Recursion

Camille Duquesne
1ère IB

Any questions/difficulties you would like to share about last session's content ?

B2 Programming

B2.4.4 Explain the fundamental concept of recursion and its applications in programming. (HL only)

B2.4.5 Construct and trace recursive algorithms in a programming language. (HL only)

B2.3.1 Construct programs that implement the correct sequence of code instructions to meet program objectives.

- The impact of instruction order on program functionality
- Ways to avoid errors, such as infinite loops, deadlock, incorrect output

What is recursion ?

Recursion is a programming concept and technique where **a function calls itself** to solve a problem by breaking it down into smaller, more manageable subproblems.

All recursive functions share a common structure consisting of two parts: **the base case** and the **recursive case**. The function keeps calling itself, the **recursive case**, with modified input until it reaches a **base case**, which is a condition that signals the termination of the recursion.

Recursion is a powerful and elegant way to solve complex problems that can be naturally divided into simpler instances of the same problem.

Factorials - iterative solution

How do you compute 5! (factorial of 5) using a loop ?

How would you write a method that computes n! with a loop ?

Factorials - recursive solution

Now let's deconstruct the factorial:

5! = 5 x 4 x 3 x 2 x 1	4! = 4 x 3 x 2 x 1	3! = 3 x 2 x 1	2! = 2 x 1	1! = 1
5! = 5 x 4!	4! = 4 x 3!	3! = 3 x 2 !	2! = 2 x 1!	

What is the pattern that repeats ?

What is the base case ?

Factorials - recursive solution

So our pattern/recursive case is n! = n x (n-1) !

And our base case is 1. When n equals 1 the recursive pattern stops being true

```
public static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        return n * factorial( n: n-1);
    }
}
```

The call stack

The **call stack** is a data structure that is used to **manage the execution of function calls** in a program.

As functions are called, their frames are pushed onto the stack. When a function completes its execution (returns), its frame is popped (removed) from the stack. This happens in the reverse order of the function calls.

We'll learn more about stacks in a few weeks !

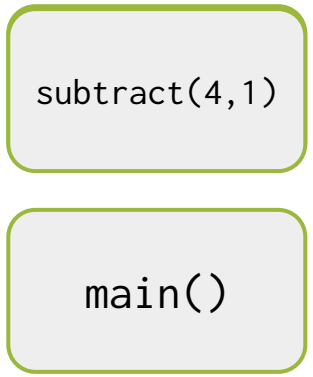
The call stack

```
method main()
    output add(12,6)
    output subtract(5,1)
end main

method add(a,b)
    output “adding”
    return a-b
end add

method subtract(a,b)
    return a-b
end subtract
```

The main



The call stack

The output

“adding”

12

4

The call stack - recursive methods

What would be the call stack of factorial(5) ?

