. . . A SMALL GROUP OF MEN AND WOMEN, LED BY JOHN VON NEUMANN AT THE INSTITUTE FOR ADVANCED STUDY IN PRINCETON, NEW JERSEY, WHO BUILT ONE OF THE FIRST COMPUTERS TO REALIZE ALAN TURING'S VISION OF A UNIVERSAL MACHINE. THEIR WORK WOULD BREAK THE DISTINCTION BETWEEN NUMBERS THAT *MEAN* THINGS AND NUMBERS THAT *DO* THINGS - AND OUR UNIVERSE WOULD NEVER BE THE SAME.

DESCRIPTIVE TEXT, TO TURING'S CATHEDRAL BY GEORGE DYSON

. . . THERE'S NOTHING NECESSARILY PHYSICAL OR EXPENSIVE OR EVEN SLOW IN THE PROCESS OF PARADIGM CHANGE. IN A SINGLE INDIVIDUAL IT CAN HAPPEN IN A MILLISECOND. ALL IT TAKES IS A CLICK IN THE MIND, A FALLING OF SCALES FROM EYES, A NEW WAY OF SEEING.

DONELLA H. MEADOWS - LEVERAGE POINTS: PLACES TO INTERVENE IN A SYSTEM

THE PROCESS OF PREPARING PROGRAMS FOR A DIGITAL COMPUTER IS ESPECIALLY ATTRACTIVE, NOT ONLY BECAUSE IT CAN BE ECONOMICALLY AND SCIENTIFICALLY REWARDING, BUT ALSO BECAUSE IT CAN BE AN AESTHETHIC EXPERIENCE MUCH LIKE COMPOSING POETRY OR MUSIC.

DONALD E. KNUTH - THE ART OF COMPUTER PROGRAMMING - VOLUME 1 - FUNDAMENTAL ALGORITHMS

OLA DAHL

INTO PROGRAMMING

theintobooks.com

A Books with Views™ production

Typeset using Tufte LaTeX, tufte-latex.googlecode.com

Draft printing, February 9, 2017

Contents

List of Figures

List of Tables

Welcome

Computers have been around us for many years now. They are placed on our desks, inside our phones, and inside our cars and washing machines. Computers do their work by following *instructions*. These instructions are bundled together, as *computer programs*. The programs are then *executed*, which means that the instructions are followed, sometimes one by one and sometimes more than one at the same time, and the computer does what the instructions tells it to do.

The act of creating computer programs is referred to as *programming*, or if you wish to make it more clear, *computer programming*. This book has the goal of giving you an introduction to computer programming. It will do so by describing the basic elements of a *programming language*, and how these elements can be used to create programs.

Before learning how to do the actual programming, it is important to gain experience with the practical *usage* of computers. I will assume that you have such practical experience, and most likely you are using a computer, or perhaps another type of electronic reading device, right now, as you read this text.

If you read the book on the web, you are typically using a *browser program*, such as Firefox or Opera. If you use another reading device, such as a tablet computer, you may be using some kind of reader software, like a Kindle reader program or an iBooks program.

There are many types of computer programs, used in different scenarios, such as programs for word processing, program for making numerical calculations, programs for reading e-books, and programs for controlling the speed of your car or the temperature of your washing machine.

The programs described in this book will be small compared to a browser program, and they will also be small compared to a car control program, but they might still be useful for learning the important concepts.

The book describes how to write the program text, also referred to as *source code*. In order for a program to *run*, which is another

word for execute, the program must be represented in a format that is understandable by the computer. This format is referred to as *machine code*. This means that the source code must be translated into machine code.

Although the book concentrates on how to write the source code, there will be references given, to places where more information on how to make actual programs run on *your own computer* can be found. It is advisable to follow along, and to make your own programs, as you read the book.

This is a Book with Views[1]. This means that the book covers several programming languages at the same time. When you read the book, you read it one view at a time, and you can easily switch betwen the views. In the e-book version of the book there are links to the views at the end of each section, and in the web version of the book there are also links to the views in the left sidebar of each page. The view you are currently reading treats the programming language Java.

[1] http://bookswithviews.com

The book is written to give an introduction, and to give possibilities to practice writing programs. When doing this, you take on the role of a programmer[2]. Perhaps you will do it as a hobby, or perhaps you will do it as a profession. In any case I hope the book might be of help in the task of learning programming.

[2] http://en.wikipedia.org/wiki/Programmer

The remained of this chapter will give an introduction to programming, in the form of a simple example. In the following chapter Names and values, we will start looking at *variables*.

