Learning goals

Before the next day, you should have achieved the following learning goals:

- Be able to write recursive for simple problems
- Understand what a recursive method looks like, and how it differs from an iterative method.

Remember that **exercises marked with a star (*) are harder and take longer to solve**, so you should not try them until you have solved all the non-star ones.

1 Recursive code, line by line

a)

What number would the following method print on screen if you called `printNumbers(6)`?

```
void printNumbers(int n) {
    if (n <= 0) {
        return;
    }
    printLine(n);
    printNumbers(n-2);
    printNumbers(n-3);
    printLine(n);
}
```

Do this exercise on paper. Then add this method to a Java Decaf program and check your answer.

b)

Do you see anything wrong with the following code? How would you fix it?

```
String buggyMethod(int n) {
    if (n == 0) {
        return "";
    }
    return n + " " + buggyMethod(n - 2);
}
```

c)

Do you see anything wrong with the following code? How would you fix it?

```
String doggyMethod(int n) {
    String result = doggyMethod(n-3) + n + doggyMethod(n-2);
    if (n <= 0) {
        return "";
    }
    return result;
}
```

d) McCarthy's 91 function

Calculate the result of calling this method with arguments 50, 73, and 95. Note that the recursion on line 5 is double.

```
int mcCarthy91(int n) {
    if (n > 100) {
        return n - 10;
    } else {
       return mcCarthy91(mcCarthy91(n+11));
    }
}
```

(Hint: If you get too confused, maybe writing a slightly modified version of this code that prints the numbers on screen can help).

2 Classics

2.1 a) Factorial

Write a small program with a method that calculates the factorial of an integer number as seen in the notes.

Is it easy to do this both iteratively and recursively? Try both ways and see which is more natural for you. If one takes too long, try the other way.

2.2 b) Fibonacci

Write a small program with a method that calculates the nth element of the Fibonacci sequence as seen in the notes.

Is it easy to do this both iteratively and recursively? Try both ways and see which is more natural for you. If one takes too long, try the other way.

When doing it recursively, do it with and without memoization. Compare the time that is needed in each case to find F(40) or F(45).

2.3 c) Hanoi towers

A legend says that, somewhere in the East, near Hanoi, there is a temple. In the temple, there are three posts. In the posts, there are 64 discs of 64 different sizes. When the world was created, all the discs were in the first post; when all the discs are moved to the last post, the world will end.

The monks in the temple move the discs between posts, but they must obey two simple rules. First, only one disc can be moved at a time, from one post to another post (it cannot be left anywhere else). Last, but not least, a disc can only be placed on top of a bigger disc, never on top of a smaller disc. The smallest disc can be placed on any post (all other discs are bigger); the biggest disc can only be placed on an empty post.

Create a method that calculates the number of moves necessary to move a set of n Hanoi disks from the initial post to the last post. **Hint:** if you want to play monk yourself in order to better understand the problem, ask the lecturer for a “Hanoi envelope”.

3 Paper sizes (*)

A Din-A0 sheet of paper is 841mm $\times$ 1189mm (its surface is one square meter). Successive measurements of paper are defined recursively as “half” or “double” the preceding size. Therefore, a Din-A1 sheet of paper is half of Din-A0, or 594mm $\times$ 841mm; while a Din-A00 is double of Din-A0, or 1189mm$\times$ 1682mm.

Create a method that takes a String parameter representing a size (e.g. ”A4”, “A00000”) and prints on screen the size of the corresponding sheet of paper. For simplicity, you can ignore rounding errors when calculating smaller sizes.

Hint: note that two consecutive sizes always share one side and only the other one is multiplied or divided by two, e.g. the *short* side of Din-A2 is as long as the *long* side of Din-A3, while the long side of Din-A2 is twice as long as the short side of Din-A3.

4 Palindrome

Create a program with a (recursive) method that takes a String paremeter and returns true if the String is a palindrome and false otherwise. Compare this recursive version with the iterative version you wrote in past weeks. Which one seems clearer to you?

5 Power

Create a class with a static method `pow` that takes to integers (`base` and `exponent`) and calculates the power, e.g. `pow(2,3)` calculates 2^3.

Is it easy to do this both iteratively and recursively?

6 Eight Dames (**)

Create a method that calculates the solution to the problem of the eight Dames: "given a chess board, and knowing that the Dame can reach any tile horizontally, vertically, or diagonally, place 8 Dames on the board so that none of them can reach the others by doing only one move". You can return the solution as an array of 8x8 booleans where eight tiles are true (those with the Dames on them) and the others are false.

For extra complexity, make your method return all the possibilities in which the problem can be solved (this may take a long time).

7 Hanoi Redux (**)

The legend of the Towers of Hanoi has inspired countless variations as mind games. Here is one of them: if we have three posts and 64 discs, where all the even-numbered discs are on the leftmost post while the odd-numbered discs are on the rightmost post, how many moves do we need to exchange the discs following the Hanoi rules for disc movement?

Write a method that calculates the number of moves necessary to exchange n discs.

8 How big is your stack? (*)

Write a method that calculates the maximum size of the stack (measured in method calls) by forcing a `StackOverflowError`. Use different methods, which different number and types of local variables, and see how the number of maximum function calls changes.