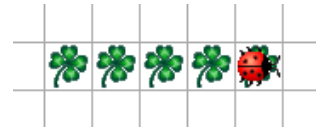


# Chapter 3: Variables

In the last chapter we have learned to repeat certain events while a condition is met.

Now we want to do the following:

*Kara is to lay a trail of five leaves.*



This would of course be quite easy if we simple call `putLeaf()` and `move()` five times. But this would not be very elegant. It would be nice if Kara counts how many leaves he has already placed. If so, Kara will need a “brain”, i.e. some kind of memory. Memory in programming can be used through the use of variables.

## Counting with Kara:

```
int i;  
i = 0;  
  
while (i < 5) {  
    putLeaf();  
    move();  
  
    i = i + 1;  
}
```

## Explanations:

- (a) With `int i;` we reserve space for a variable named `i` and the type `int`. We say: The variable `i` is **declared**. In Java, there are different types of variables that can be used (see tables below).
- (b) With `i = 0;` the variable `i` is assigned the value `0`. Since it is the first assignment for the variable, we also say: The variable `i` is **initialized**.
- (c) We can declare a variable and initialize it in one step:  
`int i = 0;`
- (d) For the condition `i < 5`, the comparison operator `<` means less than (more comparison operators, see tables below).
- (e) For the assignment `i = i + 1`, we must first look at the right part. It means: “Take the current value of `i`, add `1` to it and save the new value again under the name `i`.”

## More Information about Variables:

- (f) It is possible to provide a variable with a final value, i.e. to make it a constant:  
`final int NUMBER = 5;`  
Then we could write the above example `while (i < NUMBER)`  
Constants are written entirely in uppercase letters.
- (g) Variables that are not constants always start with lower case.

## Elementary Data Types in Java

These data types are called “elementary” because they are the basic data types in Java. (Later we will learn how to create variable types for entire objects).

### Integers and Characters

type	from	up to and including	memory required
<b>byte</b>	-128	127	8 bit
<b>short</b>	-32'768	32'767	16 bit
<b>int</b>	-2'147'483'648	2'147'483'647	32 bit
<b>long</b>	-9'223'372'036'854'775'808	9'223'372'036'854'775'807	64 bit
<b>char</b>	0	65'635	16 bit

### Floating Point Numbers

type	from	up to and including	memory required
<b>float</b>	$-3.4 \cdot 10^{38}$	$3.4 \cdot 10^{38}$	32 bit
<b>double</b>	$-1.7 \cdot 10^{308}$	$1.7 \cdot 10^{308}$	64 bit

### Logical Values

type	range of values	memory required
<b>boolean</b>	<b>true</b> or <b>false</b>	1 bit

## Comparison Operators

The following operators can be used for comparisons in Java. The result is always a **boolean** (either **true** or **false**).

operator	meaning	example
<b>&lt;</b>	Kleiner als	<code>k &lt; 12</code>
<b>&lt;=</b>	Kleiner als oder gleich	<code>k &lt;= 23</code>
<b>&gt;</b>	Grösser als	<code>k &gt; 67</code>
<b>&gt;=</b>	Grösser als oder gleich	<code>k &gt;= 45</code>
<b>==</b>	Gleich	<code>k == 2</code>
<b>!=</b>	Ungleich	<code>k != 32</code>

**Note:** The comparison to “equality” has always two equal signs **==**. A single equal sign **=** is used for assignments!

## Arithmetic Operations

To calculate we use the following arithmetic operators:

operator	meaning	example
<b>+</b>	Addition	<code>h = wert + 34</code>
<b>-</b>	Subtraktion	<code>z = 3.4 - t</code>
<b>*</b>	Multiplikation	<code>value = h * 3.56</code>
<b>/</b>	Division	<code>d = m / v</code>
<b>%</b>	Modulo (liefert den Rest der Division)	<code>count = w % 2</code>

**TASK 21: COUNTING LEAFS**

Kara is going horizontally from left to right up to the tree and counting leafs.

**Notes:** In the end, you can write the result with the following command to the 'console':

```
System.out.println("The result is: " + count);
```

Text must always be written in quotes. The plus sign is to append the value of the variable **count** (which may of course be named differently).