Say hello

It is customary to write a small program first, before making larger programs. A small program, with its only task being the display of a text message, saying "Hello, world", is often used. Such a program can be referred to as a Hello world program[3].

[3] http://en.wikipedia.org/wiki/Hello_world_program

A Hello world program written in

Java often starts with a *class definition*. A class can be thought of as an aggregation of instructions and data, put together for a specific purpose. The instructions inside a class are typically divided into groups, each group in turn having a specific purpose. Such groups are referred to as *methods*.

The class definition here starts with the line

```
class Hello
```

which gives the name `Hello` to the class. In this example, the class `Hello` will be the main class used in the program, and the only class created by the programmer. An already defined class, named `System`,

will also be used, for the purpose of getting access to a method called `println`. This method, which is used in the program as

```
System.out.println("Hello, world");
```

makes the string `Hello, world` appear on the screen of the computer where the program is run. The complete program is shown in Figure 1.

```
class Hello
{
    public static void main(String[] args)
    {
        System.out.println("Hello, world");
    }
}
```

Figure 1: A hello world program in Java.

The program in Figure 1 contains a method named *main*. The purpose of this method is to define a starting point, so that the operating system[4] knows where to start the program when it is executed.

[4] http://www.howstuffworks.com/operating-system.htm

How to make it run

The program in Figure 1 can be executed on a computer. I assume that you have a computer, perhaps a computer looking like the computer in Figure 2.

Figure 2: A computer, of a possible type that you may want to use for programming.

It may also be a computer in the form of a mobile phone, or some other device where a computer resides inside.

When you program using the language

Java you need a *compiler*. You also need a Java *virtual machine*. It is possible to obtain a compiler and a Java virtual machine by using the Java SE Development Kit[5], which is available for Windows, Linux, and for Mac OS.

[5] http://www.oracle.com/technetwork/java/javase/downloads/jdk-7u2-download-1377129.html

Assuming the program in Figure 1 is stored in a file named `Hello.java`, the program can be made to run using a Java compiler and a Java virtual machine. First, the program needs to be *compiled*. This can be done using the command

```
javac Hello.java
```

which takes the file `Hello.java`, and as a result creates a file named `Hello.class`.

The program can then be *executed*, by giving the command

```
java Hello
```

Acknowledgements

This book has been produced using several open source software products and open standards. It is written using XML, and translated, using Python[6] software, to html for the web version, epub for the e-book version, and LaTeX to be converted to pdf for the print version. The mobi version is currently created using Calibre[7]. Emacs and PSGML[8] are used for editing. The book contains images from the Open Clipart Library[9].

[6] http://python.org/

[7] http://calibre-ebook.com/

[8] http://sourceforge.net/projects/psgml/

[9] http://openclipart.org/

Names and values

Programs need to keep track of information when performing their tasks. A program computing your salary needs to keep track of your salary, your tax, and most likely also some information about your bank account. For this purpose, programs often use *variables*, which can be assigned *values*.

Variables can be viewed as *named storage places*, where values of different *types* can be stored. Among the different types you will find numerical values, such as the number π, but also strings of characters, like the string `"Hello, world"` that was used in the program in Figure 1.

Values

A value can be assigned directly to a variable, using an explicit value. Such an explicit value is called a *literal*. A character literal, in this case the character literal `b`, can be assigned to a variable with the name `a` using the assignment

```
char a = 'b';
```

As can be seen, the assigment is done by placing the character `b`, enclosed in single quotes, at the right hand side of an equals sign.

In this assignment, the word `char` is placed in front of the variable `a`. This means that the variable is being *declared*, to be of the type `char`. A variable of type `char` can hold values which are *characters*, such as the letters a to z, but also special signs, like semicolon (;), comma (,), and exclamation mark (!).

A variable can also be assigned a numerical value. There are two kinds of numerical values. One kind is referred to as *floating point* values, and the other kind is referred to as *integer values*. A floating point variable represents decimal numbers and an integer variable represents whole numbers. As an example, the decimal value for π, rounded by the programmer to 7 decimals, can be assigned to a variable named `pi`, using the assignment

```
pi = 3.1415927;
```

In the above assignment, the variable `pi` is assumed to be declared, before it is being assigned a value. The declararation is done, using the word *double* to indicate a floating-point variable with double precision[10], as

```
static double pi;
```

[10] http://en.wikipedia.org/wiki/Double-precision_floating-point_format

In this declaration, the word *static* is also used, in front of the the word `double`. The reason for this is that the class to which the variable `pi` belongs will never be *instantiated* in the program where it is being used. The concept of *class instantiation* can be thought of as creating a variable, referred to as an *object*, then being an *instance* of the class from which is created. The so created object contains variables for the data belonging to the class, and it also contains the methods defined as belonging to the class.

