Object-Oriented Programming with Java, part I ››

Authors: Arto Vihavainen, Matti Luukkainen
Translators to English: Emilia Hjelm, Alex H. Virtanen, Matti Luukkainen, Virpi Sumu, Birunthan Mohanathas

24. A QUICK RECAP

Let us start week 6 with two assignments that use the most important topics of week 5. You might want to read chapter 23.10 before assignment 94 and chapters 23.6 and 23.12 before assignment 95.

Exercise 94: PhoneBook

In this assignment we are implementing a simple phone book.

EXERCISE 94.1: PERSON

Start by programing the class `Person` which works as follows:

```
public static void main(String[] args) {
    Person pekka = new Person("Pekka Mikkola", "040-123123");

    System.out.println(pekka.getName());
    System.out.println(pekka.getNumber());

    System.out.println(pekka);
    pekka.changeNumber("050-333444");
    System.out.println(pekka);
}
```

The output is:

```
Pekka Mikkola
040-123123
Pekka Mikkola  number: 040-123123
Pekka Mikkola  number: 050-333444
```

So you have to implement the following class:

- the method `public String toString()`, which returns the string representation formulated as the above example shows
- constructor that sets the person name and phone number
- `public String getName()`, that returns the name
- `public String getNumber()`, that returns the phone number
- the method `public void changeNumber(String newNumber)`, that can be used to change the phone number of the person

EXERCISE 94.2: ADDING PERSONS TO PHONEBOOK

Program the class `Phonebook` that stores `Person`-objects using an ArrayList. At this stage you'll need the following methods:

- `public void add(String name, String number)` creates a `Person`-object and adds it to the ArrayList inside the Phonebook
- `public void printAll()`, prints all the persons inside the Phonebook

With the code:

```
public static void main(String[] args) {
    Phonebook phonebook = new Phonebook();

    phonebook.add("Pekka Mikkola", "040-123123");
    phonebook.add("Edsger Dijkstra", "045-456123");
    phonebook.add("Donald Knuth", "050-222333");

    phonebook.printAll();
}
```

the output should be:

```
Pekka Mikkola  number: 040-123123
Edsger Dijkstra  number: 045-456123
Donald Knuth  number: 050-222333
```

EXERCISE 94.3: SEARCHING FOR NUMBERS FROM THE PHONEBOOKS

Extend the class Phonebook with the method `public String searchNumber(String name)`, that returns the phone number corresponding to the given name. If the sought person is not known the string "number not known" is returned.

Example code:

```
public static void main(String[] args) {
    Phonebook phonebook = new Phonebook();
    phonebook.add("Pekka Mikkola", "040-123123");
    phonebook.add("Edsger Dijkstra", "045-456123");
    phonebook.add("Donald Knuth", "050-222333");

    String number = phonebook.searchNumber("Pekka Mikkola");
    System.out.println( number );

    number = phonebook.searchNumber("Martti Tienari");
    System.out.println( number );
}
```

output:

```
040-123123
number not known
```

Exercise 95: Money

In a previous assignment we stored the balance of a LyyraCard using a double variable. In reality money should not be represented as a double since the double arithmetics is not accurate. A better idea would be to implement a class that represents money. We'll start with the following class skeleton:

```
public class Money {

    private final int euros;
    private final int cents;

    public Money(int euros, int cents) {

        if (cents > 99) {
            euros += cents / 100;
            cents %= 100;
        }

        this.euros = euros;
        this.cents = cents;
    }

    public int euros(){
        return euros;
    }

    public int cents(){
        return cents;
    }

    public String toString() {
        String zero = "";
        if (cents <= 10) {
            zero = "0";
        }

        return euros + "." + zero + cents + "e";
    }
}
```

Notice that the instance variables `euros` and `cents` have been defined as `final` meaning that once the variables have been set, the value of those can not be changed. An object value of which can not be changed is said to be *immutable*. If we need to e.g. calculate

the sum of two money objects, we need to create a new money object that represents the sum of the originals.

In the following we'll create three methods that are needed in operating with money.

EXERCISE 95.1: PLUS

Let us start by implementing the method `public Money plus(Money added)`, that returns a *new Money object* that has a value equal to the sum of the object for which the method was called and the object given as parameter.

Examples of the method usage:

```
Money a = new Money(10,0);
Money b = new Money(5,0);

Money c = a.plus(b);

System.out.println(a);  // 10.00e
System.out.println(b);  // 5.00e
System.out.println(c);  // 15.00e

a = a.plus(c);          // NOTE: new Money-object is created and reference to tha
                        //           is assigned to variable a.
                        //       The Money object 10.00e that variable a used to
                        //           is not referenced anymore

System.out.println(a);  // 25.00e
System.out.println(b);  // 5.00e
System.out.println(c);  // 15.00e
```

EXERCISE 95.2: LESS

Create the method `public boolean less(Money compared)`, that returns true if the object for which the method was called is less valuable than the object given as parameter.

```
Money a = new Money(10,0);
Money b = new Money(3,0);
Money c = new Money(5,0);

System.out.println(a.less(b));  // false
System.out.println(b.less(c));  // true
```

EXERCISE 95.3: MINUS

And finally create the method `public Money minus(Money decremented)`, that returns a *new Money object* that has a value equal to the object for which the method was called minus

the object given as parameter. If the value would be negative, the resulting Money object should have the value 0.

Examples of the method usage:

```
Money a = new Money(10,0);
Money b = new Money(3,50);

Money c = a.minus(b);

System.out.println(a);  // 10.00e
System.out.println(b);  // 3.50e
System.out.println(c);  // 6.50e

c = c.minus(a);         // NOTE: new Money-object is created and reference to tha
                        //       the Money object 6.50e that variable c used to l

System.out.println(a);  // 10.00e
System.out.println(b);  // 3.50e
System.out.println(c);  // 0.00e
```

24.1 CHARACTER STRINGS ARE IMMUTABLE

The String objects of Java, as with the Money class objects, are unchangeable, *immutable*. If for example a new object is concatenated to the end of a character string with the + operator, the original character string doesn't become longer, but a new character string object is born:

```
String characterString = "test";
characterString + "tail";

System.out.println( characterString );  // test
```

We see that the character string cannot be changed, but we can add the value of the new character string - that was born from concatenation - to the old variable:

```
String characterString = "test";
characterString = characterString + "tail";    // or characterString += "tail";

System.out.println( characterString );  // testtail
```

Now the variable `characterString` refers to a *new* character string object, which was created by combining the previous character string value the variable referred to ("test") with the "tail" character string. Nothing refers to the "test" character string object anymore.

25. ARRAY

During the course, we've used ArrayLists numerous times to store different kinds of objects. ArrayList is easy to use because it offers a lot of ready-made tools that make the programmer's life a little easier: automatic growing of a list, thanks to the list which doesn't run out of space (unless of course the list grows so large that it makes the program take up all the memory that is reserved for it), for example.

Array is an object that can be understood as a series of *pigeonholes* for values. The *length* or *size* of an array is the number of spots in that array - the number of items you can put in the array. The values of an array are called *cells* of the array. Unlike with ArrayLists, the size of the array (the amount of cells in an array) cannot be changed, growing an array always requires creating a new array and copying the cells of the old array to the new one.

