve my solution by ...

### Method

My approach was good because ...  
 What I could do better next time is ...

Fill the following table with smileys indicating how things went.

	I can do it
	I did it a bit but didn't fully get it
	I didn't get it at all

	I can use variables to store values.
	I can use a trace table to analyse code.
	I can apply plans for counting, repeating tasks and swapping values.
	I can control how often a loop is executed using either a counter or a conditional statement.
	I understand how to use methods that have parameters.

## Saving and Handing in

You have just finished the assignment. Save your work! You will need this for future assignments. In the Greenfoot menu at the top of the screen, select 'Scenario' and then 'Save'. You now have all the scenario components in one folder. The folder has the name you chose when you selected 'Save As ...'.

### Handing in

Hand in the following:

- Your code: The java file `MyDodo.java`;
- Flowcharts: paste (photo's of) your flowcharts in a Word document;
- The reflection sheet: complete and hand it in.