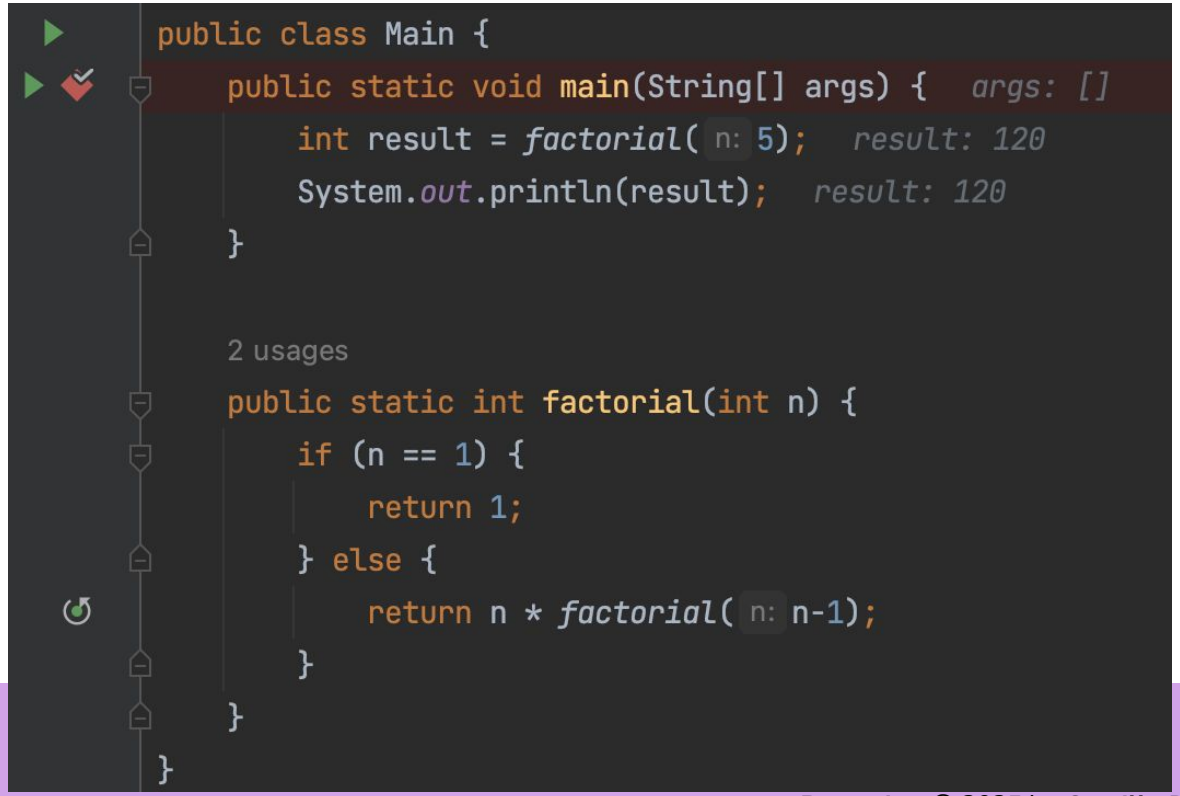
```java
public static int factorial(int n) {
    if (n == 1) {
        return 1;
    } else {
        return n * factorial( n: n-1);
    }
}
```

Winding and unwinding

Winding is the process that occurs when recursive calls are made until the base case is reached eg. pushing all functions call onto the stack

Unwinding is the process that occurs when the base case is reached, and the values are returned to build a solution eg. removing/popping all functions call from the stack

How can I view the call stack in IntelliJ ?



If you add a breakpoint (red dot on the right) and then press debug, you will be able to see the call stack !

https://blog.jetbrains.com/idea/2020/05/debugger-basics-in-intellij-idea/

When NOT to use recursion ?

If your **recursion depth is too deep** (i.e., you have many recursive calls), it can lead to a stack overflow error. In contrast, iterative solutions typically do not have stack depth limitations.

For some problems, iterative solutions can be more efficient than recursive ones in terms of both **time and memory usage.**

Iterative solutions can be more straightforward to implement and understand, particularly for problems that do not naturally lend themselves to recursive decomposition. Recursive code may require **additional mental effort to follow.**

Advantages vs disadvantages of recursion

Advantages	Disadvantages
• Elegant and simple • Concise • Readability • Useful for complex problems • Can represent a certain pattern closer to reality	• Stack overflow • Difficult debugging • High memory usage • Slower execution time

What does this addIntUpTo(4) output ? Draw the call stack

```java
public static int addIntUpTo(int n){
    if (n == 1) {
        return 1;
    } else {
        return n + addIntUpTo( n: n - 1);
    }
}
```

What does foo(5) output ?

```
public static int foo(int n){
    if (n <= 1) {
        return 1;
    } else {
        return foo( n: n-1) + foo( n: n - 2);
    }
}
```

Is the following sequence of numbers recursive ?

If yes what is the base case ? What is the recursive case ?

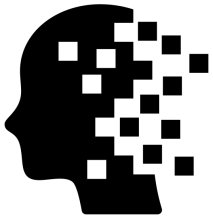
```
1, 2, 4, 8, 16, 32, 64, ...
```

Is the following sequence of numbers recursive ?