An array can be created in two ways. Let's take a look at the way in which we give content to the array at creation. An array of the integer type that consists of 3 cells is defined as follows:

```
int[] numbers = {100, 1, 42};
```

The type of the Array object is denoted as `int[]`, which stands for an array, the cells of which are of the type `int`. In the example the name of the array-object is `numbers` and it holds 3 number values `{100, 1, 42}`. The array is formatted with a block, in which the values to be inserted into the array are separated by commas.

The values of the array can be of any variable type that we've seen earlier. Below we've first introduced an array containing character strings and then an array containing floating numbers.

```
String[] characterStringArray = {"Matti P.", "Matti V."};
double[] floatingNumberArray = {1.20, 3.14, 100.0, 0.6666666667};
```

The cells of the array are referred to with *indexes* that are integers. The index tells the position of the cell in the array. The first item in an array is in position 0, the next one in position 1, and so forth. When inspecting a certain value of an array, the index is given after the name of the array object in brackets.

```
// index            0   1    2    3   4   5     6     7
int[] numbers = {100,  1,  42,  23,  1,  1, 3200, 3201};

System.out.println(numbers[0]);    // prints the number in the array's index 0: the
System.out.println(numbers[2]);    // prints the number in the array's index 2, the
```

The size (length) of the array above is 8.

You'll probably notice that the `get`-method of ArrayList works pretty much the same as getting from a certain index of an array. Only the notation - the syntax - is different when dealing with arrays.

Setting an individual value to a certain position in an array happens the same way as with regular variables, only with arrays the index also has to be mentioned. The index is mentioned inside brackets.

```
int[] numbers = {100,1,42};

numbers[0] = 1;     // setting value 1 to index 0
numbers[1] = 101;   // setting value 101 to index 1

// the numbers array now looks like {1,101,42}
```

If an index points *past an array*, that is, to a cell that doesn't exist, we will get an error: *ArrayIndexOutOfBoundsException*, which means that the index that we pointed at doesn't exist. So we cannot refer to a cell that is past the array - to an index that is smaller than 0, or larger or equals the size of the array.

We'll notice that the array clearly is related to ArrayList. Arrays, as with lists, have their cells in a certain order!

25.1 ITERATION OF AN ARRAY

The size of an array object can be found out by typing `array.length` into the code, notice that you don't use parentheses with this one. `array.length()` does not work!

Iterating through the cells of an array is easy to implement with the help of the `while`-command:

```
int[] numbers = {1, 8, 10, 3, 5};

int i = 0;
while (i < numbers.length ) {
    System.out.println(numbers[i]);
    i++;
}
```

With the help of variable `i` we go through the indexes 0, 1, 2, 3, and 4, and print the value of the variable in each cell. First `numbers[0]` gets printed, then `numbers[1]` and so forth. The variable `i` stops getting increased when the array has been iterated through, that is when `i`'s value is equal to the length of the array.

When iterating through an array it isn't always necessary to list the indexes of it, the only interesting thing is the values of the array. In this case we can use the for-each-structure - that we became familiar with earlier - to go through the values. Now only the name of a variable is given in the frame of the loop, to which each of the values of the array are set one after the other. The name of the array is separated with a colon.

```
int[] numbers = {1,8,10,3,5};

for (int number : numbers) {
    System.out.println(number);
}
```

```
String[] names = {"Juhana L.", "Matti P.", "Matti L.", "Pekka M."};

for (String name : names) {
    System.out.println(name);
}
```

Notice: when using a for-each-type of loop you cannot set values to the cells of the array! With the format of the for-sentence we inspect next that can be done too.

25.2 ANOTHER FORM OF THE FOR COMMAND

So far when doing loops, we've used while and the for-each form of the for sentence. Another form of the for-loop exists, which is handy especially when handling arrays. In the following we print the numbers 0, 1 and 2 with a for loop:

```
for (int i = 0; i < 3; i++ ) {
    System.out.println(i);
}
```

The for in the example works *exactly* as the while below:

```
int i = 0;  // formatting the variable that will be used in the loop
while ( i < 3 ) {  // condition
    System.out.println(i);
    i++;   // updating the variable that is used in the loop
}
```

a for command, as shown in `for (int i = 0; i < 3; i++ )` above, has three parts to it: *formatting the loop variables; condition; updating the loop variables*:

- In the first part, the variables that are used in the loop are formatted. In the example above we formatted the variable `i` with `int i=0`. The first part is run only once, at the beginning of a for run.
- In the second part the condition is defined, which defines how long the code is run in the code block that is related to the for loop. In our example the condition was `i < 3`. The validity of the condition is checked *before each round of the loop*. The condition works exactly the same as the a condition of a while loop works.
- The third part, which in our example is `i++` , is always run once at the end of each round of the loop.

Compared to *while*, *for* is a slightly clearer way of implementing loops of whose amount of runs is based on, for example, growing a counter. When going through an array the case is usually exactly this. In the following we print the contents of the `numbers` array with for:

```
int[] numbers = {1, 3, 5, 9, 17, 31, 57, 105};

for(int i = 3; i < 7; i++) {
```

```
    System.out.println(numbers[i]);
}
```

Naturally with for you don't have to start from 0 and the iteration can be done 'from top down'. For example, the cells in indexes 6, 5, 4, and 3 can be printed like this:

```
int[] numbers = {1, 3, 5, 9, 17, 31, 57, 105};

for(int i = 6; i>2 ; i--) {
    System.out.println(numbers[i]);
}
```

25.3 FOR AND ARRAY LENGTH

Going through all cells of an array with for happens like this:

```
int[] numbers = {1, 8, 10, 3, 5};

for (int i = 0; i < numbers.length; i++ ) {
    System.out.println(numbers[i]);
}
```

Notice, that in the condition `i < numbers.length` we compare the value of the loop variable to the length we get from the array. The condition should not in any case be "hardcoded" as, for example, `i < 5` because often the length of the array can't be known for sure beforehand.

25.4 ARRAY AS A PARAMETER

Arrays can be used - just as any other objects - as a parameters to a method. Notice that, as with all objects, the method gets a reference to an array, so all changes done to the content of the array in the method also show up in the main program.

```
public static void listCells(int[] integerArray) {

    System.out.println("the cells of the array are: ");
    for( int number : integerArray) {
        System.out.print(number + " ");
    }

    System.out.println("");
}

public static void  main(String[] args) {
    int[] numbers = { 1, 2, 3, 4, 5 };
```

```
    listCells(numbers);
}
```

As we already know, the name of the parameter within a method can be freely chosen. The name does not need to be the same as in the one used in calling it. Above, the array is called `integerArray` within the method and the caller of the method knows the array as `numbers`.

Exercise 96: Sum of the array

Implement the method `public static sum(int[] array)`, which returns the sum of the numbers in the array given as parameter.

Program skeleton:

```
public class Main {
    public static void main(String[] args) {
        int[] array = {5, 1, 3, 4, 2};
        System.out.println(sum(array));
    }

    public static int sum(int[] array) {
        // write code here
        return 0;
    }
}
```

The output should be:

```
15
```

NOTE: in this and some of the following assignments methods are `static` as the they used to be in the assignments for weeks 2 and 3. The reason for this is that the methods are not instance methods, i.e. not operating with instance variables of objects, instead they are working at "class level" and operating just with the values and objects given as parameter. In chapter 31 we'll elaborate more on the question whether a method should be static or not.

