Peter the Great
Saint-Petersburg Polytechnic University

Ocaml

Императивные и объектные возможности

30/10/2025

Презентацию подготовили:
- Дрекалов Никита
- Соколов Артём

POLYTECH

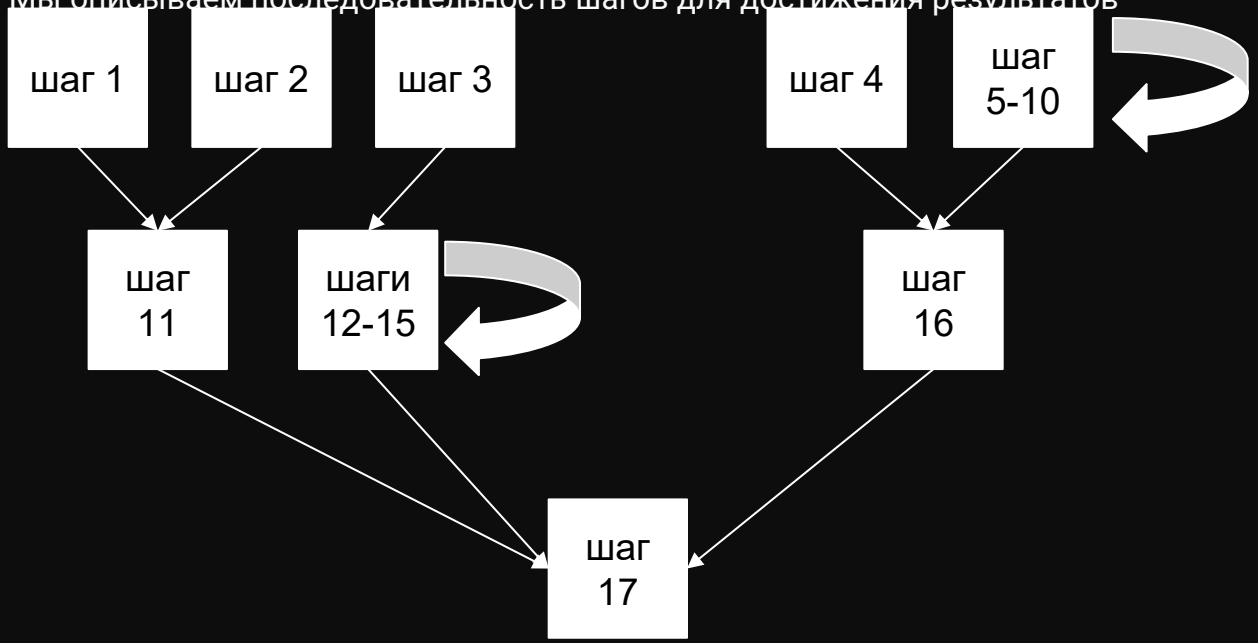
План презентации

- Вспоминаем, что такое императивный подход
  - Циклы
  - Рекурсия
  - Массивы
- Объекты
  - `Базовый синтаксис`
  - `Наследование`
  - `Полиморфизм`
  - `Инкапсуляция`

Императивный подход

Императивный подход

Мы описываем последовательность шагов для достижения результатов



POLYTECH

Декларативный подход

Мы описываем желаемый результат

Циклы

```
let imperative_sum () =
  let total = ref 0 in
  for i = 1 to 10 do
    total := !total + i
  done;
  !total
```

Мы описываем, как мы получаем результат

```
let functional_sum () =
  List.init 10 (fun i -> i + 1)
  |> List.fold_left ( + ) 0
```

Мы описываем, что мы хотим получить, а именно буквально “сумма элементов списка от 0 до 10”

Рекурсия

```
let rec recursive_sum n =
  if n <= 0 then 0
  else n + recursive_sum (n - 1)

let tail_recursive_sum n =
  let rec aux n acc =
    if n <= 0 then acc
    else aux (n - 1) (acc + n)
  in
  aux n 0
```

Рекурсия вместо цикла

Оптимизированная “хвостовая” рекурсия

Массивы (1/2)

```
let fibonacci_imperative n =
  if n <= 1 then n
  else
    let fib_array = Array.make (n + 1) 0 in
    fib_array.(0) <- 0;
    fib_array.(1) <- 1;

    for i = 2 to n do
      fib_array.(i) <- fib_array.(i - 1) + fib_array.(i - 2)
    done;

    fib_array.(n)
```

