

Paradigma Orientata Obiect

Cursul nr. 3
Mihai Zaharia

Proiectarea orientată obiect (detalii la curs Lupu)

O metodologie orientată pe rezolvarea problemei care produce o soluție a problemei în termeni de entități încapsulate numite obiecte.

Obiect

O entitate sau un lucru care are sens în contextul problemei.

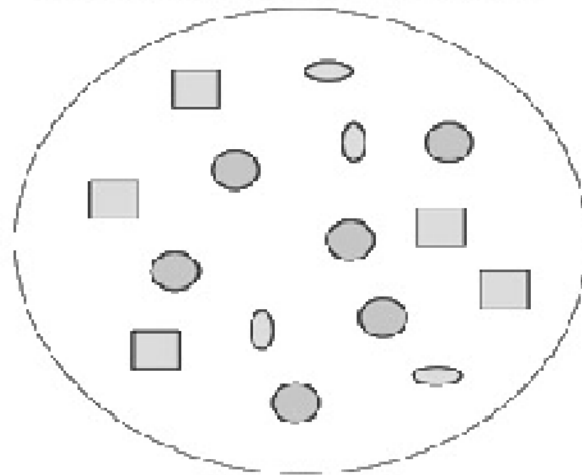
De exemplu un student, o mașină, o dată și o oră

Problemele sunt rezolvate prin:

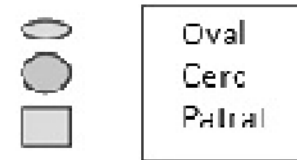
1. Izolarea obiectelor implicate
2. Determinarea proprietăților și acțiunilor (sau responsabilităților) acestora și
3. Descrierea colaborării între obiecte cu scopul rezolvării problemei

Fazele pentru rezolvarea și implementarea problemei în OO

Problema Spațiului de obiecte



De la Abstractizare spre Clase
(descrierea obiectelor)



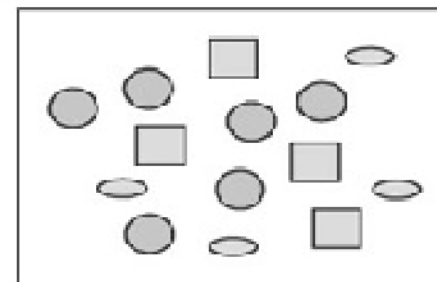
1. Faza de rezolvare a problemei

Definirea tipurilor claselor



2. Faza de implementare

Spațiul program al obiectelor



Variable **var**&**val**

- declarare & initializare variabila RO

```
val name = "kotlin"
```

- declarare & initializare variabila normala

```
var name: String name = "kotlin"
```

Inferența de tip

Deși kotlin este un limbaj cu tipuri tari de date el nu necesită declararea obligatorie de tip deci ca și python suportă inferența de tip.

```
fun plusOne(x: Int) = x + 1
```

Câteodată este util să lucrăm explicit:

```
val explicitType: Number = 12.3
```

Tipuri de date

- Ca și în Python în Kotlin orice este un obiect

NUMERE

```
val int = 123
```

```
val long = 123456L
```

```
val double = 12.34
```

```
val float = 12.34F
```

```
val hexadecimal = 0xAB
```

```
val binary = 0b01010101
```

Conversii implicite?

```
val int = 123
```

```
val long = int.toLong()
```

```
val float = 12.34F
```

```
val double = float.toDouble()
```

Tip	Dim
<i>Long</i>	64
<i>Int</i>	32
<i>Short</i>	16
<i>Byte</i>	8
<i>Double</i>	64
<i>Float</i>	32

toByte(),
toShort(),
toInt(),
toLong(),
toFloat(),
toDouble(),
toChar().

Operatori pe biți

- Nu sunt definiți ca operatori speciali dar pot fi apelați ca atare
- `val leftShift = 1 shl 2`
- `val rightShift = 1 shr 2`
- `val unsignedRightShift = 1 ushr 2`
- `val and = 1 and 0x00001111`
- `val or = 1 or 0x00001111`
- `val xor = 1 xor 0x00001111`
- `val inv = 1.inv()`

Variable logice (bool)

- `val x = 1 val y = 2 val z = 2`
- `val isTrue = x < y && x < z`
- `val alsoTrue = x == y || y == z`

Caractere

- Sunt clasice cu simple ghilimele și suportă caracterele de control standard - \t, \b, \n, \r, ', ", \\, \\$.