Exercise 97: Elegant printing of an array

Implement the method `public static int printElegantly(int[] array)`, which prints the numbers in the array on the same row. In the printout all the numbers should be separated with comma and whitespace and there should not be a comma trailing the last number.

Program skeleton:

```
public class Main {
    public static void main(String[] args) {
        int[] array = {5, 1, 3, 4, 2};
        printElegantly(array);
    }

    public static void printElegantly(int[] array) {
        // write code here
    }
}
```

The output should be:

```
5, 1, 3, 4, 2
```

25.5 CREATING A NEW ARRAY

If the size of the array isn't always the same, that is, if its size depends on user input for example, the previously introduced way of creating arrays will not do. It is also possible to create a table so that its size is defined with the help of a variable:

```
int cells = 99;
int[] array = new int[cells];
```

Above we create an array of the type int, that has 99 cells. With this alternative way creation of an array happens just like with any other object; with the command `new`. Following the `new` is the type of the array and in the brackets is the size of the array.

```
int cells = 99;
int[] array = new int[cells]; //creating an array of the size of the value in the '

if(array.length == cells) {
    System.out.println("The length of the array is " + cells);
} else {
    System.out.println("Something unreal happened. The length of the array is something else than
}
```

In the following example there is a program that prompts for the user the amount of values and subsequently the values. After this the program prints the values in the same order again. The values given by the user are stored in the array.

```
System.out.print("How many values? ");
int amountOfValues = Integer.parseInt(reader.nextLine());

int[] values = new int[amountOfValues];

System.out.println("Enter values:");
for(int i = 0; i < amountOfValues; i++) {
    values[i] = Integer.parseInt(reader.nextLine());
}

System.out.println("Values again:");
for(int i = 0; i < amountOfValues; i++) {
    System.out.println(values[i]);
}
```

A run of the program could look something like this:

```
How many values? 4
Enter values:
4
8
2
1
Values again:
4
2
8
1
```

25.6 AN ARRAY AS THE RETURN VALUE

Since methods can return objects, they can also return arrays. This particular method that returns an array looks like this -- notice that arrays might as well contain objects.

```
public static String[] giveStringTable() {
    String[] tchrs = new String[3];

    tchrs[0] = "Bonus";
    tchrs[1] = "Ihq";
    tchrs[2] = "Lennon";

    return tchrs;
}

public static void main(String[] args){
    String[] teachers = giveStringTable();

    for ( String teacher : teachers)
```

```
        System.out.println( teacher );
}
```

Exercise 98: Reversing and copying of an array

EXERCISE 98.1: COPY

Implement the method `public static int[] copy(int[] array)` that creates a copy of the parameter. **Tip:** since you are supposed to create a copy of the parameter, the method should create a new array where the contents of the parameter is copied.

In the following an example of the usage (note how code uses a handy helper method to print arrays):

```
public static void main(String[] args) {
    int[] original = {1, 2, 3, 4};
    int[] copied = copy(original);

    // change the copied
    copied[0] = 99;

    // print both
    System.out.println( "original: " + Arrays.toString(original));
    System.out.println( "copied: " + Arrays.toString(copied));
}
```

As seen in the output, the change made to the copy does not affect the original:

```
original: [1, 2, 3, 4]
copied: [99, 2, 3, 4]
```

EXERCISE 98.2: REVERSE COPY

Implement the method `public static int[] reverseCopy(int[] array)` that creates an array which contains the elements of the parameter but in reversed order. The parameter array must remain the same.

E.g. if the parameter contains values *5*, *6*, *7* the method returns *a new array* that contains the values *7*, *6*, *5*.

In the following an example of the usage:

```
public static void main(String[] args) {
    int[] original = {1, 2, 3, 4};
    int[] reverse = reverseCopy(original);
```

```
    // print both
    System.out.println( "original: " +Arrays.toString(original));
    System.out.println( "reversed: " +Arrays.toString(reverse));
}
```

The output should reveal that the parameter remains intact:

```
original: [1, 2, 3, 4]
reversed: [4, 3, 2, 1]
```

26. ABOUT BLOCKS AND NESTED LOOPS

A piece of code that begins with a curly bracket `{` and ends with a curly bracket `}` is called a *block*. As we've already seen, blocks are used - among other things - to denote the code of conditional and loop sentences. *An important feature of a block is that variables defined within it only exist within it.*.

In the following example we define the string variable `stringDefinedWithinBlock` within the block of a conditional sentence, which therefor will only exist within the block. The variable introduced within the block cannot be printed outside of it!

```
int number = 5;

if( number == 5 ){
    String stringDefinedWithinBlock = "Yeah!";
}

System.out.println(stringDefinedWithinBlock); // does not work!
```

However, you can use and manipulate variables defined outside of the block in the block.

```
int number = 5;

if( number == 5 ) {
    number = 6;
}

System.out.println(number); // prints 6
```

You can have any kind of code within a block. For example, a for loop can have another for loop within it or say, a while loop. Let's inspect the following program:

```
for(int i = 0; i < 3; i++) {
   System.out.print(i + ": ");

   for(int j = 0; j < 3; j++) {
      System.out.print(j + " ");
   }

   System.out.println();
}
```

The program prints the following:

```
0: 0 1 2
1: 0 1 2
2: 0 1 2
```

So what happens in the program? If we only think about the outer for loop, its functionality is easy to understand:

```
for(int i = 0; i < 3; i++) {
   System.out.print(i + ": ");

   // the inner for-loop

   System.out.println();
}
```

So first `i=0` prints `0:` and a carriage return. After this i grows and 1 is printed and so forth, so the outer for makes this happen:

```
0:
1:
2:
```

The inner for loop is also easy to understand separately. It prints out `0 1 2`. When we combine these two, we'll notice that the inner for loop carries out its print just before the outer for loop's carriage return.

26.1 VARIABLES DEFINED OUTSIDE OF A FOR LOOP AS ITS CONDITION

Let's inspect the following alteration to the previous example:

```
for(int i = 0; i < 3; i++) {
    System.out.print(i + ": ");

    for(int j = 0; j <= i; j++) {
        System.out.print(j + " ");
    }

    System.out.println();
}
```

The amount of runs the inner for loop does now depends on the value of the variable `i` of the outer loop. So when `i=0` the inner loop prints 0, when `i=1` the inner loop prints 0 1. The entire output of the program is as follows:

```
0: 0
1: 0 1
2: 0 1 2
```

The following program prints out the multiplication tables of the numbers 1 .. 10.

```
for(int i = 1; i <= 10; i++) {

    for(int j = 1; j <= 10; j++) {
        System.out.print(i * j + " ");
    }

    System.out.println();
}
```

The output looks like this:

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

The topmost row has the multiplication table of 1. At the beginning `i=1` and the inner loop's variable `j` gets the values 1...10. For each `i, j` value pair their product is printed. So at the beginning `i=1, j=1`, then `i=1, j=2`, ..., `i=1, j=10` next `i=2, j=1`, and so forth.

Of course the multiplication table program can be cut in to smaller pieces, too. We can define the methods `public void printMultiplicationTableRow(int multiplier, int howManyTimes)` and `public void printMultiplicationTable(int upTo)`, in this case the structure of our program could be as follows:

```
public class MultiplicationTable {

    public void print(int upTo) {
        for(int i = 1; i <= upTo; i++) {
            printMultiplicationTableRow(i, upTo);

            System.out.println();
        }
    }

    public void printMultiplicationTableRow(int multiplier, int howManyTimes) {
        for(int i = 1; j <= howManyTimes; i++) {
            System.out.print(i * multiplier + " ");
        }
    }
}
```

Now calling `new MultiplicationTable().print(5);` prints the tables below.

