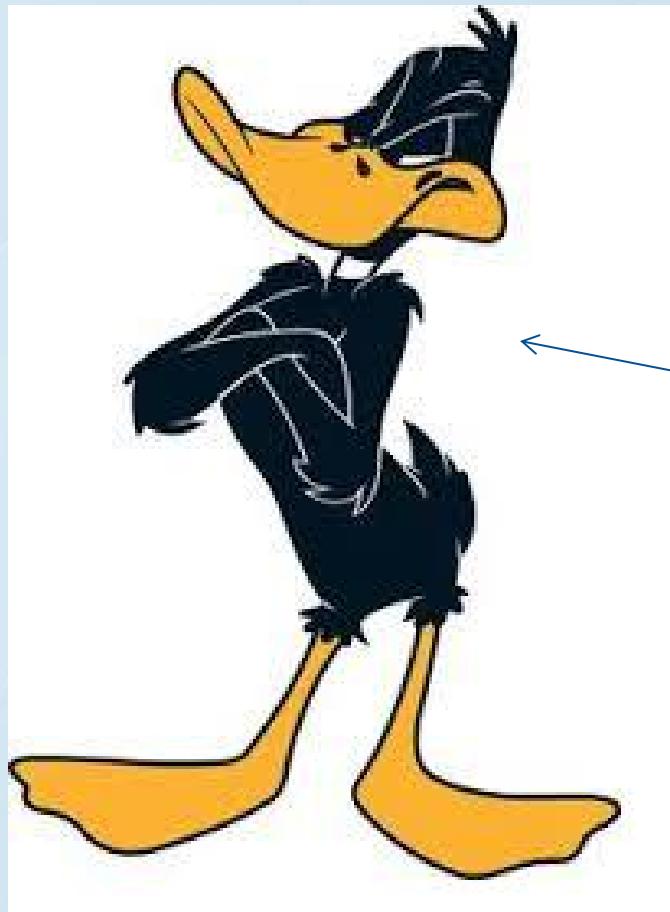


Paradigma de Programare



Titular
M. Mihai Zaharia

Observații generale

- Linux?
- boot "bare metal"
- Aspecte generale cu privire la examinare
 - teorie
 - lab
- Bologna vs Harețian?

Cum este cu înregistrările????

- No way!!!
- Why?

Laptop recomandat

- minim 16 GB RAM(daca are linux)
- minim 20 (daca are win in special 10)
- ideal 32 GB
- evitati procesoarele cu extensia M,LV etc
- preferabil cele cu HQ
- minim I5 (dar de ultima gen)
- recomandat tastură luminată
- recomandat I7 sau I9
- 2 hdd-uri din care cel de boot si os obligatoriu SSD
- la limita merge si un SSHD dar prost (pentru cei care nu pot baga al doilea hdd) -
 - min Mon 15,6 inch glossy/mate
 - verificati ca ecranul are buna vizibilitate si in soare direct
 - preferabil cu placa retea intel in extremis qualcomm/atheros
 - firme low cost - acer deoarece au un program de proiectare bios/hw care tine cont de linux
- nu este obligatorie placa video performanta - papa baterie si cam atat daca nu ai ce face dezactivati-o din bios sau in linux din bumble bee (dar preferabil nvidia pt cuda)

Obiectivele cursului

1. Limbajele care vor fi folosite de-a lungul cursului sunt:

1. *UML*
2. *C/C++*
3. *Java Script*
4. *Python*
5. *Kotlin*
6. *R*
7. *Polyglot*
8. *Prolog*

Criterii folosite în evaluarea activității

- **Participarea:** la orele de curs și de laborator:
 - Neparticiparea la mai mult de 50% din laboratoare conduce la refacerea disciplinei.
 - neparticiparea la curs conduce la probleme la examenul teoretic
- **Laboratoare:** pentru a putea lua 10 la laborator trebuie ca studentii să fie capabili la intrebarile asistenților cu privire la conținutul cursului curent (și pentru care a fost creat laboratorul) - 30% - atenție asistentii nu stau să repredea!! ci numai notează corectitudinea răspunsului și trec mai departe
- **Examen final 70%** (este o singura notă) defalcată astfel:
 - Proba de **laborator** – **ELIMINA-TORIE 40 %** cu biletă și două ore maxim la dispoziție. Un subiect din două trebuie să fie îndeplinit integral pentru a se putea nota (min 5).
 - Proba **teoretică** – “**PICĂ-TORIE**” **40%** - test docimologic - conține și întrebări cu caracter practic specifice laboratorului (min 5)
 - **Teme acasă: semi** - “**PICĂ-TORIE**” la majoritatea laboratoarelor **20%** (atenție se poate să aveți 5 la lab și teorie și să picați din teme nepredate)
- În caz de variantă on line (SARS-CoV-II)
 - testul practic - se aleg automat două probleme din pool și se trimit
 - testul teoretic - oral cu întrebări selectate automat din pool

Îndatoririle studentului

1. **Citiți materialele** recomandate cu o zi înainte de fiecare curs – elaborați o lista de întrebări eventual.
2. **Rezolvați temele** pentru acasă în săptămâna în care le-ați primit.
3. **Participați la cursuri** (suportul de curs livrat va conține uneori numai desenele/cod din slide-uri).
4. **Rezervați un minim de 2-4 ore** de studiu individual pe săptămâna pentru aceasta materie.
5. **Verificați înțelegerea teoretică și practică** a noțiunilor asimilate prin ajutarea colegilor cu răspunsuri legate de partea teoretică sau ajutor (NU tema copiată) în rezolvarea problemelor practice.

TIOBE Programming Community Index for 2017

Feb 2017	Feb 2016	Programming Language	Ratings	Change
1	1	Java	16.676%	-4.47%
2	2	C	8.445%	-7.15%
3	3	C++	5.429%	-1.48%
4	4	C#	4.902%	+0.50%
5	5	Python	4.043%	-0.14%
6	6	PHP	3.072%	+0.30%
7	9	JavaScript	2.872%	+0.67%
8	7	Visual Basic .NET	2.824%	+0.37%
9	10	Delphi/Object Pascal	2.479%	+0.32%
10	8	Perl	2.171%	-0.08%
11	11	Ruby	2.153%	+0.10%
12	16	Swift	2.125%	+0.75%
13	13	Assembly language	2.107%	+0.28%
14	38	Go	2.105%	+1.81%
15	17	R	1.922%	+0.73%
16	12	Visual Basic	1.875%	+0.02%
17	18	MATLAB	1.723%	+0.63%
18	19	PL/SQL	1.549%	+0.49%
19	14	Objective-C	1.536%	+0.13%
20	23	Scratch		

TIOBE Programming Community Index for 2018

<https://www.tiobe.com/tiobe-index/>

Feb 2019	Feb 2018	Change	Programming Language	Ratings	Change
1	1		Java	15.876%	+0.89%
2	2		C	12.424%	+0.57%
3	4	▼	Python	7.574%	+2.41%
4	3		C++	7.444%	+1.72%
5	6		Visual Basic .NET	7.095%	+3.02%
6	8		JavaScript	2.848%	-0.32%
7	5		C#	2.846%	-1.61%
8	7		PHP	2.271%	-1.15%
9	11		SQL	1.900%	-0.46%
10	20		Objective-C	1.447%	+0.32%
11	15		Assembly language	1.377%	-0.46%
12	19		MATLAB	1.196%	-0.03%
13	17		Perl	1.102%	-0.66%
14	9		Delphi/Object Pascal	1.066%	-1.52%
15	13		R	1.043%	-1.04%
16	10		Ruby	1.037%	-1.50%
17	12		Visual Basic	0.991%	-1.19%
18	18		Go	0.960%	-0.46%
19	49		Groovy	0.936%	+0.75%
20	16		Swift	0.918%	-0.88%



Nov 2022

Nov 2021

Change

Products
ProgrammingQuality Models
LanguageMarkets
RatingsSchedule a demo
Change

1	1		Python	17.18%	+5.41%
2	2		C	15.08%	+4.35%
3	3		Java	11.98%	+1.26%
4	4		C++	10.75%	+2.46%
5	5		C#	4.25%	-1.81%
6	6		Visual Basic	4.11%	-1.61%
7	7		JavaScript	2.74%	+0.08%
8	8		Assembly language	2.18%	-0.34%
9	9		SQL	1.82%	-0.30%
10	10		PHP	1.69%	-0.12%
11	18		Go	1.15%	-0.06%
12	15		R	1.14%	-0.14%
13	11		Classic Visual Basic	1.10%	-0.46%
14	17		Delphi/Object Pascal	1.08%	-0.14%
15	20		MATLAB	1.02%	-0.15%

Ergonomie

1. Alegere monitor – mat
2. Dimensiune minima ecran
3. Alegere densitate / rezoluție ecran
4. Reglare luminozitate și culoarea albastră
5. Poziție corectă de lucru – desktop/laptop
6. Sporturi recomandate