If yes what is the base case ? What is the recursive case ?

```
1, 4, 9, 16, 25, 36, 49, ...
```

Pause & Recall



Created by Victoruler
from Noun Project

Close your eyes and try to recall as many things as possible that were covered during this lesson.

Alternatively, you can keep your eyes open and write down as many things you remember on a piece of paper.

This will help strengthen your memory of key concepts 💪

Exercise 1 *(GenAI 🟠 Orange)*

Identify two essential features of a recursive algorithm

Exercise 2 *(GenAI 🟠 Orange)*

Consider the following recursive method:

```
public static void recursionEx(int x){
    if (x != 0) {
        System.out.println(x);
        recursionEx( x: x-1);
        System.out.println(x);
    }
}
```

Copy and complete the trace table to show the behavior of `recursionEx(3)`

parameter passed to method	output
3	

Exercise 3 *(GenAI 🟠 Orange)*

Explain what would happen if we tried to call `recursionEx(-3)` in the previous exercise.

Exercise 4 *(GenAI 🟠 Orange)*

Compute the value of mystery(7). Show all your working

```
public static int mystery(int n){
    if (n > 0) {
        return 3  + mystery( n: n-3);
    } else {
        return 3;
    }
}
```

Exercise 5 *(GenAI 🟠 Orange)*

If the following sequence of numbers is recursive, write a general java recursive method for generating such sequence

```
1, 3, 6, 10, 15, 21, 28, ...
```

Exercise 6 *(GenAI 🟠 Orange)*

Write a recursive Java function to calculate the sum of all elements in an array of integers.

Exercise 7 *(GenAI 🟠 Orange)*

Outline one disadvantage of solving problems recursively.

Exercise 8 *(GenAI 🟠 Orange)*

Design a recursive Java function to check if a given string is a palindrome (reads the same forwards and backwards).

Explain why recursion is suitable for writing the previous method.

Exercise 9 *(GenAI 🟠 Orange)*

If the following sequence of numbers is recursive, write a general java recursive method for generating such sequence

```
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, ...
```

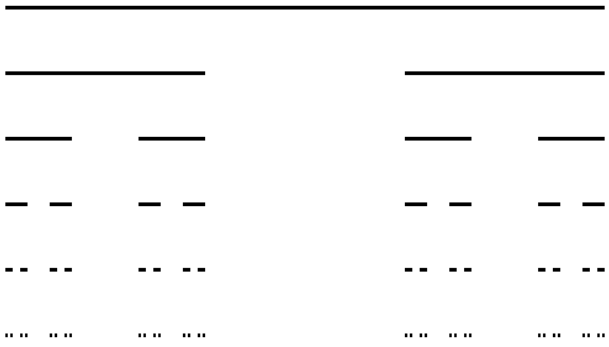
Exercise 10 *(GenAI 🟠 Orange)*

Code both a recursive and non recursive function that prints out the fibonacci sequence.

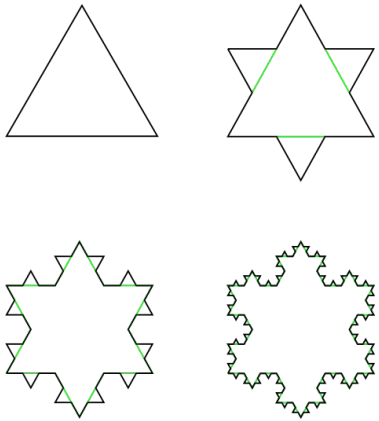
Bonus - Exercise 11 *(GenAI 🟠 Orange)*

Code a geometrical pattern, such as the cantor set or the koch snowflake.

Read some documentation online to see how you can draw shapes in java.



https://nilesjohnson.net/teaching/cantor.gif



https://en.wikipedia.org/wiki/koch_snowflake?width=1000&height=540

Homework

Finish all the exercises

Create flashcards for the following terms:

General case

Base case

Winding

Unwinding

Call stack