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25
```

Exercise 99: Array to stars

Implement the method `public static printArrayAsStars(int[] array)`, which prints a line with stars for each number in the array. The line length is determined by the number.

The program skeleton:

```
public class Main {
    public static void main(String[] args) {
        int[] array = {5, 1, 3, 4, 2};
        printArrayAsStars(array);
    }

    public static void printArrayAsStars(int[] array) {
        // code here
    }
}
```

The above example should cause the following output:

```
*****
*
***
****
**
```

As seen the first line has 5 stars and the reason for that is that is that the first element of the array is 5. The next line has one star since the second element of the array is 1, etc.

Exercise 100: Night sky

Let us implement a program that prints the Night sky. The sky has a star density. If the density is e.g. `0.1`, roughly 10% of the sky is covered with stars.

Stars print out as `*`-characters. Below an example that demonstrates how the `NightSky` could be used when all the steps of the assignment are done.

```
NightSky NightSky = new NightSky(0.1, 40, 10);
NightSky.print();
System.out.println("Number of stars: " + NightSky.starsInLastPrint());
System.out.println("");

NightSky = new NightSky(0.2, 15, 6);
NightSky.print();
System.out.println("Number of stars: " + NightSky.starsInLastPrint());
```

```
         *      *                     *
     *               * *           *       **
                                              *
          *         *        *           *   *
  *      *                           *
*               * *                         *
*   * *             *            * *   **
                                    *   *
            *                 *
      *                                   *
Number of stars: 36

 * * *      *
      * *    *
*      *
    *   *        *
*         *    * *
* ** **       *
Number of stars: 22
```

Note! in the assignment use the `for`-clause. Despite that the previous chapter described nested loops, in this assignment we "hide" the nested loop within a method.

EXERCISE 100.1: CLASS NIGHTSKY AND A STAR LINE

Create the class `NightSky`, that has three object variables: density (`double`), width (`int`), and height (`int`). The class should have 3 constructors:

- `public NightSky(double density)` creates a NightSky object with the given star density. Width gets the value `20` and height the value `10`.
- `public NightSky(int width, int height)` creates a NightSky object with the given width and height. Density gets the value `0.1`.
- `public NightSky(double density, int width, int height)` creates a NightSky-object with the given density, width and height

Add to the class `NightSky` the method `printLine`, that prints one line of starts. The line length is determined by the value of the instance variable `width` and the instance variable `density` determines the star probability. For each printed character you should use a `Random` object to decide if it prints out as a white space or a star. The method `nextDouble` will probably be of use now.

In the following example:

```
NightSky NightSky = new NightSky(0.1, 40, 10);
NightSky.printLine();
```

```
          *  *                  *
```

EXERCISE 100.2: PRINTING THE NIGHT SKY

Add the class `NightSky` the method `print`, that prints the night sky of the given size. Use the method `printLine` to print each separate line of the night sky. An example in the following:

```
NightSky NightSky = new NightSky(8, 4);
NightSky.print();
```

```
   *

 *
   *
```

EXERCISE 100.3: COUNTING THE NUMBER OF STARS

Add the class `NightSky` an instance variable starsInLastPrint (`int`) and the method `starsInLastPrint()`, that returns the number of stars printed in the previous night sky. Example in the below:

```
NightSky NightSky = new NightSky(8, 4);
NightSky.print();
System.out.println("Number of stars: " + NightSky.starsInLastPrint());
System.out.println("");

NightSky.print();
System.out.println("Number of stars: " + NightSky.starsInLastPrint());
```

```
 *

Number of stars: 1

 *
      *
*

Number of stars: 3
```

27. TO STATIC OR NOT TO STATIC?

When we started using objects, the material advised to leave out the keyword 'static' when defining their methods. However, up until week 3 all of the methods included that keyword. So what is it all about?

The following example has a method `resetArray`, that works as its name implies; it sets all of the cells of an array that it receives as a parameter to 0.

```
public class Program {

    public static void resetArray(int[] table) {
        for ( int i=0; i < table.length; i++ )
            table[i] = 0;
    }
```

```
    public static void main(String[] args) {
        int[] values = { 1, 2, 3, 4, 5 };

        for ( int number : values ) {
            System.out.print( number + " " );  // prints 1, 2, 3, 4, 5
        }

        System.out.println();

        resetArray(values);

        for ( int number : values ) {
            System.out.print( number + " " );  // prints 0, 0, 0, 0, 0
        }
    }
}
```

We notice that the method definition now has the keyword `static`. The reason for that is that the method does not operate on any object, instead it is a *class method* or in other words *static methods*. In contrast to instance methods, static methods are not connected to any particular object and thus the reference `this` is not valid within static methods. A static method can operate only with data that is given it as parameter. The parameter of a static method can naturally be an object.

Since static methods are not connected to any object, those can not be called through the object name: `objectName.methodName()` but should be called as in the above example by using only the method name.

If the static method is called from a different class, the call is of the form `ClassName.staticMethodName()`. The below example demonstrates that:

```
public class Program {
    public static void main(String[] args) {
        int[] values = { 1, 2, 3, 4, 5 };

        for ( int value : values ) {
            System.out.print( value + " " );  // prints: 1, 2, 3, 4, 5
        }

        System.out.println();

        ArrayHandling.resetArray(values);

        for ( int value : values ) {
            System.out.print( value + " " );  // prints: 0, 0, 0, 0, 0
        }
    }
}
```

```
public class ArrayHandling {
    public static void resetArray(int[] array) {
```

```
        for ( int i=0; i < array.length; i++ ) {
            array[i] = 0;
        }
    }
}
```

The static method that has been defined within another class will now be called with `ArrayHandling.resetArray(parameter);`.

27.1 WHEN STATIC METHODS SHOULD BE USED

All object state-handling methods should be defined as normal object methods. For example, all of the methods of the `Person, MyDate, Clock, Team, ...` classes we defined during the previous weeks should be defined as normal object methods, not as statics.

Lets get back to the `Person` class yet again. In the following is a part of the class definition. All of the object variables are referred to with the `this` keyword because we emphasize that we are handling the object variables 'within' the said object..