Șiruri de caractere

- `val string = "string with \n new line"`
- mai există ceva numit șir brut(`raw`)

Tablouri

- `val array = arrayOf(1, 2, 3)`
- `val perfectSquares = Array(10, { k -> k * k })`
- `val element1 = array[0]` `val element2 = array[1]` `array[2] = 5`

Tablouri cu tip

- `ByteArray`, `CharArray`, `ShortArray`, `IntArray`, `LongArray`, `BooleanArray`, `FloatArray`, and `DoubleArray`

• Exemple inițializări variabile

- `val aToZ = "a".."z"`
- `val isTrue = "c" in aToZ`
- `val oneToNine = 1..9`
- `val isFalse = 11 in oneToNine`

Cicluri

```
while (true) {println("This will print out for a long time!")}
```

```
val list = listOf(1, 2, 3, 4) for (k in list) { println(k)}
```

```
val set = setOf(1, 2, 3, 4) for (k in set) { println(k)}
```

```
val oneToTen = 1..10 for (k in oneToTen) {
```

```
    for (j in 1..5)
```

```
        {println(k * j) }
```

```
}
```

operator element hasNext(): Boolean

operator element next(): T

```
val string = "print my characters" for (char in string) { println(char)}
```

```
for (index in array.indices) {println("Element $index is ${array[index]}")}
```


Gestiunea exceptiilor

```
fun readFile(path: Path): Unit
{
    val input = Files.newInputStream(path)
        try
        {
            var byte = input.read()
            while (byte != -1)
                { println(byte) byte = input.read() }
        } catch (e: IOException)
        {
            println("Error reading from file. Error was ${e.message}")
        }
        finally
        { input.close() }
}
```

Instanțierea unei clase

```
val file = File("/etc/nginx/nginx.conf")
```

```
val date = BigDecimal(100)
```

Egalitatea de referință și de structură

- pentru egalitatea de referință vom folosi `===` sau `!==`
- exemplu de gândire greșită:
- `val a = File("/mobydick.doc")`
- `val b = File("/mobydick.doc")`
- `val sameRef = a === b` //va fi False
- pentru egalitatea de structură vom folosi `==` sau `!=`
- `val a = File("/mobydick.doc")`
- `val b = File("/mobydick.doc")`
- `val structural = a == b` //va fi True

This

- `class Person(name: String)`
- `{ fun printMe() = println(this) }`
- i se mai spune și “current receiver”

Scope

```
class Building(val address: String)
{
    inner class Reception(telephone: String)
        { fun printAddress() = println(this@Building.address) }
}
```

Vizibilitate

Public

Private

class Person

{ private fun age(): Int = 21 }

Protected

Internal

internal class Person

{ fun age(): Int = 21 }

Controlul ... fluxului de execuție ca expresie

- "hello".startsWith("h")
- val a = 1

```
public boolean isZero(int x)
{
    boolean isZero;
    if (x == 0)
        isZero = true;
    else
        isZero = false;
    return isZero;
}
```

Controlul ... fluxului de execuție

```
val date = Date()
val today = if (date.year == 2019) true
              else false
fun isZero(x: Int): Boolean
{
    return if (x == 0) true
           else false
}
```

O abordare similară poate fi folosită și pentru blocurile try..catch

```
val success = try {
    readFile() true
} catch (e: IOException)
{ false }
```

Null

- `var str: String? = null`

NULL SAFETY!!!!

Nullable and non-nullable types

- `val name: String = null // grr...errr`
- `var name: String = "mike"`
- `name = null // grr...errr`
- `val name: String? = null // i'mm happy`
- `var name: String? = "harry"`
- `name = null // i'mm happy`
- `fun name1(): String = ... fun name2(): String? = ...`

Verificarea și conversia de tip

```
fun isString(any: Any): Boolean  
{ return      if (any is String) true  
              else false }
```

- În Java cu cast explicit

```
public void printStringLength(Object obj) //Object superior în ierarhie  
{ if (obj instanceof String)  
    {String str = (String) obj System.out.print(str.length())} }
```

- În Kotlin

```
fun printStringLength(any: Any)  
{ if (any is String)  
    { println(any.length)} }
```


Conversia explicită de tip

- `fun length(any: Any): Int`
- `{ val string = any as String return string.length }`
- `val string: String? = any as String`
- atunci:
- `val any = "/home/mike"`
- `val string: String? = any as String`
- `val file: File? = any as File`

When ca switch case (cu argument)

- cu else pe post de default

```
fun whatNumber(x: Int)
{ when (x)
  { 0      -> println("x is zero")
    1      -> println("x is 1")
    else   -> println("X is neither 0 or 1") }
}
fun isMinOrMax(x:Int): Boolean // imbunătățire cod
{
  val isZero = when (x)
    {Int.MIN_VALUE -> true Int.MAX_VALUE -> true else -> false}
  return isZero
}
```

When can switch case (cu argument)

```
fun isZeroOrOne(x:Int): Boolean //cod profesional
{ return when (x)
    { 0, 1 -> true
      else -> false }    }
fun isSingleDigit(x:      Int): Boolean // cu interval
{ return when (x)
    {in -9..9 -> true else -> false }  }
fun startsWithFoo(any: Any): Boolean // cu smart case
{
return when (any)
    { is String -> any.startsWith("Foo") else -> false }
}
```

