Hello, Turing

[workshop]

today we will discuss:

Procedural/Functional Paradigm

Speakers:

@fredcolin079 Alisher Toleberdyyev

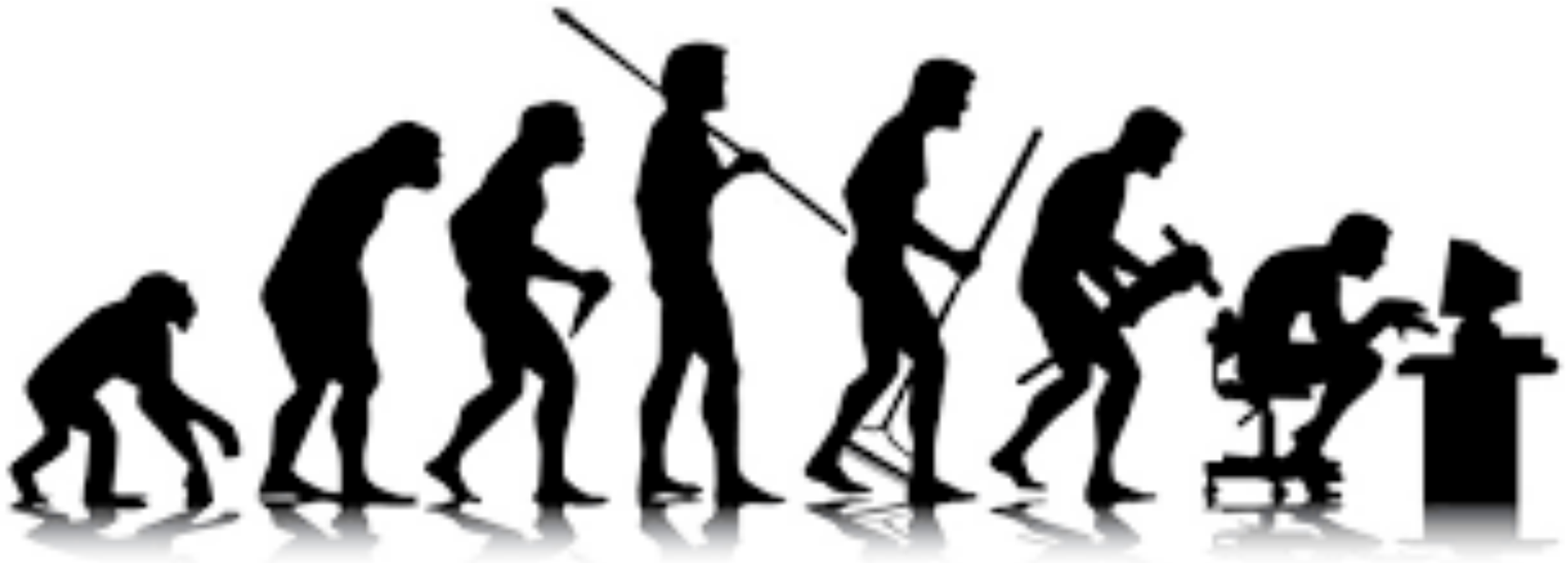
1st

We’ll talk about

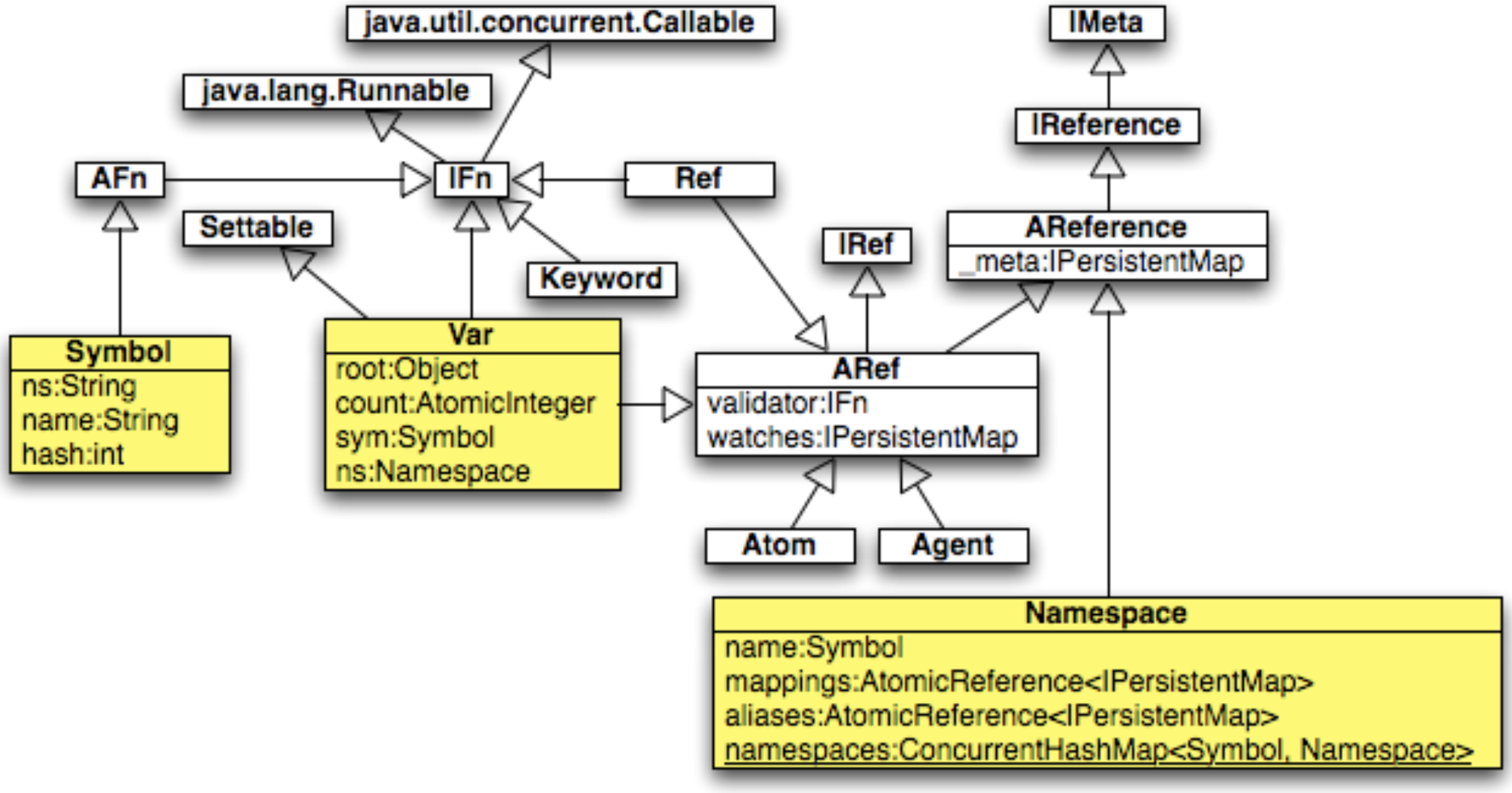
Procedural paradigm

Процеду́рное программи́рование — **программирование** на императивном языке, при котором последовательно выполняемые операторы можно собрать в подпрограммы, то есть более крупные целостные единицы кода, с помощью механизмов самого языка.

Serial, Step by Step



СЦЕНАРИЙ ВАШЕГО ФИЛЬМА



Функциона́льное программи́рование — раздел дискретной математики и парадигма **программирования**, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании последних (в отличие от функций как подпрограмм в процедурном **программировании**).

Functional Programing

```
let meetups = [
  {name:'JavaScript', isActive:true, members:700},
  {name:'Angular', isActive:true, members:900},
  {name:'Node', isActive:false, members:600},
  {name:'React', isActive:true, members:500}
];
let sumFPChain = meetups.filter((m)=>{
    return m.isActive;
  })
  .map((m)=>{
