## Java exercises

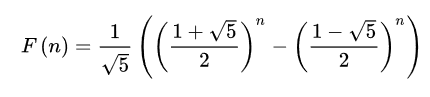
1. Write code to calculate the distance between two points in a two-dimensional space. The formula for calculating the distance is .
2. Write code to display all numbers between n and m which are not divisible by 7 and/or 17.
3. Write code that displays all divisors for a given number.
4. We have the following banknotes: 100, 50, 20, 10 and 5. Make a program to break an amount of money into smallest possible number of banknotes. You can assume the amount is an integer number.
5. Write a program where the user tries to guess what number (between 1 and 100) is stored in the program. The program must give a hint to the user whenever he/she/non-binary guesses wrong. The program stops after the guess is correct.
6. Seconds is given as an integer.
   1. Write a program to display it in years, days, hours, minutes and seconds.
   2. Make a cleaver testing program for point a).
7. A triangle can be formed if the sum of the lengths of every two sides is longer than the length of the third side. Make a program that checks if given three lengths can be used to form a triangle.
8. Code a program to display a square of given size (height and width). Only the frame of the square is to be displayed.
9. Code a program to display an isosceles triangle. The height of the triangle is given. If the given height is 3, then the triangle should look like:

\*

\*\*\*

\*\*\*\*\*

1. Reverse a given integer number. Example: 2385 becomes 5832.
2. Write a program that reads an integer of 5 digits. Separate all the digits. All possible errors of the given number must be handled. Think hard of all the possible errors.
3. Define a two-dimensional String-array. Fill the array with strings representing the indexes. Example: Index [0][0] includes the string ”00” and index [7][4] includes the string “74”.
4. Draw 100 random numbers between 1 and 100 and save them in an array.
   1. Traverse through the array and display the positions were the value is 5 or under. How many such positions should there be?
   2. Make point a) to a subroutine which gets the array and the value to search as parameters.
5. Draw 100 random numbers between -50 and 50.
   1. Save the negative ones in one array and the positive ones in another array. How long should the arrays be?
   2. The length of the arrays must be exactly the same as the numbers stored in them. This means you have to make a dynamic array *by hand*. You may not use an ArrayList or any other data structure present in Java.
6. Reversing a string. Input: ”Pig is an animal”, Output: “lamina na si giP”.
   1. Write the code in the main method.
   2. Write a subroutine which gets the input as a parameter and returns the reversed string.
   3. Write a class to hold a String. Make a method which reverses the String saved in the class. The method changes the inner state of the class.
7. Counting characters.
   1. Write code to count how many times different characters occurs in a String.
   2. Make the previous a subroutine. The subroutine gets the String as a parameter and returns the result. What is returned is the most important thing here.
   3. After we have handled data structures rethink this. Is there a better way for the return value?
8. Calculating the average of numbers.
   1. Write code to calculate the average of 5 numbers.
   2. Write a subroutine which calculates the average of n numbers. The numbers are passed as a parameter to the subroutine and the average is returned to the caller.
9. Make a class which holds an undefined amount of numbers.
   1. The user must be able to insert and delete numbers from the class.
   2. The class has a method for returning the average of the numbers.
   3. The class holds the average of the numbers. This means the class *updates* the average whenever necessary. Jari remembers to explain this.
   4. After finishing part c) think: why would we ever write such code? It has *side-effects* and as such violates f.ex. the paradigm of functional programming.
10. Make a class to hold the time of the day. The time is constructed of hours, minutes and seconds.
    1. Make methods which increases/decreases the time.
    2. Make a comparator which orders the time-objects into ascending order.
    3. Make a comparator which orders the time-objects into descending order.
11. Make a class to hold a date. The date includes the number of the day, the number of the month and the number of the year. The class also holds the time of the day which it inherits from the class made in exercise 19.
    1. Make methods which increases/decreases the date.
    2. Make comparators (ascending and descending) which orders the dates.
12. The formula below can be used to calculate the Fibonacci numbers. We could mathematically prove that the formula is correct, but that is quite difficult, especially if you’re not a mathematician. Instead we can prove it works by coding a program to test the formula. Do that. The formula is copied from: https://fi.wikipedia.org/wiki/Fibonaccin\_lukujono.



1. A Fibonacci number is calculated as the sum of the two previous Fibonacci numbers, i.e. Fi = Fi-1 + Fi-2. The two first numbers are 0 and 1 and after that the series continues as explained. Make a class to calculate any “nacci” series. See <https://fi.wikipedia.org/wiki/Fibonaccin_lukujono>, for details.
2. The Koodit folder includes a file named Jobs.txt. It has 1080 lines of job descriptions. A single line is constructed of
   1. a unique *id*
   2. a *day* identifier (ma1-su3 = the first Monday – the last Sunday)
   3. the *code* of the job
   4. the *duration* of the job in minutes
   5. the *starting time* of the job
   6. the *ending time* of the job

Make a class to hold the values.

1. With the previous data, make data structures to hold
   1. all the jobs in the order they are in the file
   2. all the codes of the jobs
   3. all job ids for every day and week, the jobs must be ordered by the ids
   4. the amount of different codes of jobs for every week
   5. all the codes of the jobs for every day
   6. the amount of different starting times for every day, must be ordered by the starting time and day
   7. all job ids for every code of job, must be ordered by the ids

When making the structures, remember that you may want to update them.