*Scope of variables: variables are visible only within the block (between the curly brackets), in which they are declared. They can also be declared outside the methods in order to be visible to the entire class.*

**TASK 22: KARA IN A BOX I**

A square area is surrounded by trees. Within the area is a set pattern of leafs, that Kara should invert. Kara starts in the upper left corner with a view to the right.



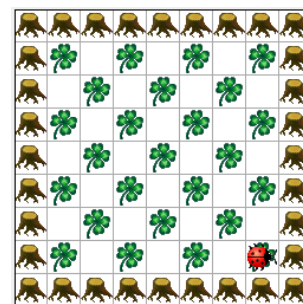
**Help:**

*In this task, it is helpful to work with boolean variables, e.g.:*

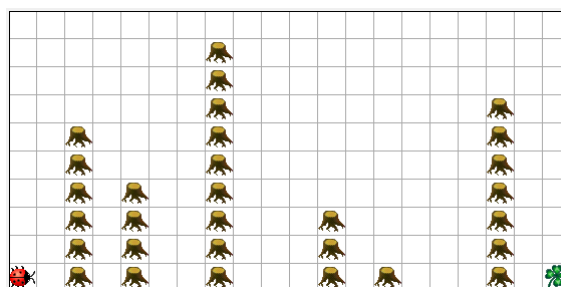
```
boolean goingRight = false;    // declaration and initialization
goingRight = !goingRight;     // switches to true from false and vice versa
if (goingRight)               // boolean as a condition
```

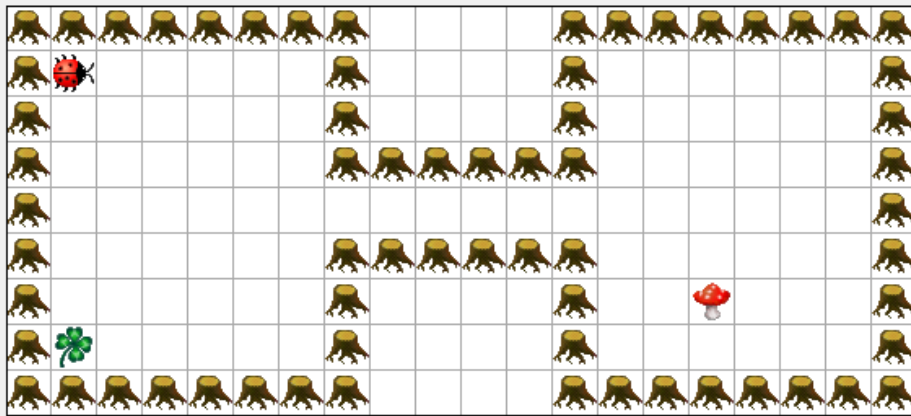
**ADDITIONAL TASK 23: KARA IN A BOX II**

A square area is surrounded by trees. Within the area is a checkerboard-like pattern of leafs to be laid by Kara. Kara starts in the upper left corner with a view to the right.

**ADDITIONAL TASK 24: THE LONGEST TREE LINE**

In this world there are several rows of trees. Kara is to determine the length (in number of trees) of the longest line of trees and output to the result to the console. Between the rows of trees is always at least one space. A leaf marks the end of the world.



**ADDITIONAL TASK 25 (VERY DIFFICULT): PUSH MUSHROOM THROUGH TUNNEL**

This world of Kara has two boxes connected by a tunnel. In the box on the left is Kara and a leaf. In the box on the right is a mushroom. Kara is to get to the other side, find the mushroom then push it to the other side. Once on the other side, the mushroom is to be pushed onto the leaf.

Kara always starts in the top left corner and the leaf is always in the bottom left corner. The Mushroom, on the other hand, can be at an arbitrary position on the right side of the tunnel.

