Some final words …

Current Challenges in OOP

- Absence of common conceptual framework
- Identifying objects
- complex hierarchies can puzzle programmers. KISS
- Writing tests and refactoring can be harder. “Everything comes with an implicit environment”. Banana vs Gorilla with bananas ☺

Still…

- OOP – one of the dominant paradigms
- Success of OOP languages - java, c#, python, Kotlin, Swift, Ruby, etc.

Approaches to programming

- OOP highlights the importance of objects
- Procedural (Imperative) programming is about execution of sequential commands
- Functional programming emphasizes the definition of functions

Nowadays the most popular approaches to programming consider fusion of OO and F programming.

Twitter case. Lambdas in java and LINQ in c#. Scala. Swift.

Valuable OOP aspects that keep omnipresent

- Encapsulation − hide internal info
- Abstraction − providing easy-to-use public interface to data
- Polymorhism

- Inheritance
  - Allows code reuse
  - But FP also allows DRY via reusable functions

Scala

- Scala provides a smooth integration of F, OOP approaches. Can work with Java, c#
- Multiple inheritance via traits and mixin composition
- Other alternatives − Clojure, O caml, Haskell, F#
- Please, read more about F-system and Scala in Chapter 10

** The next level following after OOP is functional programming, where inputs and outputs can be functions. For example, in Scala (another functional languages you can try: f#, Haskell, Clojure, OCaml…) we can define the function that accepts a function as a parameter:

```
def fold(f: (Int)=>Unit, b:Int, c:Int):Unit= if(b<=c) {f(b); fold(f, b+1, c) }
```

So, this function accepts as parameters a function f that takes Int and returns nothing (Unit is like void), and two integers b and c. What it does is applies function f to b increasing b while b is <=c. Your task is to create *smth similar* in java – some method that can accept a function as a parameter.

Sudoku in Java

```
public class SudokuApp {
    static int size=9;
    static boolean CantPut(int i, int j, int value, int [][]board){
        if( i>=size || j>=size) return false;
        for(int k=0; k<size;k++)
            if(board[i][k]==value || board[k][j]==value || board[i/3*3+k/3][j/3*3+k%3]==value) return true;
        return false;
    }
    static boolean find(int i, int j, int[][]board){
        if(i==size) {i=0;
                    if(++j == size) {print(board); return true;}}
        if(board[i][j]!=0) return find(i+1,j, board);
        for(int value=1; value<=size;value++){
            if(!CantPut(i, j, value, board)){
                board[i][j] = value;
                if (find(i+1,j,board)) return true;
            }
        }
        board[i][j]=0;
        return false;
    }
    static void print(int [][]board){
        for(int i=0; i<size; i++)  System.out.println(Arrays.toString(board[i]));
    }
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int [][]board = new int[size][size];
        for(int i=0; i<size; i++){
            String s = in.next();
            for(int j=0; j<s.length(); j++) board[i][j] = s.charAt(j)-'0';}
        find(0,0,board);
    }
}
```

Sudoku in Scala

```
1 object SudokuApp extends Application {
2   var board: Array[Array[Char]] = (List.tabulate(9)((x:Int)⇒readLine.toArray)).toArray; // val squares = List.tabulate(6)(n ⇒ n * n)
3 var size: Int = 9; var sols: Int=0;
4 def print = println(board.deep.mkString("\n"))
5 def CantPut(k:Int, i:Int, j:Int, value:Char): Boolean = {//looping 0 ... 8
6   if(i ⩾ size || j ⩾ size || k ⩾ size) return false;
7   return board(i)(k) == value || board(k)(j) == value || board(i/3*3 + k/3)(j/3*3 + k % 3) == value || CantPut(k+1, i, j, value)
8 }
9 def fold(f: (Int)⇒Unit, b:Int, c:Int):Unit= if(b ⩽ c) {f(b); fold(f, b+1, c) }
10
11 def find(i: Int, j: Int):Unit = (i, j) match {
12 case (9, 8) ⇒ {print;sols=sols+1;sols}
13 case (9, j) ⇒ {find(0, j+1)}
14 case (i, j) ⇒ if(board(i)(j) ≠ '0') find(i+1, j) else
15   fold(( value:Int) ⇒
16   if(!CantPut(0, i, j, (value + 48).asInstanceOf[Char]))  {
17     board(i)(j) = (value + 48).asInstanceOf[Char];
18     find(i+1, j);
19     board(i)(j) = '0'; },1, 9)}
20 find(0,0)
21 println(sols) }
22
```

The Scientific Method

- **Scientific Method:** Systematic process for generating, rigorously and unequivocally, new scientific knowledge
- Typical stages (*high level overview*):
  1 – Observation *(...and previous knowledge in the field)*
  2 – Formulation of hypothesis *(thesis proposal and CAT)*
  3 – Design and experimentation *(artifacts, tests and measurements)*
  4 – Interpretation of results
  5 – Conclusions and new observations *(thesis)*
  6 – Dissemination of conclusions *(dissertation and published articles)*

Basic terms in research

- Hypothesis
- Significance
- Contribution
- Science, Research, Technology
- H-index
- “Wall of knowledge”
- Hypothetico-deductive method

Think about your work proposal

- Six essential questions that must be answered by a proposal (expect multiple iterations)

1. What is the problem? From the literature, there are 2 possibilities:
   - New problem > *find a solution*
   - Known problem & existing solutions > *find a better solution*
2. What has been done (by others) already to solve this problem?
3. What is missing? What is not good in other approaches/solutions?
4. What are you planning to do?
5. What will be the result(s) in the end?
6. Rough idea of the way to the end.

My Creation is Better

- Discovering a fact about nature (or about the math world), it is a contribution *per se,* no matter how small
- But [in a synthetic field] anyone can create some new thing
- One must show that the creation is **better**
  - *Solves a problem in less time*
  - *Solves a larger class of problems*
  - *Is more efficient of resources*
  - *Is more expressive by some criterion*
  - *Is more visually appealing*
  - *Presents a totally new capability*
  - *…*
- The "better" property is not simply an observation.