Criterii generale de proiectare a unui limbaj

- Putere (excelează în rezolvarea...)
- Flexibilitate
- Expresivitate
- Ușor de scris (vezi C vs Pascal)
- Implementare eficientă
- Support pentru abstractizări

- Simplitate
- Claritate
- Consistență (ortogonalitate)
(puține structuri de control cu puține posibile combinații)
- Usurință în urmărirea cod
- Aplicabilitatea în domeniul problemei (>=gen4)
- Portabilitate

Definirea unui limbaj

- **Sintaxa:** se definește gramatica unui limbaj
 - Ce înseamnă o propoziție corectă? dar un program corect?
 - De obicei se folosesc notații formale ca BNF sau forma sa extinsă EBNF.
- **Semantica:** interesul elementelor limbajului
 - De obicei a fost legat de limbajele naturale (umane)
 - Notații formale există dar nu sunt foarte folosite

Sintaxa

- Definește simbolurile și gramatica unui limbaj
 - BNF sau
 - EBNF

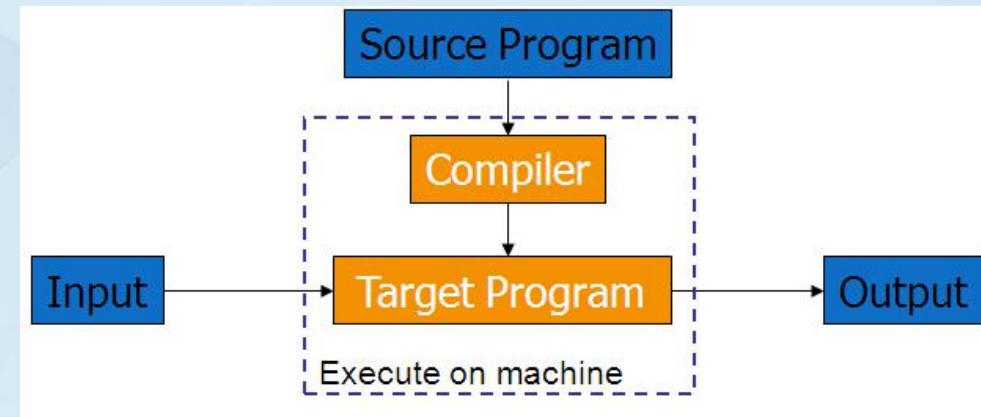
*if-statement ::= if (expression) statement-block
[else statement-block]*

statement-block ::= statement ';' | '{' statement ';' [...] '}'

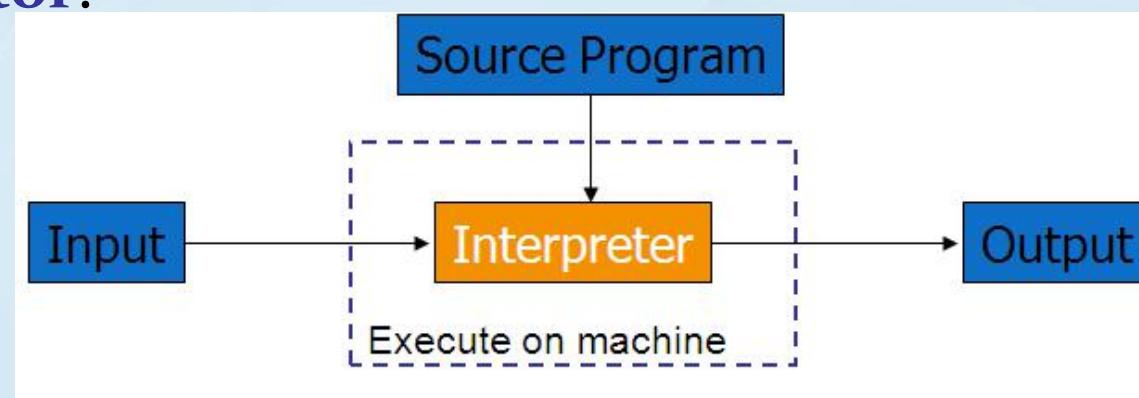
*statement ::= if-statement | assignment-statement |
while-statement | ...etc...*

Strategii de implementare

- **Compilator:**

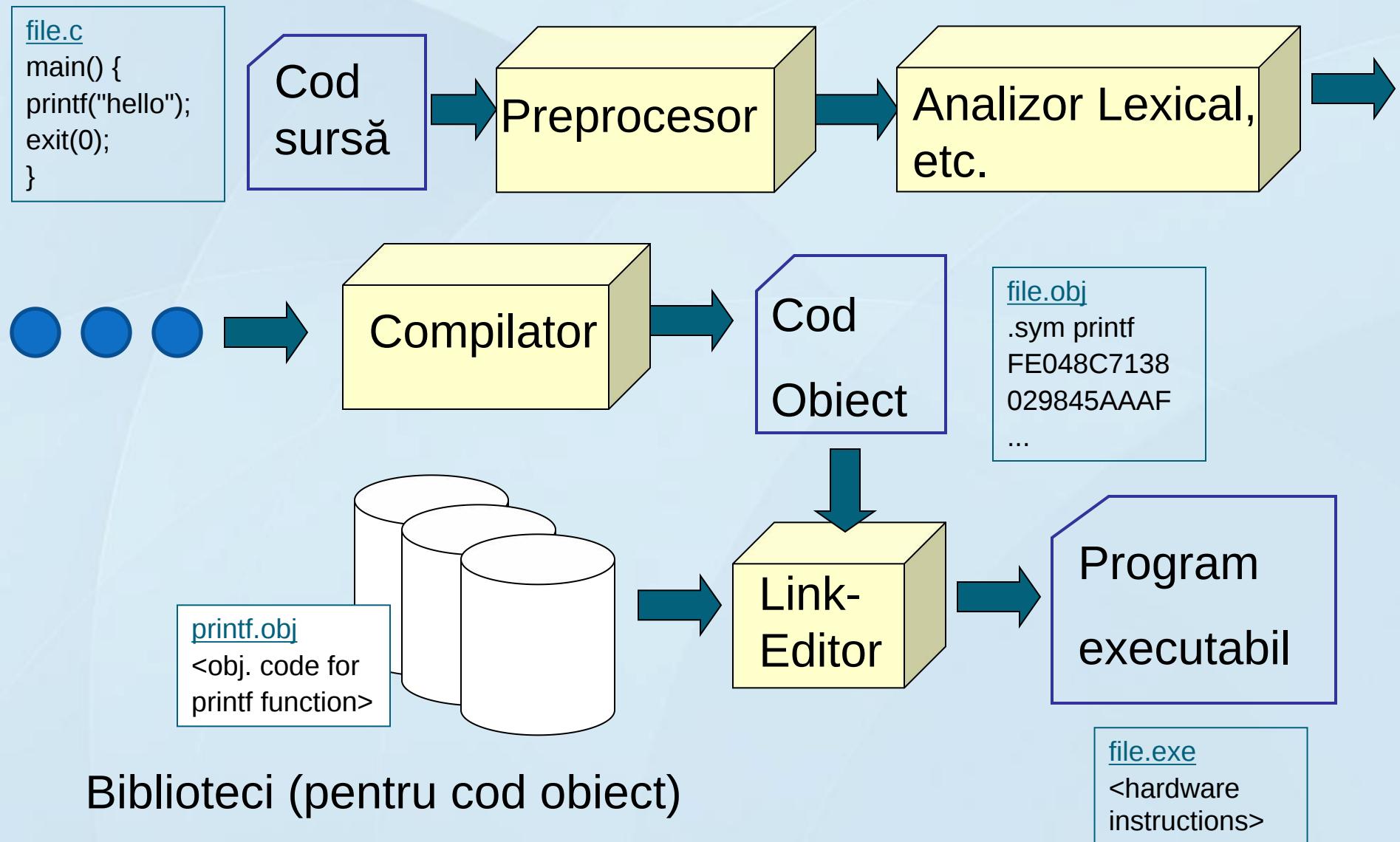


- **Interpreter:**



- **Hibrid:**

Compilarea unui program



Etape ale compilării

Program
Sursă

Analizor Lexical

Codul este convertit în expresii cu simboluri speciale și analizat

Analizor Sintactic/Parser

Se verifică analiza sintactică și atât

Analizor Semantic

Se adaugă informație semantică peste arborele de parsare și se crează tabelul de simboluri

Generator de cod intermediar
(via assembly sau nu)

