Парадигмы программирования: Функциональное, Объектно-ориентированное, Аспектно-ориентированное

Егоркин Станислав и Зайдиев Артур

27 ноября 2025 г.

План

Функциональное программирование
- Чистые функции
- Функции высшего порядка
- Неизменяемость
- Рекурсия

Объектно-ориентированное программирование
- Классы и объекты
- Наследование
- Полиморфизм
- Инкапсуляция

Аспектно-ориентированное программирование
- Аспекты
- Точки соединения
- Советы
- Примеры в Scala

Функциональное программирование: Чистые функции

Чистые функции возвращают значение без побочных эффектов

```
def add(a: Int, b: Int): Int = a + b
println(add(3, 5)) // Вывод: 8

// Дополнительный пример
def multiply(x: Int, y: Int): Int = x * y
println(multiply(4, 2)) // Вывод: 8
```

Функциональное программирование: Функции высшего порядка

Функции, принимающие или возвращающие другие функции

```
def applyFunc(f: Int => Int, x: Int): Int = f(x)
val double = (n: Int) => n * 2
println(applyFunc(double, 5)) // 10

// Дополнительно: map
val list = List(1, 2, 3)
println(list.map(double)) // List(2,4,6)
```

Функциональное программирование: Неизменяемость

Данные не изменяются после создания

```
val list = List(1, 2, 3)
// list(0) = 4 // Ошибка
val newList = list :+ 4
println(newList) // List(1,2,3,4)

// Дополнительно: map
val updated = list.map(_ + 1)
println(updated) // List(2,3,4)
```

Функциональное программирование: Рекурсия

Рекурсия вместо циклов

```
def factorial(n: Int): Int = if (n <= 1) 1 else n * factorial(n - 1)
println(factorial(5)) // 120

// Дополнительно: хвостовая рекурсия
@tailrec def sum(list: List[Int], acc: Int = 0): Int = list match {
  case Nil => acc
  case h :: t => sum(t, acc + h)
}
println(sum(List(1,2,3))) // 6
```

Объектно-ориентированное: Классы и объекты

Классы определяют структуру объектов

```
class Person(val name: String)
val p = new Person("Alice")
println(p.name) // Alice

// Дополнительно
class Point(x: Int, y: Int)
val pt = new Point(0, 0)
```

Объектно-ориентированное: Наследование

Наследование для повторного использования кода

```
class Employee(name: String, val salary: Int) extends Person(name)
val e = new Employee("Bob", 50000)
println(e.name) // Bob

// Дополнительно
println(e.salary) // 50000
```

Объектно-ориентированное: Полиморфизм

Полиморфизм позволяет использовать подтипы

```
val person: Person = new Employee("Charlie", 60000)
println(person.name) // Charlie

// Переопределение метода
override def toString: String = s"$name earns $salary"
println(person) // Charlie earns 60000
```

Объектно-ориентированное: Инкапсуляция

Инкапсуляция скрывает детали реализации

```
class BankAccount(private var balance: Double) {
  def deposit(amount: Double): Unit = balance += amount
  def getBalance: Double = balance
}
val acc = new BankAccount(100.0)
acc.deposit(50.0)
println(acc.getBalance) // 150.0
```

Аспектно-ориентированное: Аспекты

Аспекты для кросс-каттинговых concerns

```
// Пример с использованием trait для logging
trait Logging {
  def log(msg: String): Unit = println(msg)
}

// Аспект как mixin
```

Аспектно-ориентированное: Точки соединения

Join points - места в коде для применения аспектов

```
class Service {
  def doSomething(): Unit = println("Doing something")
}

// Точка - вызов метода doSomething
```

Аспектно-ориентированное: Советы

Advice: before, after, around

```
class LoggedService extends Service with Logging {
  override def doSomething(): Unit = {
    log("Before")
    super.doSomething()
    log("After")
  }
}
val ls = new LoggedService()
ls.doSomething() // Вывод: Before, Doing something, After
```

Аспектно-ориентированное: Примеры в Scala

Использование mixin для AOP-подобного поведения

```
trait Timing {
  def timed[T](block: => T): T = {
    val start = System.currentTimeMillis()
    val res = block
    val end = System.currentTimeMillis()
    println(s"Took ${end - start} ms")
    res
  }
}
class TimedService extends Service with Timing {
  override def doSomething(): Unit = timed { super.doSomething() }
}
```

Вывод и литература

- Функциональное: фокус на функциях и неизменяемости
- OOP: фокус на объектах и инкапсуляции
- AOP: фокус на разделении concerns

Список литературы:
- https://en.wikipedia.org/wiki/Functional_programming
- https://en.wikipedia.org/wiki/Object-oriented_programming
- https://en.wikipedia.org/wiki/Aspect-oriented_programming
- https://docs.scala-lang.org/

Спасибо за внимание!