```
public class Person {
    private String name;
    private int age;

    public Person(String name) {
        this.age = 0;
        this.name = name;
    }

    public boolean isAdult(){
        if ( this.age < 18 ) {
            return false;
        }

        return true;
    }

    public void becomeOlder() {
        this.age++;
    }

    public String getName() {
        return this.name;
    }
}
```

Because the methods manipulate the object, they do not need to be defined as static, or in other words "not belonging to the object". If we try to do this, the program won't work:

```
public class Person {
    //...

    public static void becomeOlder() {
        this.age++;
    }
}
```

As a result we'll get an error *non-static variable age can not be referenced from static context*, which means that a static method cannot handle an object method.

So when should a static method be used then? Let us inspect the Person object handling an example familiar from chapter 23:

```
public class Program {
    public static void main(String[] args) {
        Person pekka = new Person("Pekka");
        Person antti = new Person("Antti");
        Person juhana = new Person("Juhana");

        for ( int i=0; i < 30; i++ ) {
            pekka.becomeOlder();
            juhana.becomeOlder();
        }

        antti.becomeOlder();

        if ( antti.isAdult() ) {
            System.out.println( antti.getName() + " is an adult" );
        } else {
            System.out.println( antti.getName() + " is a minor" );
        }

        if ( pekka.isAdult() ) {
            System.out.println( pekka.getName() + " is an adult" );
        } else {
            System.out.println( pekka.getName() + " is a minor" );
        }

        if ( juhana.isAdult() ) {
            System.out.println( juhana.getName() + " is an adult" );
        } else {
            System.out.println( juhana.getName() + " is a minor" );
        }
    }
}
```

We'll notice that the piece of code that reports the matureness of persons is copy-pasted twice in the program. It looks really bad!

Reporting the maturity of a person is an excellent candidate for a static method. Let's rewrite the Program using that method:

```
public class Main {

    public static void main(String[] args) {
        Person pekka = new Person("Pekka");
        Person antti = new Person("Antti");
        Person juhana = new Person("Juhana");

        for ( int i=0; i < 30; i++ ) {
            pekka.becomeOlder();
            juhana.becomeOlder();
        }

        antti.becomeOlder();

        reportMaturity(antti);

        reportMaturity(pekka);

        reportMaturity(juhana);
    }

    private static void reportMaturity(Person person) {
        if ( person.isAdult() ) {
            System.out.println(person.getName() + " is an adult");
        } else {
            System.out.println(person.getName() + " is a minor");
        }
    }
}
```

The method `reportMaturity` is defined as static so it doesn't belong to any object, **but** the method receives a Person object as a parameter. The method is not defined within the Person-class since even though it handles a Person object that it receives as a parameter, it is an assistance method of the main program we just wrote. With the method we've made main more readable.

Exercise 101: The library information system

In this assignment we are implementing a simple information system prototype for a library. The prototype will have functionality for searching books by the title, publisher or publishing year.

The main building blocks of the system are the classes `Book` and `Library`. Objects of the class `Book` represent the information of a single book. Object of the class `Library` holds a set of books and provides various ways to search for the books within the library.

EXERCISE 101.1: BOOK

Let us start with the class Book. The class has instance variables `title` for the book title, `publisher` for the name of the publisher, and `year` for the publishing year. The title and the

publisher are of the type String and the publishing year is represented as an integer.

Now implement the class `Book`. The class should have the constructor `public Book(String title, String publisher, int year)` and methods `public String title()`, `public String publisher()`, `public int year()` and `public String toString()`.

Example usage:

```
Book cheese = new Book("Cheese Problems Solved", "Woodhead Publishing", 2007);
System.out.println(cheese.title());
System.out.println(cheese.publisher());
System.out.println(cheese.year());

System.out.println(cheese);
```

The output should be:

```
Cheese Problems Solved
Woodhead Publishing
2007
Cheese Problems Solved, Woodhead Publishing, 2007
```

EXERCISE 101.2: LIBRARY

Implement the class `Library`, with constructor `public Library()` and methods `public void addBook(Book newBook)` and `public void printBooks()`

Example usage below.

```
Library Library = new Library();

Book cheese = new Book("Cheese Problems Solved", "Woodhead Publishing", 2007);
Library.addBook(cheese);

Book nhl = new Book("NHL Hockey", "Stanley Kupp", 1952);
Library.addBook(nhl);

Library.addBook(new Book("Battle Axes", "Tom A. Hawk", 1851));

Library.printBooks();
```

The output should be:

```
Cheese Problems Solved, Woodhead Publishing, 2007
NHL Hockey, Stanley Kupp, 1952
Battle Axes, Tom A. Hawk, 1851
```

EXERCISE 101.3: SEARCH FUNCTIONALITY

Add to the class `Library` the methods `public ArrayList<Book> searchByTitle(String title)`, `public ArrayList<Book> searchByPublisher(String publisher)` and `public ArrayList<Book> searchByYear(int year)`. The methods return the list of books that match the given title, publisher or year.

Note: you are supposed to do a method that returns an ArrayList. Use the following skeleton as starting point:

```
public class Library {
   // ...

   public ArrayList<Book> searchByTitle(String title) {
     ArrayList<Book> found = new ArrayList<Book>();

     // iterate the list of books and add all the matching books to the list fou

     return found;
   }
```

Note: when you do the search by a string (title or publisher), do not look for exact matches (with the method `equals`) instead use the method `contains` of the class `String`.

Example usage:

```
Library Library = new Library();

Library.addBook(new Book("Cheese Problems Solved", "Woodhead Publishing", 2007));
Library.addBook(new Book("The Stinky Cheese Man and Other Fairly Stupid Tales", "Penguin Grou
Library.addBook(new Book("NHL Hockey", "Stanley Kupp", 1952));
Library.addBook(new Book("Battle Axes", "Tom A. Hawk", 1851));

ArrayList<Book> result = Library.searchByTitle("Cheese");
for (Book book: result) {
    System.out.println(book);
}

System.out.println("---");
for (Book book: Library.searchByPublisher("Penguin Group  ")) {
    System.out.println(book);
}

System.out.println("---");
for (Book book: Library.searchByYear(1851)) {
    System.out.println(book);
}
```

The output should be:

```
Cheese Problems Solved, Woodhead Publishing, 2007
The Stinky Cheese Man and Other Fairly Stupid Tales, Penguin Group, 1992
---
---
Battle Axes, Tom A. Hawk, 1851
```

EXERCISE 101.4: IMPROVED SEARCH

There are some minor problems with the implemented search functionality. One particular problem is that the search differentiates upper and lower case letters. In the above example the search by title with the search term `"cheese"` produced an empty list as answer. The example where the search term contained extra white spaces did not give the expected answer, either. We'd like the search functionality to be case insensitive and not disturbed by the extra white spaces at the start or at the end of the search terms. We will implement a small helper library `StringUtils` that will then be used in the Library for the more flexible search functionality.

Implement the class `StringUtils` with a **static** method `public static boolean included(String word, String searched)`, which checks if the string `searched` is contained within the string `word`. As described in the previous paragraph, the method should be case insensitive and should not care about trailing and ending white spaces in the string `searched`. If either of the strings is *null*, the method should return `false`.

Tip: The methods `trim` and `toUpperCase()` of the class `String` might be helpful.

When you have completed the method, use it in the search functionality of the class `Library`.

Use the method as follows:

```
if(StringUtils.included(book.title(), searchedTitle)) {
    // Book found!
}
```

The improved library with the example:

```
Library Library = new Library();

Library.addBook(new Book("Cheese Problems Solved", "Woodhead Publishing", 2007));
Library.addBook(new Book("The Stinky Cheese Man and Other Fairly Stupid Tales", "Penguin Grou
Library.addBook(new Book("NHL Hockey", "Stanley Kupp", 1952));
Library.addBook(new Book("Battle Axes", "Tom A. Hawk", 1851));

for (Book book: Library.searchByTitle("CHEESE")) {
    System.out.println(book);
}
```

```
System.out.println("---");
for (Book book: Library.searchByPublisher("PENGUIN  ")) {
    System.out.println(book);
}
```

should output the following:

```
Cheese Problems Solved, Woodhead Publishing, 2007
The Stinky Cheese Man and Other Fairly Stupid Tales, Penguin Group, 1992
---
The Stinky Cheese Man and Other Fairly Stupid Tales, Penguin Group, 1992
```

28. ASSIGNMENTS WHERE YOU ARE FREE TO DECIDE HOW TO STRUCTURE THE PROGRAM.

Exercise 102: Grade distribution

This assignment corresponds to three assignment points.

Note1: Your program should use only one Scanner object, i.e., it is allowed to call `new Scanner` only once. If you need scanner in multiple places, you can pass it as parameter:

```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    // ...

    doSomething(scanner);
}

public static void doSomething(Scanner scanner) {
    String riw = scanner.nextLine();
    // ...
}
```

If another object needs a scanner, you can pass it as constructor parameter and save in instance variable.

Note2: Do not save anything in static variables. The main method is executed by the tests multiple times so the use of static variables might cause problems.

The input of the program is a set of exam scores of a course. Each score is an integer. When -1 is entered, the program stops asking for further input.

Inputting the exam scores should work as follows:

```
Type exam scores, -1 completes:
34
41
53
36
55
27
43
40
-1
```

After the scores have been read, the program prints the grade distribution and acceptance percentage of the course in the following form:

```
Grade distribution:
5: **
4:
3: ***
2: *
1: *
0: *
Acceptance percentage: 87.5
```

Grade distribution is formed as follows:

- Each exam score is mapped to a grade using the same formula as in exercise 18. If the score is not within the range 0-60 it is not taken into account.
- The number of grades are printed as stars, e.g. if there are 2 scores that correspond to grade 5, the line *5: ** is printed. If there are no scores that correspond to a particular grade, as is the case with grade 4 in the above example, the printed line is *4:*

All the grades besides zeros are accepted, so in the above 7 out of 8 participants were accepted. Acceptance percentage is calculated with the formula *100*accepted/allScores*.

Exercise 103: Birdwatchers database

Note1: Your program should use only one Scanner object, i.e., it is allowed to call `new Scanner` only once.

Note2: Do not save anything in static variables. The main method is executed by the tests multiple times so the use of static variables might cause problems.

This assignment corresponds to three assignment points.

In this assignment you are supposed to design and implement an observation database for a bird watcher. The database contains birds, each of which have a name and a Latin name, both Strings. Database also tracks how many times each bird has been observed.

The program should implement the following commands:

- `Add` - adds a bird
- `Observation` - adds an observation
- `Statistics` - prints all the birds
- `Show` - prints one bird
- `Quit` - terminates the program

The program should also handle the invalid inputs (see `Turing` below).

The following is an example how the program is supposed to work:

```
? Add
Name: Raven
Latin Name: Corvus Corvus
? Add
Name: Seagull
Latin Name: Dorkus Dorkus
? Observation
What was observed:? Seagull
? Observation
What was observed:? Turing
Is not a bird!
? Observation
What was observed:? Seagull
? Statistics
Seagull (Dorkus Dorkus): 2 observations
Raven (Corvus Corvus): 0 observations
? Show
What? Seagull
Seagull (Dorkus Dorkus): 2 observations
? Quit
```

Note you may structure your program freely, it is only required that the output of the program is as in the above example.

29. SORTING AN ARRAY

We'll get back to arrays again.

29.1 SORTING AN ARRAY WITH THE READY-MADE TOOLS OF JAVA.

As we've seen, there's all kinds of useful things already in Java. For example for handling ArrayLists you can find many useful help methods in the class Collections. For arrays you can find helpful methods in the class `Arrays`. Sorting a table can be done with `Arrays.sort(array)`.

Note: To be able to use the command you must have the following definition at the top of the program file:

```
import java.util.Arrays;
```

If you forget to write the `import` line, NetBeans will offer help with writing it. Try clicking the picture of the "bulp" that appears to the left from the line of code that is underlined with red.

The following program creates arrays and sorts the values in the array with the Arrays.sort -command.

```
int[] values = {-3, -111, 7, 42};
Arrays.sort(values);
for(int value: values) {
    System.out.println(value);
}
```

```
-111
-3
7
42
```

29.2 IMPLEMENTATION OF A SORTING ALGORITHM

It's easy to sort an array with the ready-made tools of Java. The general knowledge of a program requires knowing at least one sorting algorithm (or in other words, a way to sort an array). Let's get familiar with the "classic" sorting algorithm, choice sorting. Let's do this with a few excercise.

Exercise 104: Sorting

Note: in this assignment you're supposed to sort the array yourself. You can't use the help of the Arrays.sort()-method or ArrayLists!

EXERCISE 104.1: SMALLEST

Implement a method `smallest`, which returns the smallest value in the array.

The frame of the method is as follows:

```
public static int smallest(int[] array) {
    // write the code here
}
```

NOTE: You can't change the array that gets passed into the method!

The following code demonstrates the functionality of the method:

```
int[] values = {6, 5, 8, 7, 11};
System.out.println("smallest: " + smallest(values));
```

```
smallest: 5
```

EXERCISE 104.2: THE INDEX OF THE SMALLEST

Implement a method `indexOfTheSmallest`, which returns the index of the smallest value in the array (the position of the value in the array, that is).

The frame of the method looks like this:

```
public static int indexOfTheSmallest(int[] array) {
    // code goes here
}
```

NOTE: You can't change the array that gets passed into the method as a parameter!

The following code demonstrates the functionality of the method:

```
// indexes:   0  1  2  3  4
int[] values = {6, 5, 8, 7, 11};
System.out.println("Index of the smallest: " + indexOfTheSmallest(values));
```

```
Index of the smallest: 1
```

The smallest value of the table is 2 and its index (its location) in the array is 1. Remember that the numbering of an array begins from 0.

EXERCISE 104.3: INDEX OF THE SMALLEST AT THE END OF AN ARRAY

Implement a method `indexOfTheSmallestStartingFrom`, which works just like the method of the previous assignment, but only takes into consideration the end of an array starting from a certain index. In addition to the array the method gets as parameter an index, from which the search for the smallest will be started.

The frame of the method is as follows:

```
public static int indexOfTheSmallestStartingFrom(int[] array, int index) {
    // write the code here
}
```

NOTE: You can't change the array that gets passed into the method as a parameter!

The following code demonstrates the functionality of the method:

```
// indexes:    0  1  2  3   4
int[] values = {-1, 6, 9, 8, 12};
System.out.println(indexOfTheSmallestStartingFrom(values, 1));
System.out.println(indexOfTheSmallestStartingFrom(values, 2));
System.out.println(indexOfTheSmallestStartingFrom(values, 4));
```

```
1
3
4
```

In the example, the first method call finds the index of the smallest value starting from index 1. Starting from index 1 the smallest value is 6, and its index is 1. Respectively the second method call looks for the index of the smallest value starting from index 2. In this case the smallest value is 8 and its index is 3. The last call starts from the last cell of the array, in this case there is no other cells so the smallets value is in index 4.

EXERCISE 104.4: SWAPPING VALUES

Create a method `swap`, to which will be passed an array and two of its indexes. The method swaps the values in the indexes around.

The frame of the method looks like this:

```
public static void swap(int[] array, int index1, int index2) {
    // code goes here
}
```

The following showcases the functionality of the method. In printing the array we'll use the Arrays.toString-method which formats the array into a string:

```
int[] values = {3, 2, 5, 4, 8};