Optimizare cod mem/speed

Generare cod

Program
țintă

Program Interpretat vs Compilat

Interpretat

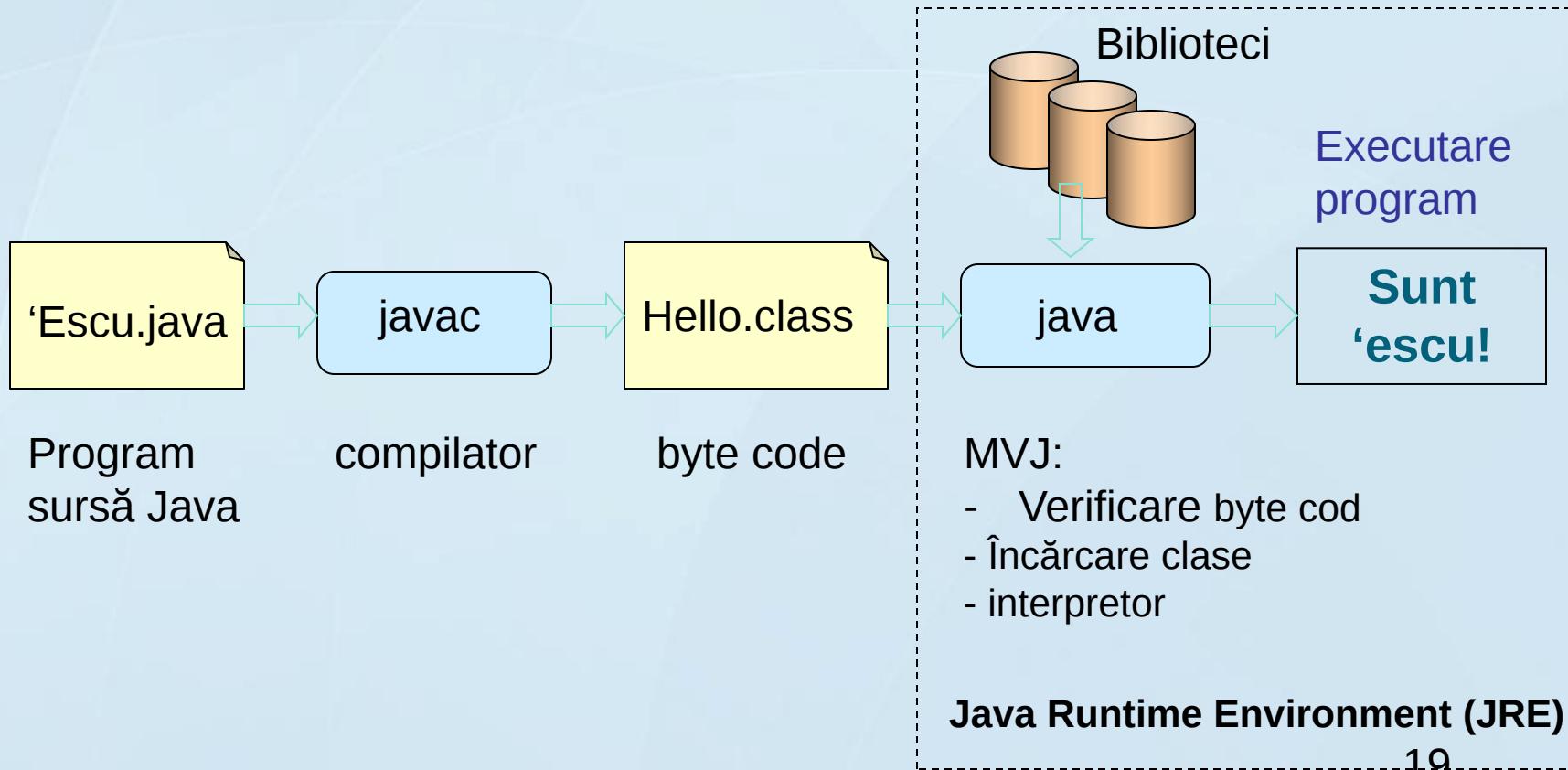
- Flexibil
- Interactiv
- Comportament dinamic mai complex
- Dezvoltare rapidă
- Programul poate fi executat imediat ce este scris/modificat
- Portabil pe orice mașină cu interpretor

Compilat

- Execuție mult mai eficientă (Java vs .Net vs. C++)
- Analiza datelor este mai amănunțită
- Mai structurat
- De obicei mai scalabil (aplicații mari dimensiuni)
- Trebuie recompilat programul după fiecare modificare
- Trebuie recompilat pentru orice diferență în OS sau HW a mașinii țintă

Java: o strategie hibridă

- Compilator Java: crează un byte code independent de mașină
- Mașina Virtuală Java (Interpreter): execută acel cod.



Classificare Erori

- **Lexicale**: erori la nivel de token, cum ar fi caractere ilegale (greu de distins din erorile de sintaxă).
- **De sintaxă**: erori gramaticale (e.g. “;” lipsa sau cuvânt cheie).
- **Statice de semantică**: care se pot detecta Înainte de execuție (variabile nedefinite, erori de tip)
- **Dinamice și de logică**

Observații cu privire la generarea erorilor

- **Un compilator** va raporta erori lexicale, de sintaxă și pe cele statice. Nu va raporta pe cele dinamice de semantică.
- **Un interpretor** de obicei va raporta erori de sintaxă și lexicale.
- Nici un **translator** nu va raporta o eroare logică.

Depanare pas cu pas???

- oare este necesara?
- **DA**
- nu confundați cu sari la ... și execută

Java code/bytocode pentru JVM

- outer:
- for (int i = 2; i < 1000; i++)
- {
- for (int j = 2; j < i; j++)
- {
- if (i % j == 0) continue outer;
- }
- System.out.println (i);
- }

0: iconst_2 //începe for
1: istore_1
2: iload_1
3: sipush 1000
6: if_icmpge 44 //comparatia cu 1000
9: iconst_2 //începe for
10: istore_2
11: iload_2
12: iload_1
13: if_icmpge 31 // comparația cu i
16: iload_1 //începe if
17: iload_2
18: irem
19: ifne 25 // comparatia cu 0
22: goto 38
25: iinc 2, 1 //j++
28: goto 11
31: getstatic #84; //apel PrintStream
34: iload_1
35: invokevirtual #85;
//PrintStream.println:(I)V
38: iinc 1, 1 // i++
41: goto 2
44: return

Probleme specifice limbajului de asamblare

Programarea este dificilă chiar folosind limbajul macro de asamblare (cu .small etc)

Limbajul nu se potriveste cu maniera în care gandesc oamenii

Programele sunt lungi și dificil de înțeles (dacă nu ai experienta și le scrii cu picioarele)

Erorile de logică în program

Limbajul este dependent de mașină

Primele încercări de abstractizare

- Fortran, primul limbaj de nivel înalt,

```
PROGRAM EXEMPLU      ! VERSION 0.0.  
CALL HELLO !apelarea subrutinei HELLO.  
CONTAINS  !CONTAINS încheie program principal  
si incepe definirea subrutinelor  
SUBROUTINE HELLO  !definirea corp  
SUBROUTINA  
WRITE(*,*)  "HELLO WORLD!" ! afisez "HELLO  
WORLD!" la CONSOLA.  
END SUBROUTINE HELLO ! Sfarsit SUBROUTINA  
END PROGRAM EXEMPLU !sfarsit PROGRAM
```

```
@echo off cls  
echo Press any key to start  
AProgram.exe!  
pause > nul  
AProgram.exe %1  
if errorlevel 1  
    goto error  
    echo AProgram has  
finished  
whatever it was doing.  
    goto end  
error: echo Something went  
        wrong with Aprogram  
end
```

Limbaje (foste) de nivel înalt

A treia generație de limbaje, cunoscute și sub denumirea de limbaje de nivel înalt oferă următoarele avantaje:

Sintaxa în limba engleză

Nume descriptive pentru a reprezenta datele

Reprezentare concisă a logicii

Folosirea simbolurilor matematice standard

`total = quantity * price * (1 + taxrate);`

Operatori distincți pentru execuția condiționată a buclelor și expresiilor

`if (total > 0) then writeln("The total is ", total)
else writeln("No sale.");`

Apariția de unități funcționale ca funcții sau clase cu izolarea variabilelor:

`radius = sqrt(x*x + y*y);`

Beneficii oferite de a treia generație

- Programarea este mai ușoară și mai rapidă (yesss!)
- Programatorul gândește la un nivel mai abstract rezolvarea problemei (deci un matematician/ informatician poate scrie cod) fără a ști asamblare
- *Apare independența de masină*
- Se poate aplica la orice nivel o abordare de tip top down, bottom up etc
- Testare
- Reutilizare
- Biblioteci

În funcție de tipul de paradigmă folosit în rezolvarea problemelor

- **Imperative**: FORTRAN, COBOL, BASIC, C, Pascal
- **Funcționale**: Scheme, Lisp
- **Limbaje Logice**: Prolog
- **Orientate obiect**:
 - Orientate obiect pure: **Java**, Python
 - Orientate obiect și imperative : C++, Perl, Visual Basic

Limbaje din generația a patra

Acestea sunt deja orientate pe anumite tipuri de aplicatii

SQL pentru baze de date

Limbajul Postscript pentru descrierea unei pagini folosit de imprimante

PDF pentru documente on-line

HTML and PHP pentru continut World Wide Web

Mathematica

Exemple - SQL

- Inserarea unui tabel (table) într-o bază de date:
INSERT INTO angajati(id, Nume, Prenume, functie)
VALUES (1445, 'John','Smith','manager');
- Extragere de informații în funcție de un criteriu dintr-un tabel:
SELECT id, Prenume, salariu **FROM** angajati
WHERE (Job = 'manager');
- SQL este considerat a fi **declarativ**

Factori care influențează dezvoltarea/ selecția limbajelor

- **Performanțele mașinii țintă și a sistemului de operare**
 - **Sistem dedicat ieftin**
 - **Sisteme dedicate medii**
 - **Sisteme dedicate scumpe**
 - **Computere de uz general**
 - **Calcul de mare performanță**
- **Domeniul Aplicației:** limbajul este influențat de tipul de informații care trebuie gestionate
 - Majoritatea limbajelor sunt folosite pentru descriere de algoritmi

Factori care influenteaza dezvoltarea/ selecția limbajelor

- **Metodologia:** o ramură în dezvoltare.
 - Propunerile de noi limbaje în general țin de experiența proiectantului dar și de problema rezolvată (generația 4)
- **Preferinte, Economie, si Patronat:**
 - Limbajele au fost în general dezvoltate de companiile care sunt varf de piață (C de AT&T – Bell Lab, Fortran de IBM, Java de Sun)
 - Pentru necesități guvernamentale (Ada/Clips de către DoD al US).
 - Preferate de experti:

Muncim sau Gândim?