When ca expresie

```
fun whenWithoutArgs(x:Int, y:Int) //ex1
```

```
{
```

```
when { x < y -> println("x is less than y") x > y -> println("X is  
greater than y") else -> println("X must equal y") }
```

```
}
```

```
when { //ex2
```

```
    x.isOdd() -> print("x is odd")
```

```
    x.isEven() -> print("x is even")
```

```
    else -> print("x is stupid")
```

```
}
```

Transmiterea rezultatelor unei funcții

```
fun largestNumber(a: Int, b: Int, c: Int): Int //ex1
```

```
{ fun largest(a: Int, b: Int): Int
    { if (a > b)
        return a
      else
        return b }
  return largest(largest(a, b), largest(b, c)) }
```

```
fun printUntilStop() //ex 2
```

```
{ val list = listOf("a", "b", "stop", "c")
  list.forEach stop@
    { if (it == "stop") return@stop
      else println(it) } }
```

```
fun printUntilStop() //ex 3
```

```
{ val list = listOf("a", "b", "stop", "c")
  list.forEach
    { if (it == "stop") return@forEach
      else println(it) } }
```

Clasa

```
class Person constructor(val firstName: String, val lastName: String,  
val age: Int?) {}  
fun main(args: Array<String>)  
{ val person1 = Person("Alex", "Smith", 29)  
  val person2 = Person("Jane", "Smith", null)  
  println("${person1.firstName},${person1.lastName} is ${person1.age}  
years old")  
  println("${person2.firstName},${person2.lastName} is  
${person2.age?.toString() ?: "?"} years old")  
}
```

Clasa - constructor cu cod în el

```
class Person (val firstName: String, val lastName: String, val age: Int?)  
{ init {  
    require(firstName.trim().length > 0)  
        { "Invalid firstName argument." }  
    require(lastName.trim().length > 0)  
        { "Invalid lastName argument."}  
    if (age != null)  
        { require(age >= 0 && age < 150)  
            { "Invalid age argument." } }  
    } }  
Person p = new Person("Jack", "Miller", 21); //ex2  
System.out.println(String.format("%s, %s is %d age old", p.getFirstName(),  
p.getLastName(), p.getAge()));
```

Class “nested”

```
class Outer //ex1
{ static class StaticNested {}
  class Inner {} }
```

```
class BasicGraph(val name: String)//ex2
{
  class Line(val x1:Int, val y1:Int, val x2: Int, val y2: Int)
  {
    fun draw(): Unit
    {println("Drawing Line from ($x1:$y1) to ($x2, $y2)")}
  }
  fun draw(): Unit
  { println("Drawing the graph $name") }
}
```

- și instanțierea

```
val line = BasicGraph.Line(1, 0, -2, 0)
line.draw()
```


Class nested

```
class BasicGraphWithInner(graphName: String) //ex1
{ private val name: String
    init { name = graphName }
    inner class InnerLine(val x1: Int, val y1: Int, val x2: Int, val y2: Int)
        { fun draw(): Unit { println("Drawing Line from ($x1:$y1) to ($x2, $y2) for graph$name ") } }
    fun draw(): Unit { println("Drawing the graph $name") }
}

class A //ex2 this@label
{ private val somefield: Int = 1
    inner class B
        {
            private val somefield: Int = 1
            fun foo(s: String)
                { println("Field <somefield> from B" + this.somefield)
                  println("Field <somefield> from B" + this@B.somefield)
                  println("Field <somefield> from A" + this@A.somefield) }
        }
}
```

Class anonime

```
class Controller
{
    private var clicks:Int=0
    fun enableHook()
    {
        button.addMouseListener(object : MouseAdapter()
            {override fun mouseClicked(e: MouseEvent)
                {clicks++}
            })
    }
}
```

Clase pentru gestiune date

Data classes

```
data class Customer(val id:Int, val name:String, var address:String)
```

Enum classes

```
enum class Day { MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
```

```
SATURDAY, SUNDAY} //ex 1
```

```
public enum class Planet(val mass: Double, val radius: Double)  
{ MERCURY(3.303e+23, 2.4397e6), VENUS(4.869e+24, 6.0518e6),  
EARTH(5.976e+24, 6.37814e6), MARS(6.421e+23, 3.3972e6),  
JUPITER(1.9e+27, 7.1492e7), SATURN(5.688e+26,  
6.0268e7),URANUS(8.686e+25, 2.5559e7), NEPTUNE(1.024e+26,  
2.4746e7); }//ex 2
```

```
Planet.valueOf("JUPITER") //ex 3
```

```
Planet.values()// exx4
```

Clase pentru gestiune avansată de date

```
interface Printable
```

```
{ fun print(): Unit }
```

```
public enum class Word : Printable
```

```
{
```

```
    HELLO {
```

```
        override fun print() {println("Word is HELLO")}
```

```
    },
```

```
    BYE {
```

```
        override fun print() { println("Word is BYE")}
```

```
    }
```

```
}
```

```
val w= Word.HELLO
```

```
w.print()
```