As an example of an assignment of an integer variable, a variable named `i` can be

assigned the value 5, and at the same time being declared as a variable of type *int*, using also the word `static` as above in the declaration of the variable `pi`, as

```
static int i = 5;
```

Variables of type `int` can hold values which are whole numbers. Positive numbers as well as negative numbers are allowed.

A complete program with variables, and with literal values being assigned to these variables, is shown in Figure 3.

```
class NamesAndValues
{
    static int i = 5;
    static double pi;

    public static void main(String [] args)
    {
        char a = 'b';
        pi = 3.1415927;
        System.out.println("i is " + i +
            " and pi is " + pi +
            " and a is " + a);
    }
}
```

Figure 3: A program with variables, and with assignments of values to these variables.

The program in Figure 3 contains the three variables described above, named `i`, `pi`, and `a`. The program also contains an instruction for printing the values of the variables.

The printing is done by a call to a method named *println*. This method belongs to a class called `out`, which in turn belongs to a class called `System`. Therefore, the names `System` and `out` are used in the method call, as

```
System.out.println("i is " + i +
    " and pi is " + pi +
    " and a is " + a);
```

In this call to the method `println`, strings of text are combined with the variables. The combination is done using the plus-sign, which connects the different parts into a new string. The so connected string appears as the argument to a call to `println`, and it is therefore printed when the program runs.

The program in Figure 3 can be run, as described in Section How to make it run. The output of the program is then shown, as

```
i is 5 and pi is 3.1415927 and a is b
```

Named storage places

The program in Figure 3 contains the three variables `i`, `pi`, and `a`. Each of these variables can hold a value. In this way, a variable can be viewed as a *storage place* for a value. Since a variable has a name, it can be viewed as a *named storage place*.

A variable also has a *type*. As an example, the variable `i` is an *integer variable*. The type of a variable determines the type of values that are allowed to be stored. Hence, the variable `i` can store integer values.

As demonstrated in the program in Figure 3, variables can be assigned literal values. Variables can also be assigned values that are the result of computations. These computations can be performed using other variables, as well as literal values.

As an example, consider a program for calculating the area and the circumference of a circle. As we have learned in school, the area of a circle is computed using the number π and the radius of the circle. Using the notation r for the radius and the notation a for the area, the formula reads

$$a = \pi r^2 \tag{1}$$

The circumference, here denoted c, of a circle is calculated using the circle diameter. Using the fact that the circle diamater is the radius multiplied by two, the formula for calculating the circumference reads

$$c = \pi \cdot 2r \tag{2}$$

Calculations corresponding to (1) and (2) can be done in a program, using a variable named `area` for the area `a`, a variable named `circ` for the circumference, and a variable named `radius` for the radius *r*.

In addition, a *constant* named *PI* can be used, for the purpose of representing π.

A constant can be defined using the keywords *static* and *final*. The constant `PI` can be defined, using these keywords together with the word `double` to indicate its type, as

```
static final double PI = 3.1415927;
```

The variable `area` is declared to be of type `double`, as

```
double area;
```

This is the case also for the variable `radius`, which is used, together with the constant `PI`, to calculate the area as

```
area = PI*radius*radius;
```

The variable `circ` is declared to be of type double, as

```
double circ;
```

It is assigned a value, as the result of the calculation of the circumference as

```
circ = PI*2*radius;
```

A complete program with variables and assignments is shown in Figure 4.

The program in Figure 4 contains the calculations for area and circumference, as shown above. It also contains statements for printing the results of the calculations.

The program in Figure 4 also contains a *string variable* named `color`.

The variable `color` is declared to be an object of the class `String`. It is declared, and initialised to the string `"blue"`, as

```
String color = "blue";
```

The program in Figure 4 can be compiled and executed. When running the program, its output becomes

```
the area is 28.2743343
the circumference is 18.8495562
the circle is blue
```

Some computations

Computer programs often perform computations. Here we consider some examples illustrating computations.

```
class MoreAssignments
{
    static final double PI = 3.1415927;

    public static void main(String [] args)
    {
        String color = "blue";
        int radius = 3;
        double area;
        double circ;
        area = PI*radius*radius;
        circ = PI*2*radius;
        System.out.println("the area is " + area);
        System.out.println("the circumference is " + circ);
        System.out.println("the circle is " + color);
    }
}
```

Figure 4: A program with variables and assignments.

A program can be used to calculate future values of an investment. Consider an example where an initial sum s is invested. It could then be of interest to calculate the future value of the investment. Assuming that the value is increased by r percent each year, the value after a number of years can be calculated.