Muncim (caz de studiu)

Se ia o problema concretă

Se citește în fugă (1 min max)

Se scrie un program (30 min)

Se depanează la el (și cîteva zile)

La sfârșit sigur merge prost dacă este testat în caz real

Timp total – nedeterminat

Resurse folosite :
nedeterminat

Gândim (caz de studiu)

Se ia o problema concretă

Se citește cat timp este necesar până la înțelegerea completa

În funcție de specificații se alege tehnologia, modelul de proiectare.

Se proiecteză și se validează aplicația stabilindu-se timpul necesar dezvoltării precum și resursele necesare.

Se trece la implementare

Se face verificarea și depanare

Se trimit versiunea beta pentru testare reală

Timp total (max 25% depășire față de estimarea inițială)

Resurse folosite (linear cu depășirea daca este cazul)

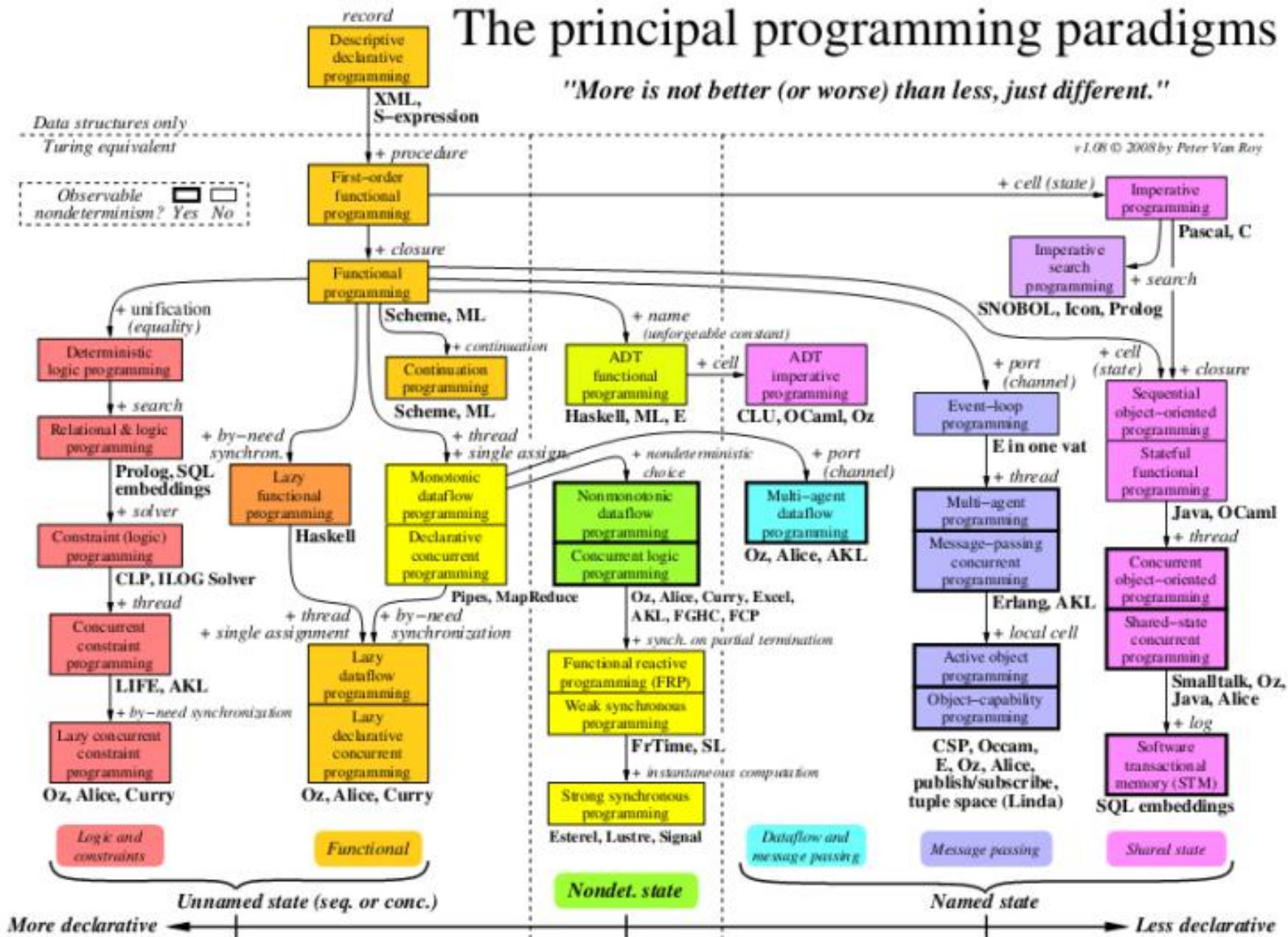
Ce este o paradigmă de programare?

- D₁ (generală) Caz exemplar, **model**, **prototip**, situație ideală, structură tip, **arhetip** standard și.a.
- D₂. (în filozofie, la L.Wittgenstein) Modelele filosofice, acele "tipare" care orientează gândirea noastră în direcții predeterminate.,
- D₃ (în filozofia limbajului) Listă de cazuri tipice de jocuri lingvistice prin care putem înțelege conceptul general..."

The principal programming paradigms

"More is not better (or worse) than less, just different."

v1.08 © 2008 by Peter Van Roy





Holy Grail of compilers

Cursul nr. 2
Mihai Zaharia

Ce înseamnă de fapt programarea?

Written by:

Application
Developer

Language
Developer

VM Expert

OS Expert

Written in:

Guest Language

Managed Host Language

Managed Host Language
or Unmanaged Language

Unmanaged Language
(typically C or C++)

Guest Language Application

Guest Language Implementation

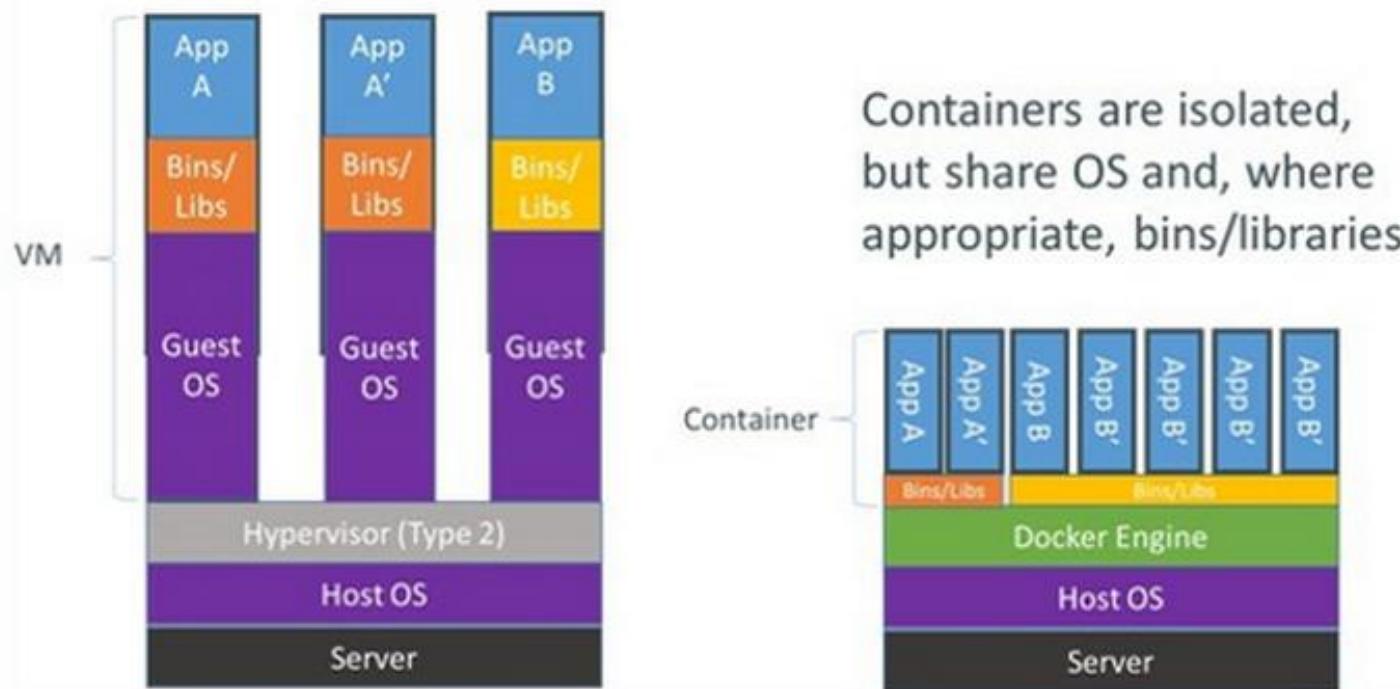
Host Services

OS

Java mai este la modă?