Массивы (2/2)

```
let fibonacci_functional n =
  let rec loop counter a b =
    if counter >= n then a
    else loop (counter + 1) b (a + b)
  in
  if n <= 1 then n
  else loop 1 0 1
```

Рекурсия вместо обхода массива

Объектные возможности

Функции высшего порядка (1/3)

- `Функции высшего порядка` - это функции, которые:
  - принимают функци(и/ю) в качестве аргуметр(а/ов)
  - возвращают функци(и/ю)

```
let pow2 x = x * x
let double_f f x = f (f x)

let result = double_f pow2 2 (* pow2 -> pow4 *)
```

Функции высшего порядка (2/3)

```
let give_me_flex () = "flex"

let _function = give_me_flex    (* val _function : unit -> string = <fun> *)
let _result   = give_me_flex () (* val _result : string = "flex" *)
```

Пример (3/3)

```
type user = { name: string; age: int; active: bool; }

let users = [
    { name = "Nikita"; age = 22; active = false };
    { name = "Artyom"; age = 22; active = true  };
    { name = "Artur";  age = 17; active = true  };
]

let process_users user_list filter_func map_func =
    List.filter filter_func user_list
    |> List.map map_func

let result = process_users
    users
    (fun usr -> usr.active)
    (fun usr -> { usr with name = usr.name ^ " is cool user" })
```

Объектный подход

Базовый синтаксис класса

```
class point x_init y_init =
  object
    val mutable x = x_init
    val mutable y = y_init

    method get_x = x
    method get_y = y

    method set_x new_x = x <- new_x
    method set_y new_y = y <- new_y

    method move dx dy =
      x <- x + dx;
      y <- y + dy
    method print =
      Printf.printf "Point(%d, %d)\n" x y
  end
```

```
let my_own_point = new point 10 20;;
my_own_point#print;; (* Point(10, 20) *)
my_own_point#set_x 4;;
my_own_point#set_y 5;;
my_own_point#print;; (* Point(4, 5) *)
```

Наследование (встраивание)

```
class colored_point x y color =
  object
    inherit point x y as super

    val mutable color = color

    method get_color = color
    method set_color new_color = color <- new_color

    method print =
      Printf.printf "ColoredPoint(%d, %d, %s)\n"
        (super#get_x) (super#get_y) color
  end
```

Встраивание

Параметризованные классы

```
class ['a] stack =
  object
    val mutable elements : 'a list = []

    method push x = elements <- x :: elements
    method pop =
      match elements with
      | hd :: tl -> elements <- tl; Some hd
      | [] -> None
    method is_empty = elements = []
  end
```

['a]

Виртуальные методы

Родительский класс

```
class virtual shape name =
  object
    method virtual area : float
    method virtual perimeter : float
    val name = name

    method get_name = name
    method print_info =
      Printf.printf "Shape: %s, Area: %.2f, Perimeter: %.2f\n"
        name (self#area) (self#perimeter)
  end
```

Дочерний класс

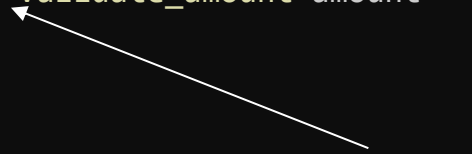
```
class circle radius =
  object
    inherit shape "Circle"

    method area = 3.14159 *. radius *.
radius
    method perimeter = 2. *. 3.14159 *.
radius
  end
```

Полиморфизм

```
class rectangle width height =
  object
    inherit shape "Rectangle"

    method area = float width *. float height
    method perimeter = 2. *. (float width +. float height)
  end

let shapes : shape list = [
  (new circle 5.0 :> shape);
  (new rectangle 4 3 :> shape)
]

let print_all_shapes shapes =
  List.iter (fun shape -> shape#print_info) shapes
```

Инкапсуляция

```
class bank_account initial_balance =
  object (self)
    val mutable balance = initial_balance
    val account_number = Random.int 10000

    method private validate_amount amount =
      amount > 0
end
```



private поле

Спасибо за внимание!