If we do, as is common in mathematics-oriented books, denote the number of years by a variable, for example called `n`, the value of the investment, after `n` years, can be calculated as

$$s_n = s\left(1 + \frac{r}{100}\right)^n \tag{3}$$

We note that the formula (3) also applies to other situations, for example when calculating your salary after `n` years, given an annual raise of `r` percent.

In a program, designed to calculate a person's salary after a given number of years, we can use a variable

```
double salary;
```

to represent the salary. The raise r, divided by 100, could be represented by a variable, initalised to an annual raise of 5 percent, as

```
double raise = 0.05;
```

Using a variable

```
int n_years;
```

to represent the number of years, the salary after 5 years and after 10 years, starting at an intial salary of 3000, can be computed by first assigning values to the variables `salary` and `n_years`, and then performing calculations according to (3). Program code for performing these operations is shown in Figure 5.

```
salary = 3000;
n_years = 5;

salary = salary * Math.pow(1 + raise, n_years);
System.out.println("my salary in " + n_years +
    " years is " + salary);

salary = salary * Math.pow(1 + raise, n_years);
System.out.println("my salary in " + 2*n_years +
    " years is " + salary);
```

Figure 5: Calculation of salary after 5 and 10 years, given an intial salary and a yearly raise.

The program code in Figure 5 uses a method called `pow`, belonging to the class `Math`, for the purpose of calculating the exponent in (3). The class `Math` is part of the Java class library[11].

[11] http://docs.oracle.com/javase/7/docs/api/

The program code in Figure 5 can be placed in a program and executed. The result of running the code inside a program becomes

```
my salary in 5 years is 3828.844687500001
my salary in 10 years is 4886.683880332327
```

As another example of computations, we consider the problem of figuring out our own weight on the moon. We may never be able to go to the moon, but nevertheless it could be interesting to see what kind of weight loss such a journey would lead to.

Here we need some physics. We might consult the great Isaac Newton[12] for this purpose. It is stated by Newton, in the law of universal gravitation[13], that given the mass m_1 of myself, the mass m_2 of the planet I am standing on, together with the radius R of the planet, my weight can be calculated. In addition, a constant named G, having the value $6.674 \cdot 10^{-11}$, is needed. The resulting formula, as given by the law of universal gravitation, reads

[12] http://en.wikipedia.org/wiki/Isaac_Newton
[13] http://en.wikipedia.org/wiki/Newton%27s_law_of_universal_gravitation

$$F = G\frac{m_1 m_2}{R^2} \tag{4}$$

This formula shows how the weight, here referred to using a variable named F to indicate that it is actually a *force*, can be calculated.

It is now possible to calculate my weight on the moon. For this purpose I choose m_1 in (4) to by my own mass, expressed in kilograms, and m_2 to be the mass of the moon. In addition, I let R be the radius of the moon.

Information from Wikipedia about the moon[14] relates the radius of the moon and the mass of the moon to the corresponding quantities for the earth, which are available from Earth Wikipedia article[15]. The numerical values found in this way can be used, together with the formula (4) in a program as follows.

[14] http://en.wikipedia.org/wiki/Moon

[15] http://en.wikipedia.org/wiki/Earth

The gravity constant and the earth parameters are defined as

```
static final double G = 6.674e-11;

static final double R_EARTH = 6371000.0;
static final double M_EARTH = 5.9736e24;
```

The actual computation of the weight of a person with a mass of 77 kg, on the moon, can then be done, using variables `m_myself` for the mass of the person, `m_moon` and `r_moon` for the mass and radius of the moon, and `weight` for the resulting weight, as

```
m_myself = 77;
m_moon = 0.0123 * M_EARTH;
r_moon = 0.273 * R_EARTH;
weight = G * m_myself * m_moon /
    (r_moon * r_moon);
System.out.print("the weight of " + m_myself + " kg");
System.out.println(" on the moon is " + weight + " N");
```

where a printout of the result also is done. The result of executing the code gives the weight on the moon for a person with a mass of 77 kg, as

```
the weight of 77.0 kg on the moon is 124.78887740836468 N
```

A comparison with the weight on earth, calculated and printed using the code

```
weight = G * m_myself * M_EARTH /
    (R_EARTH * R_EARTH);
System.out.print("the weight of " + m_myself + " kg");
System.out.println(" on earth is " + weight + " N");
```

which when executed gives the result

```
the weight of 1.0 kg on earth is 9.81986088519482 N
the weight of 77.0 kg on earth is 756.1292881600011 N
```

shows that the weight on the moon is approximately one sixth of the weight on the earth.