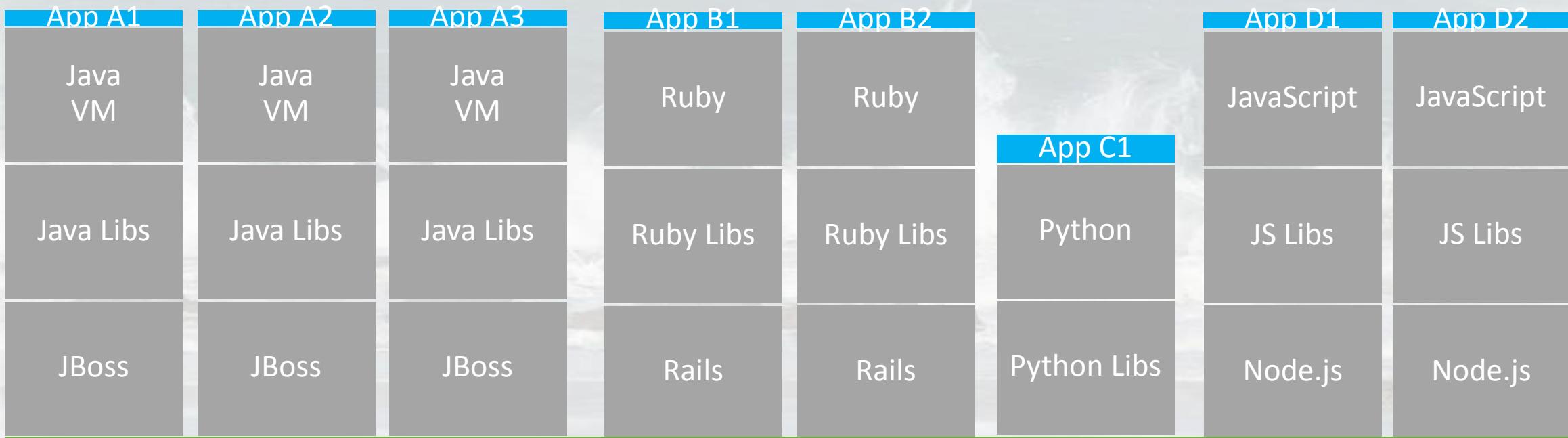
- Încă.... nu se știe cât

Containers vs. VMs



Aplicațiile din containere sunt și ele mari

- “Hello World” în Java: 24 MB
- “Hello World” în JavaScript (V8): 18 MB
- “Hello World” în Ruby/Rails: 8 MB



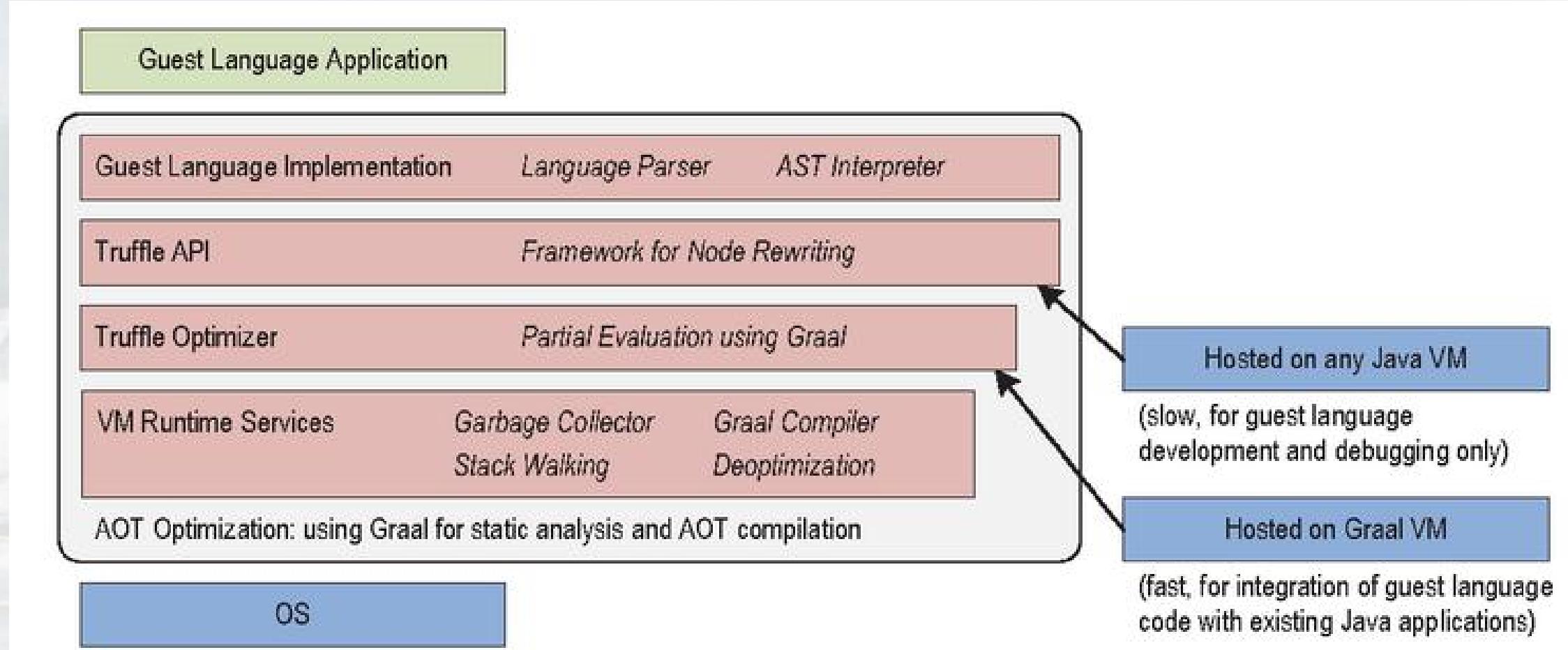
Host Operating System & Container Virtualization

Cum s-a ajuns aici?