System.out.println( Arrays.toString(values) );

swap(values, 1, 0);
System.out.println( Arrays.toString(values) );

swap(values, 0, 3);
System.out.println( Arrays.toString(values) );
```

```
[3, 2, 5, 4, 8]
[2, 3, 5, 4, 8]
[4, 3, 5, 2, 8]
```

EXERCISE 104.5: SORTING

Now we've got a set of useful methods, with which we can implement a sorting algorithm known as selection sorting.

The idea of selection sorting is this:

- Move the smallest number of the array to index 0.
- Move the second smallest number to the index 1.
- Move the third smallest number to the index 2.
- and so forth

In other words:

- Inspect the array starting from index 0. Swap the value in index 0 and the smallest value in the array starting from index 0.
- Inspect the array starting from index 1. Swap the value in index 1 and the smallest value in the array starting from index 1.
- Inspect the array starting from index 2. Swap the value in index 2 and the smallest value in the array starting from index 2.
- and so forth

Implement the method `sort`, which is based on the idea above. The method ought to have a loop that goes through the indexes of the array. The methods `smallestIndexStartingFrom`

and `swap` are surely useful. Also print the contents of the array before sorting and after each round to be able to make sure that the algorithm works correctly.

Body of the method:

```
public static void sort(int[] array) {
}
```

Test the functionality of the method at least with this example:

```
int[] values = {8, 3, 7, 9, 1, 2, 4};
sort(values);
```

The program should print the following. Notice that you're to print the content of the array after each swap!

```
[8, 3, 7, 9, 1, 2, 4]
[1, 3, 7, 9, 8, 2, 4]
[1, 2, 7, 9, 8, 3, 4]
[1, 2, 3, 9, 8, 7, 4]
[1, 2, 3, 4, 8, 7, 9]
[1, 2, 3, 4, 7, 8, 9]
[1, 2, 3, 4, 7, 8, 9]
```

You'll notice how the array little by little gets sorted out starting from the beginning and advances towards the end.

30. SEARCHING

In addition to sorting, another very typical problem that a programmer runs into is finding a certain value in an array. Earlier, we've implemented methods that search for values in lists and arrays. In the case of arrays, values and strings can be searched for in the following way:

```
public static boolean isInArray(int[] array, int searchingFor) {
    for ( int value : array ) {
       if ( value == searchingFor )  {
           return true;
       }
    }
```

```
    return false;
}

public static boolean isWordInArray(String[] array, String searchingFor) {
    for ( String word: array ) {
        if ( word.equals(searchingFor) )  {
            return true;
        }
    }

    return false;
}
```

An implementation like this is the best we've been able to do so far. The downside of the method is that, if the array has a very large amount of values in it, the search will take a lot of time. In the worst case scenario the method goes through every single cell in the array. This means that going through an array that has 16777216 cells does 16777216 cell inspections.

On the other hand, if the values in an array are *ordered by size*, the search can be done in a notably faster way by applying a technique called *binary search*. Let's investigate the idea of binary search with this array:

```
// indexes   0   1   2   3    4   5    6   7   8   9  10
// values     -7  -3   3   7   11  15   17  21  24  28  30
```

Let's assume that we want to find the value 17. Let's utilize the information that the values of the array are in order instead of going through the array from the beginning. Let's inspect the middle cell of the array. The middle cell is 5 (the largest index 10 divided by two). The middle cell is marked with the asterisk:

```
                                *
// indexes   0   1   2   3    4   5    6   7   8   9  10
// values     -7  -3   3   7   11  15   17  21  24  28  30
```

At the middle is the value 15, which was not the value we were looking for. We're looking for the value 17, so since the cells of the array are ordered by size, the value cannot be on the left side of the 15. So we can determine that all indexes that are smaller or equal to 5 do not have the value we are looking for.

The area where we are searching for the value we want to find can now be limited to values that are on the right side of the index 5, or in other words, in the indexes [6, 10] (6, 7, 8, 9, 10). In the following, the searched value cannot be in the part of the array which is grey:

```
// indexes    0   1   2   3   4    5    6     7   8   9  10
// values      -7  -3   3   7  11   15   17    21  24  28  30
```

Next, let's inspect the middle index of the area that we have left; the middle index of indexes 6-10. The middle index can be found by getting the sum of the smallest and largest index and dividing it by two: (6+10)/2 = 16/2 = 8. The index 8 is marked with the asterisk below.

```
                                                  *
// indexes    0   1   2   3   4    5    6   7   8   9  10
// values      -7  -3   3   7  11   15   17   21  24  28  30
```

In index 8, we have the value 24, which was not the value we were looking for. Because the values in the array are ordered by size, the value we are searching for can not, in any case, be on the right side of the value 24. We can deduce that all indexes that are larger or equal to 8 can not contain the value we are looking for. The search area gets narrowed down again, the grey areas have been dealt with:

```
// indexes    0   1   2   3   4    5    6   7   8   9  10
// values      -7  -3   3   7  11   15   17   21  24  28  30
```

The search continues. Let's inspect the middle index of the area that we have left to search, that is, the middle index of indexes 6-7. The middle index can again be found out by getting the sum of the smallest and largest index of the search area and then dividing it by two: (6+7)/2 = 6.5, which is rounded down to 6. The spot has been marked with the asterisk.

```
                                      *
// indexes    0   1   2   3   4    5    6    7   8   9  10
// values      -7  -3   3   7  11   15   17   21  24  28  30
```

In the index 6 we have the value 17, which is the same as the value we've been looking for. We can stop the search and report that the value we searched for is in the array. If the value wouldn't have been in the array - for example if the searched-for value would've been 16 - the search area would have eventually been reduced to nothing.

```
                                      *
// indexes    0   1   2   3   4   56   7   8   9  10
// values      -7  -3   3   7  11   1517  21  24  28  30
```

So for the idea of binary search to become clear to you, simulate with pen and paper how the binary search works when the array is the one below and first you're searching for value 33 and then value 1.

```
// indexes   0   1   2   3   4   5   6   7   8   9  10  11  12  13
// values     -5  -2   3   5   8  11  14  20  22  26  29  33  38  41
```

With the help of binary search we look for cells by always halving the inspected area. This enables us to search in a very efficient way. For example, an array of size 16 can be divided in half up to 4 times, so 16 -> 8 -> 4 -> 2 -> 1. On the other hand, an array that has 16777216 cells can be halved up to 24 times. This means that with binary search we only need to inspect up to 24 cells in an array that has 16777216 cells in order to find our desired cell.

The efficiency of binary search can be inspected with logarithms. A base two logarithm (log_2) of the number 16777216 is 24 -- with the base two logarithm we can calculate how many times a number can be halved. Respectively the base two logarithm of the number 4294967296 ($log_2\ 4294967296$) is 32. This means that searching from a sorted array of 4294967296 different values would only take up to 32 cell inspections. Efficiency is an essential part of computer science.

Exercise 105: Guessing game

In this assignment we'll make an AI, which guesses the number the player is thinking about. The AI assumes that the number is between *lowerLimit...upperLimit*. The start of the game provides these limits to the method as parameters that makes the game happen. The AI asks the player questions in the format "Is your number greater than X?" and deduce the correct answer from the answers the player gives.

The AI keeps track of the search area with the help of the variables lowerLimit and upperLimit. The AI always asks if the player's number is greater than the average of these two numbers, and based on the answers the search area gets halved each time. In the end the lowerLimit and upperLimit are the same and the number the user is thinking of has been revealed.

In the following example the user chooses the number 44:

```
Think of a number between 1...100.
I promise you that I can guess the number you are thinking of with 7 questions.