    return m.members- (0.1*m.members);
  })
  .reduce((acc, m)=>{
    return acc + m;
  },0);
console.log(sumFPChain);   // Output will be 1890
```

Imperative Programming

```
const arr = [1, 2, 3, 4, 5, 6 ,7 ,8 ,9]

function getOdds(arr){
   let odds = [ ];

    for(let i = 0; i < arr.length + 1; i++){
        if ( i % 2 !== 0 ){
           odds.push( i )
        };
      };
    return odds
  };

console.log(getOdds(arr))
// logs [1, 3, 5, 7, 9]
```

Functional
Programming

```
function getOdds2(arr){
   return arr.filter(num => num % 2 !== 0)
}

console.log(getOdds2(arr))
// logs [ 1, 3, 5, 7, 9 ]

// this can be even shorter
const getOdds3 = arr => arr.filter(num => num % 2 !== 0)

console.log(getOdds3(arr))
// logs [ 1, 3, 5, 7, 9 ]
```

OO pattern/principle	FP pattern/principle
• Single Responsibility Principle	• Functions
• Open/Closed principle	• Functions
• Dependency Inversion Principle	• Functions, also
• Interface Segregation Principle	• Functions
• Factory pattern	• Yes, functions
• Strategy pattern	• Oh my, functions again!
• Decorator pattern	• Functions
• Visitor pattern	• Functions □

FIN