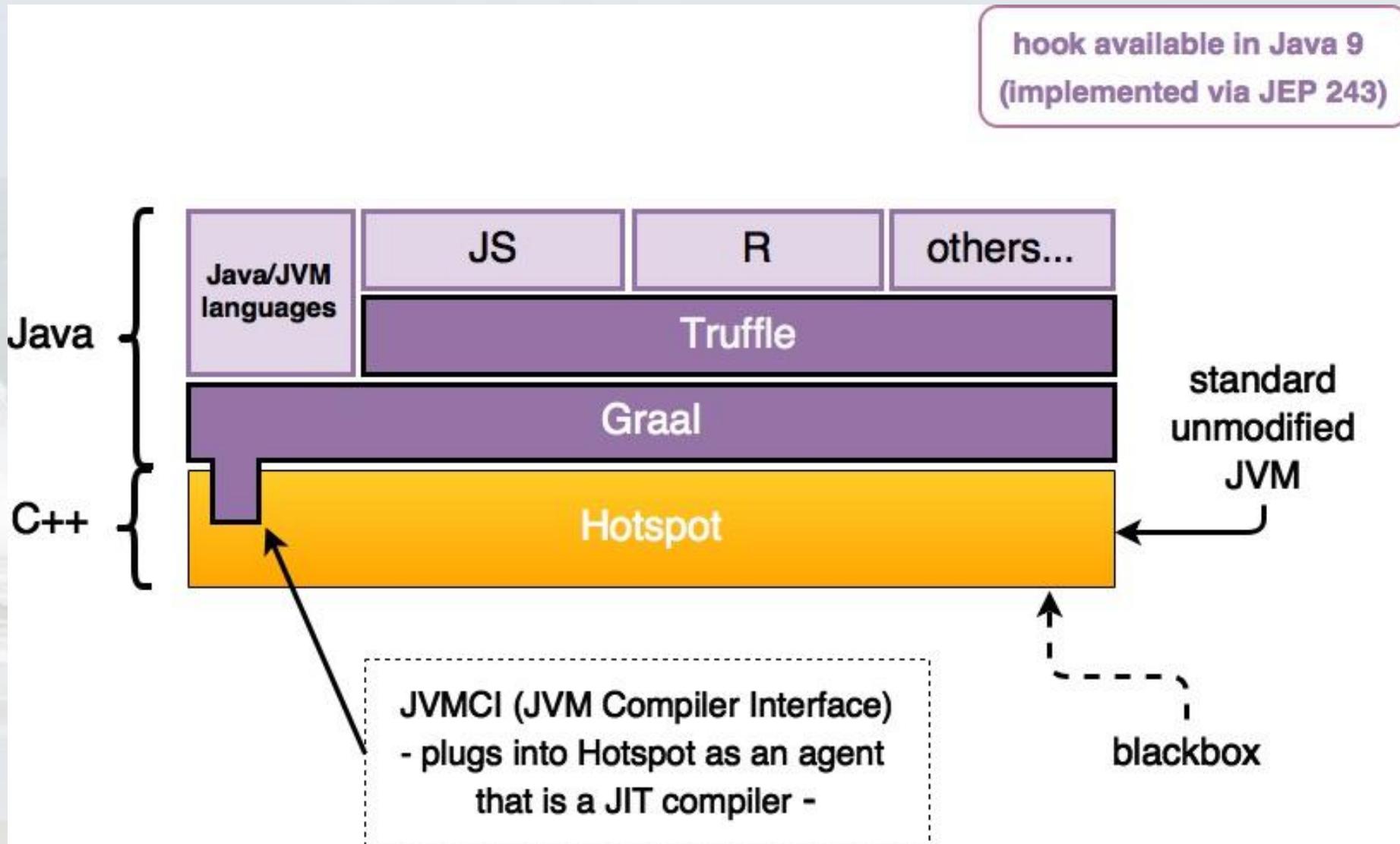
Wiirthinger et al 2013

Oracle Labs - Institute for System Software.
Johannes Kepler University Linz

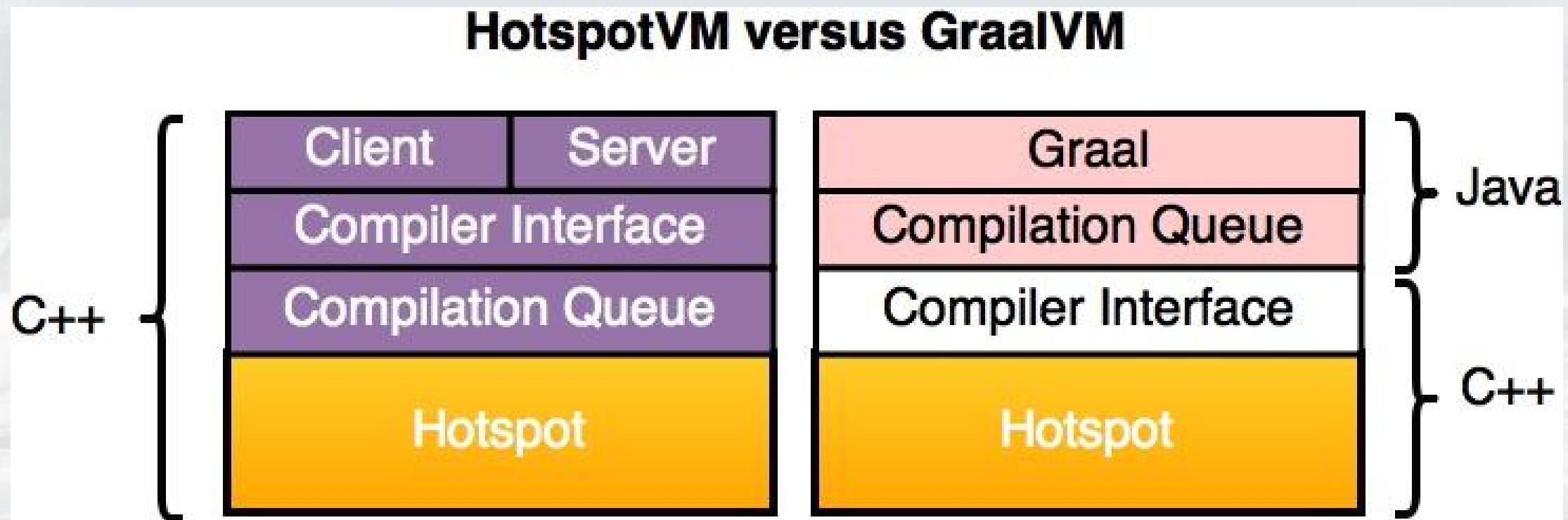
Arhitectura originală/initială



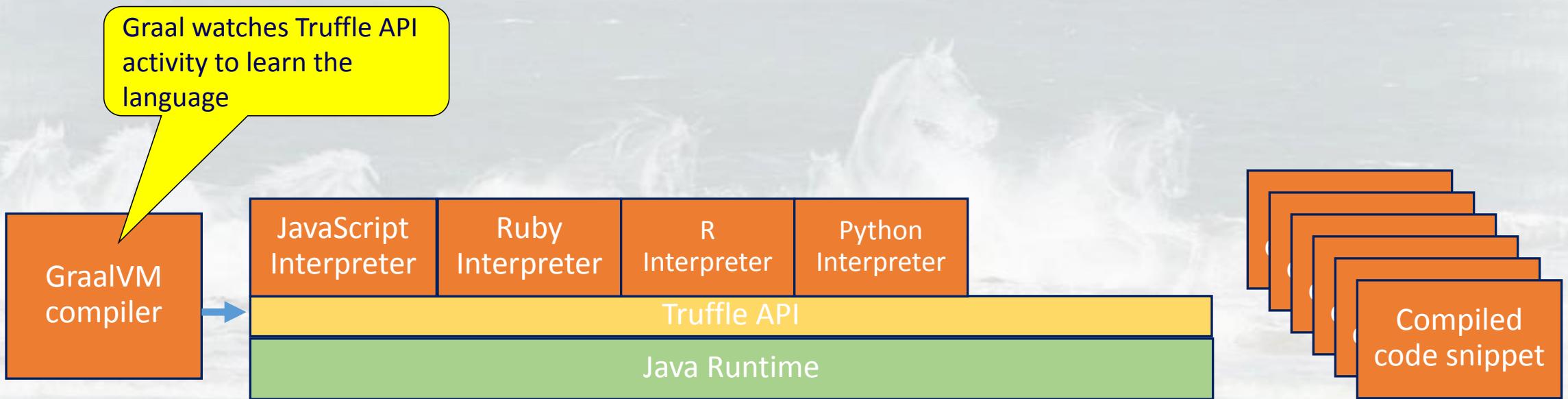
Structura internă



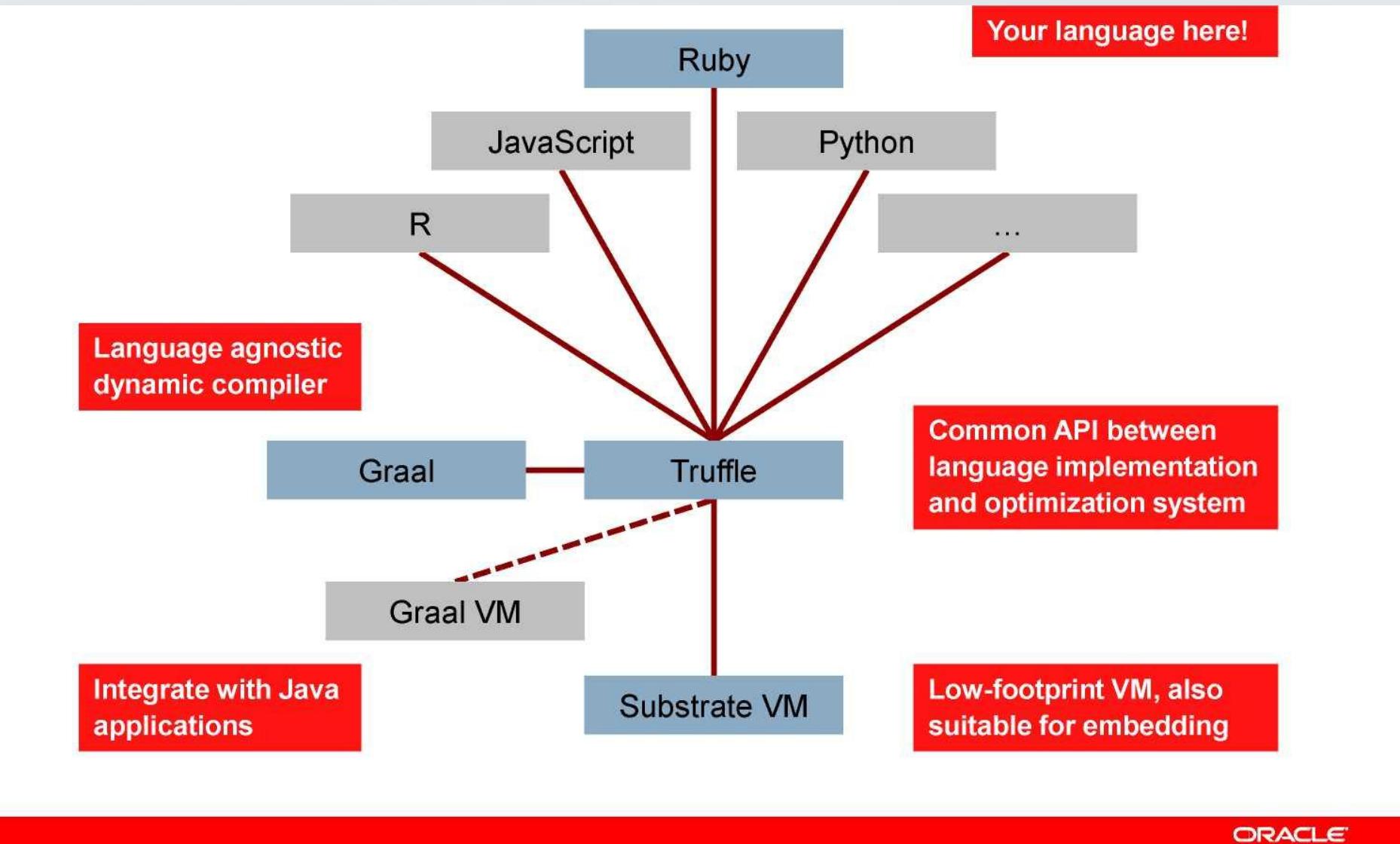
HotSpot