Next I'll present you a series of questions. Answer them honestly.

Is your number greater than 50? (y/n)
n
Is your number greater than 25? (y/n)
y
Is your number greater than 38? (y/n)
y
Is your number greater than 44? (y/n)
n
Is your number greater than 41? (y/n)
y
Is your number greater than 43? (y/n)
y
The number you're thinking of is 44.
```

In the above example the possible value range is first 1...100. When the user tells the program that the number is not greater than 50 the possible range is 1...50. When the user says that the number is greater than 25, the range is 26...50. The deduction proceeds in the same fashion until the number 44 is reached.

In accordance to the principles of halving, or binary search, the possible search area is halved after each question in which case the number of required questions is small. Even between the numbers 1...100000 it shouldn't take more than 20 questions.

The program skeleton of the class `GuessingGame` that implements this is the following:

```
public class GuessingGame {

    private Scanner reader;
```

```
    public GuessingGame() {
        this.reader = new Scanner(System.in);
    }

    public void play(int lowerLimit, int upperLimit) {
        instructions(upperLimit, lowerlimit);

        // write the game logic here
    }

    // implement here the methods isGreaterThan and average

    public void instructions(int lowerLimit, int upperLimit) {
        int maxQuestions = howManyTimesHalvable(upperLimit - lowerLimit);

        System.out.println("Think of a number between " + lowerLimit + "..." + upperLim

        System.out.println("I promise you that I can guess the number you are thinking of wit
        System.out.println("");
        System.out.println("Next I'll present you with a series of questions. Answer them hon
        System.out.println("");
    }

    // a helper method:
    public static int howManyTimesHalvable(int number) {
        // we create a base two logarithm  of the given value
        // Below we swap the base number to base two logarithms!
        return (int) (Math.log(number) / Math.log(2)) + 1;
    }
}
```

The game is started the in following manner:

```
GuessingGame game = new GuessingGame();

// we play two rounds
game.play(1,10);  // value to be guessed now within range 1-10
game.play(10,99);  // value to be guessed now within range 10-99
```

We'll implement this assignment in steps.

EXERCISE 105.1: IS GREATER THAN

Implement the method `public boolean isGreaterThan(int value)`, which presents the user with a question:

```
"Is your number greater than given value? (y/n)"
```

The method returns the value `true` if the user replies "y", otherwise `false`.

Test your method

```
GuessingGame game = new GuessingGame();

System.out.println(game.isGreaterThan(32));
```

```
Is your number greater than 32? (y/n)
y
true
```

EXERCISE 105.2: AVERAGE

Implement the method `public int average(int firstNumber, int secondNumber)`, which calculates the average of the given values. Notice that Java rounds floating numbers down automatically, in our case this is perfectly fine.

```
GuessingGame game = new GuessingGame();
System.out.println(game.average(3, 4));
```

```
3
```

```
GuessingGame game = new GuessingGame();
System.out.println(game.average(6, 12));
```

```
9
```

EXERCISE 105.3: GUESSING LOGIC

Write the actual guessing logic in the method `play` of the class `GuessingGame`. You'll need at least one loop and a query in which you ask the user if their number is greater than the average of the lowerLimit and upperLimit. Change the upperLimit or lowerLimit depending on the user's reply.

Keep doing the loop until lowerLimit and upperLimit are the same! You can also test the game with smaller lower- and upperLimit values:

```
Think of a number between 1...4.
I promise you that I can guess the number you are thinking of with 2 questions.

Next I'll present you with a series of questions. Answer them honestly.

Is your number greater than 2? (y/n)
k
Is your number greater than 3? (y/n)
k
The number you're thinking of is 4.
```

Exercise 106: Implementation of binary search

The template you get from the test automaton has a start for an implementation of binary search. The class `BinarySearch` holds a method `public static boolean search(int[] array, int searchedValue)`, the job of which is to figure out, by using binary search, if the value given as a parameter is in the sorted array that is also given as parameter.

The method `search` does not work yet, however. Finish the method's implementation into a real binary search.

For testing, a separate main program can be found in the class `Main`, which has a frame like this:

```
import java.util.Arrays;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        // Here you can test binary search
        int[] array = { -3, 2, 3, 4, 7, 8, 12 };
        Scanner reader = new Scanner(System.in);

        System.out.print("Values of the array: " + Arrays.toString(array));
        System.out.println();

        System.out.print("Enter searched number: ");
        String searchedValue = reader.nextLine();
        System.out.println();

        boolean result = BinarySearch.search(array, Integer.parseInt(searchedValu

        // Print the binary search result here
    }
}
```

The execution of the program looks like this:

```
Values of the array: [-3, 2, 3, 4, 7, 8, 12]

Enter searcher number: 8

Value 8 is in the array
```

```
Values of the array: [-3, 2, 3, 4, 7, 8, 12]

Enter searcher number: 99

Value 99 is not in the array
```

31. ABOUT ARRAYS AND OBJECTS

If need be, any type of object can be put into an array. In the following, an example of an array into which will be put *Person* objects:

```
public static void main(String[] args) {
    Person[] persons = new Person[3];

    persons[0] = new Person("Pekka");
    persons[1] = new Person("Antti");
    persons[2] = new Person("Juhana");

    for ( int i=0; i < 30; i++ ) {
        persons[0].becomeOlder();
        persons[1].becomeOlder();
        persons[2].becomeOlder();
    }

    for ( Person person : persons ) {
        reportMaturity(person);
    }
}
```

First we create an array that can hold 3 Person objects. We put Pekka in slot 0, Antti in 1 and Juhana in 2. We age all by 30 years and check all of their matureness with the help of the method from the previous chapter.

The same example with ArrayLists:

```
public static void main(String[] args) {
    ArrayList<Person> persons = new ArrayList<Person>();

    persons.add( new Person("Pekka") );
    persons.add( new Person("Antti") );
    persons.add( new Person("Juhana") );

    for ( int i=0; i < 30; i++ ) {
        for ( Person person : persons ) {
            person.becomeOlder();
        }

        //  or persons.get(0).becomeOlder();
        //     persons.get(1).becomeOlder();
        //     ...
    }

    for ( Person person : persons ) {
        reportMaturity(person);
    }
}
```

In most situations it's better to use ArrayList instead of an array. However there can be cases where an array is adequate and is simpler to use.

A week always consists of seven days. It would be meaningful to form it out of exactly 7 Day objects. Since there's always 7 Day objects, an array will suit the situation very well:

```
public class Day {
    private String name;
    // ...
}

public class Week {
    private Day[] days;

    public Week(){
        days = new Day[7];
        days[0] = new Day("Monday");
        days[1] = new Day("Tuesday");
        // ...
    }
}
```

32. FINAL WORDS

Object-Oriented Programming with Java, Part I ends here. Congratulations for making it this far! If you want to continue learning and programming, start our second part of the course here: Object-Oriented Programming with Java, Part II

Assuming you have NetBeans and TMC already installed in your computer, in NetBeans just change the current course from TMC -> Settings to **2013-OOProgrammingWithJava-PART2** and you should be good to go!