Note: This problem may be solved in pairs and in collaboration. Then some sub-problems can be divided among each other:

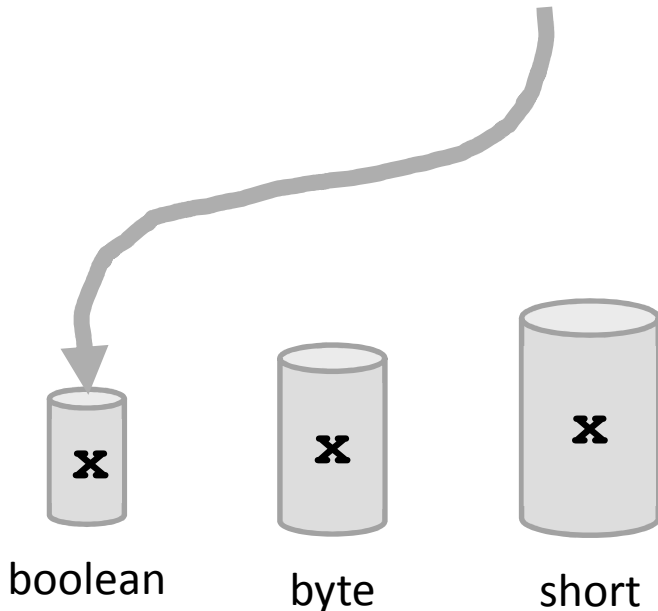
- Find the tunnel entrance
- Find Mushroom
- Place the Mushroom in the tunnel entrance
- Push the Mushroom on to the leaf

### More about Variables

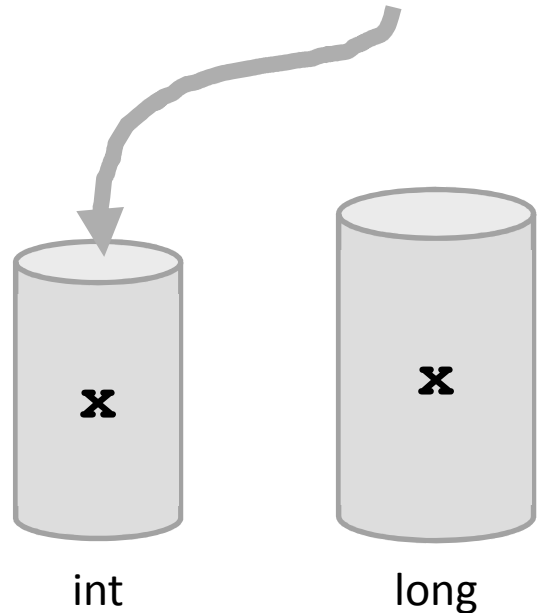
So far, we have had a quick introduction to variables. Now a few additional comments on the different types:

#### Elementary Data Types

```
boolean x = true;
```



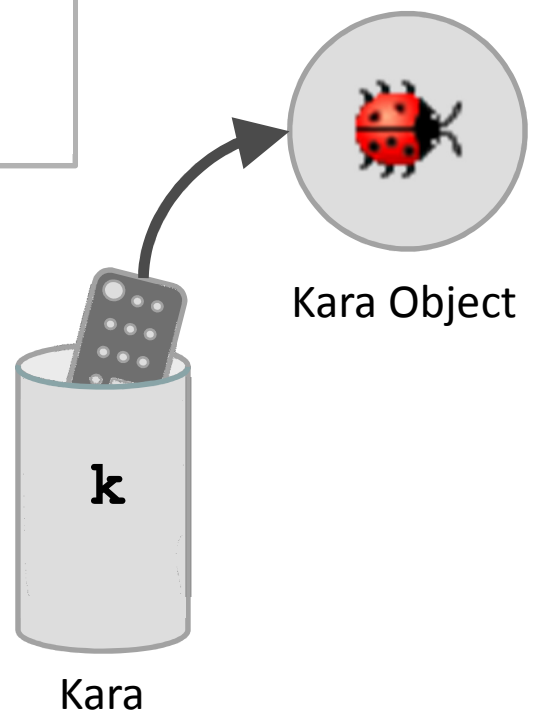
```
int x = 12;
```



#### Reference Types

```
Kara k = new Kara();  
k.move();
```

→ The value in `k` is a reference to the Kara-object. With the dot operator (`k.`) `k` can be used like a remote control for the Kara object!



**Note about Kara:**

- Ideas and concepts were developed by Jürg Nievergelt, Werner Hartmann, Raimond Reichert et al., <http://www.swisseduc.ch/informatik/karatojava/>, February 2011.
- Some Kara exercises are based on material by Horst Gierhardt, <http://www.swisseduc.ch/informatik/karatojava/javakara/material/>, February 2011.