GraalVM este o virtualizare a Polyglot (Multilingual)

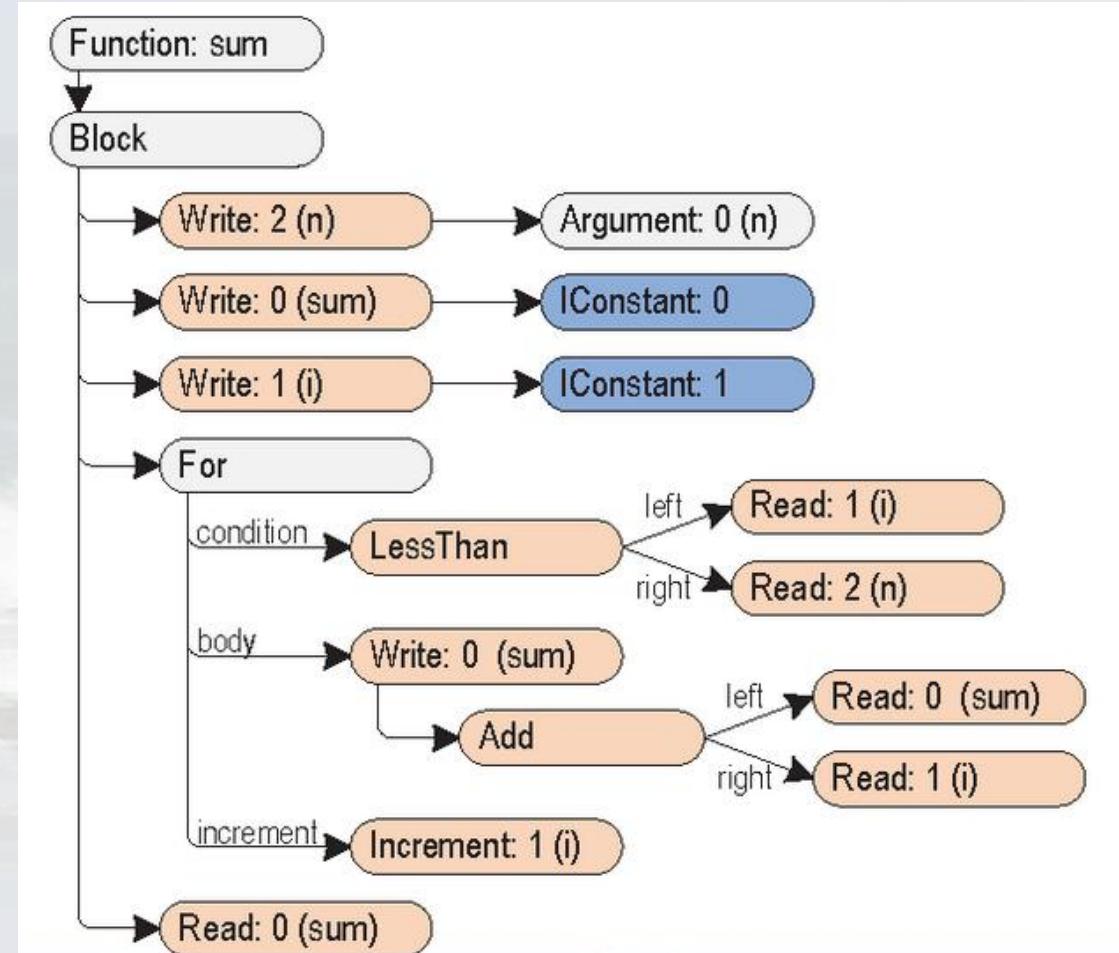


Limbaje suportate

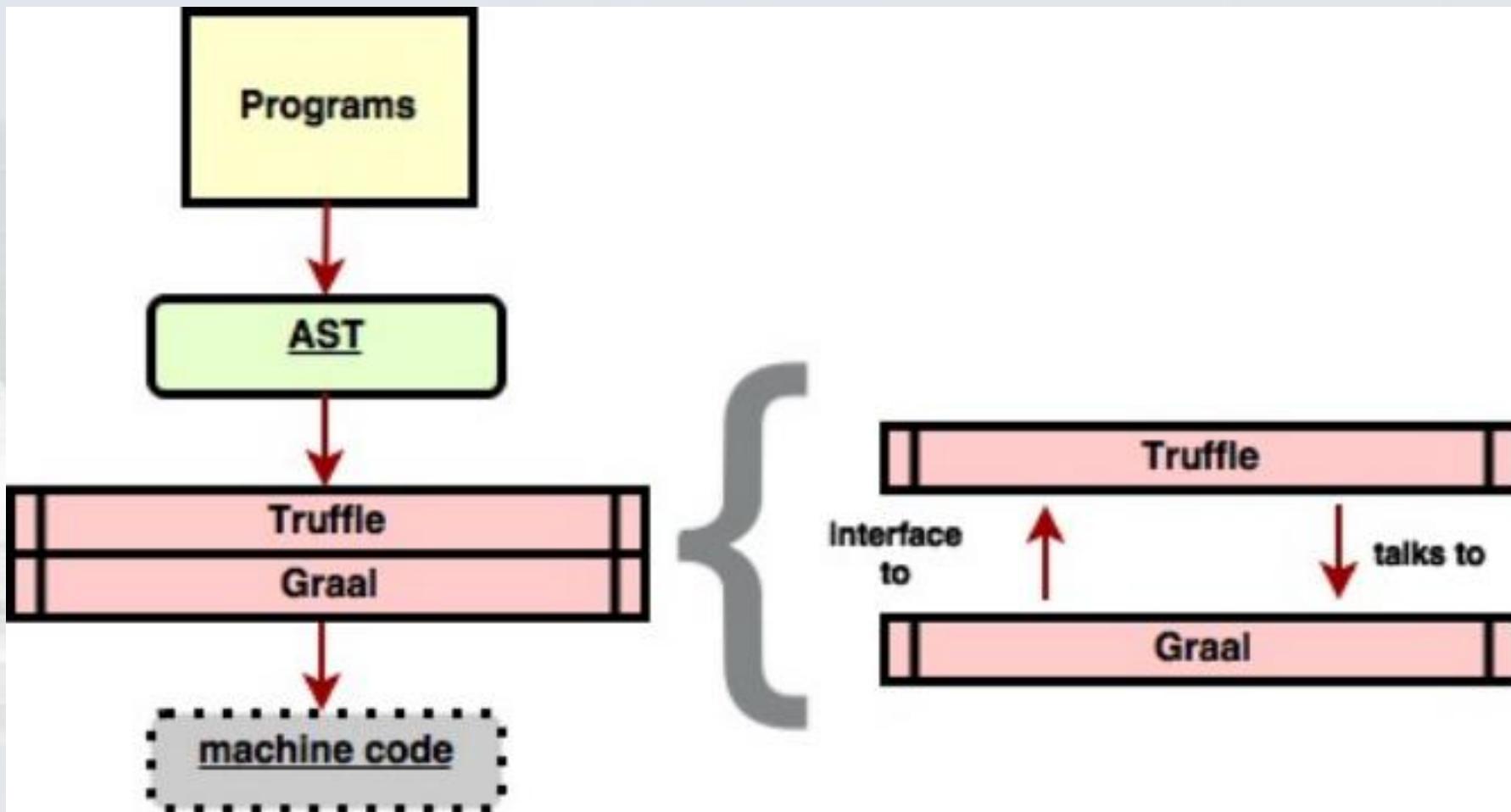


Ce este un AST-Abstract Syntax Tree

```
function sum(n) {  
    var sum = 0;  
    for (var i = 1; i < n; i++) {  
        sum += i;  
    }  
    return sum;  
}
```



Maniera de tratare a unui program



Creșteri de performanță obținute prin utilizare Graal

Datele de mai jos sunt date de ORACLE pe baza unor analize complexe care au implicat utilizarea unui număr mare de benchmark-uri (programe test standard)

	Range			Speedup (Geomean)	Comparison Versus
Java	0.8x	-	2x	1.1x	JDK8
Scala	0.8x	-	2x	1.3x	JDK8
JavaScript	0.5x	-	1.5x	1.05x	Google V8
Ruby	1x	-	100x	5x	JRuby
R	1x	-	100x	5x	GNU R
C/C++	0.4x	-	1.2x	0.9x	LLVM native

compilatoare și translatoare

- calcul funcțional
- church &turing
- Untyped λ -calculus

Expresii Lambda

Prin utilizarea lor funcțiile pot fi create fără a le asigna un nume explicit:

$$\lambda x \rightarrow x+x$$

În matematică este folosit simbolul \mapsto , deci $x \mapsto x+x$.

La ce e bun calculul Lambda ?

Expresiile Lambda pot fi folosite pentru a da un înțeles formal funcțiilor Curry.

De exemplu:

$$\text{add } x \ y = x+y$$

înseamna

$$\text{add} = \lambda x \rightarrow (\lambda y \rightarrow x+y)$$

Funcții Curry

Funcțiile cu argumente multiple sunt permise prin întoarcerea funcțiilor ca rezultat

```
add'      :: Int → (Int → Int)  
add' x y = x+y
```

add și add' produc același rezultat final dar add primește două argumente simultan în timp ce add' ia câte unul odată

```
add     :: (Int,Int) → Int  
add'    :: Int → (Int → Int)
```

Funcțiile cu mai mult de două argumente pot fi transformate în funcții Curry prin apel recursiv de funcții la parametri

```
mult      :: Int → (Int → (Int → Int))  
mult x y z = x*y*z
```

Care este avantajul funcțiilor Curry?

Sunt mult mai flexibile decât funcțiile bazate pe tuple în special datorită faptului că pot fi aplicate lor parțial:

```
add' 1 :: Int → Int  
take 5 :: [Int] → [Int]  
drop 5 :: [Int] → [Int]
```

Convenții specifice funcțiilor Curry

Pentru a evita folosirea în exces a parantezelor atunci când se folosesc funcții Curry s-au adoptat două convenții simple:

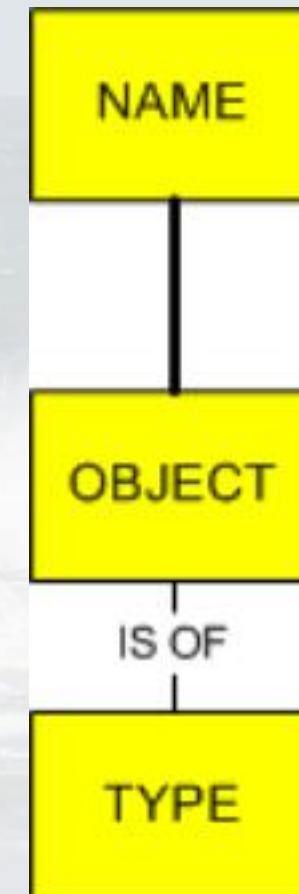
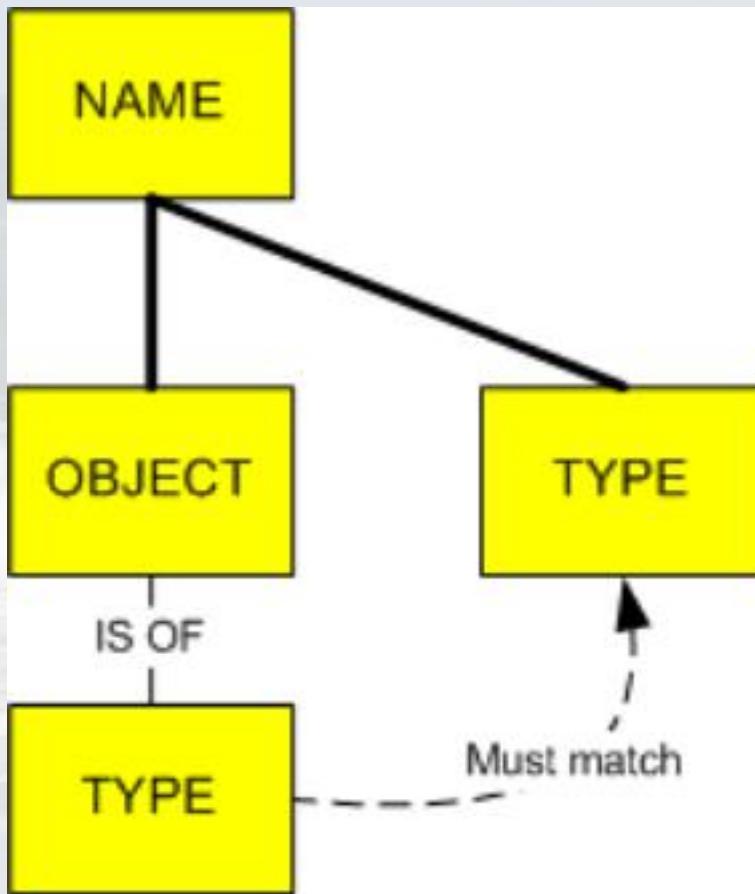
- Utilizarea săgeții → care face asociere la dreapta

```
Int → Int → Int →  
Int
```

- Ca o consecință apare o a doua convenție: funcțiile vor folosi asocierea la stânga

```
mult x y z
```

Stabilirea dinamică vs statică a tipurilor într-un limbaj de programare



mere vs pere

De exemplu într-un limbaj cu tipuri statice următoarea secvență de cod este ilegală

employeeName = 9 (s-a făcut asociere de tip int și gata!)

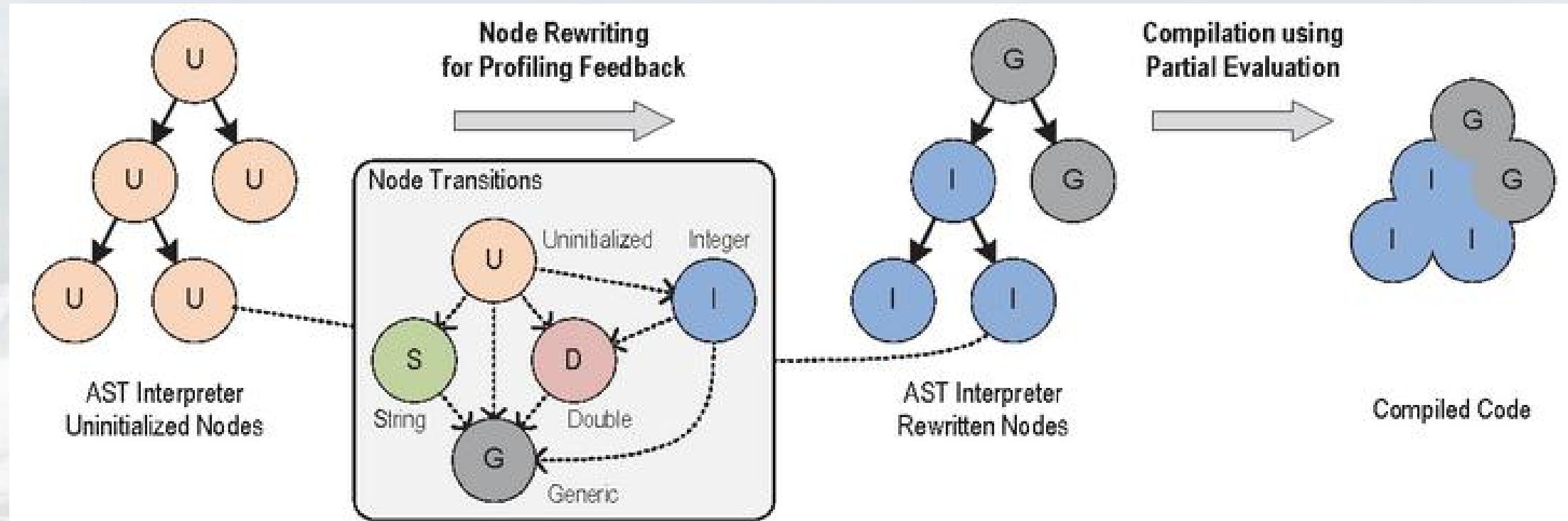
employeeName = "Steve Ferg"

exemplu

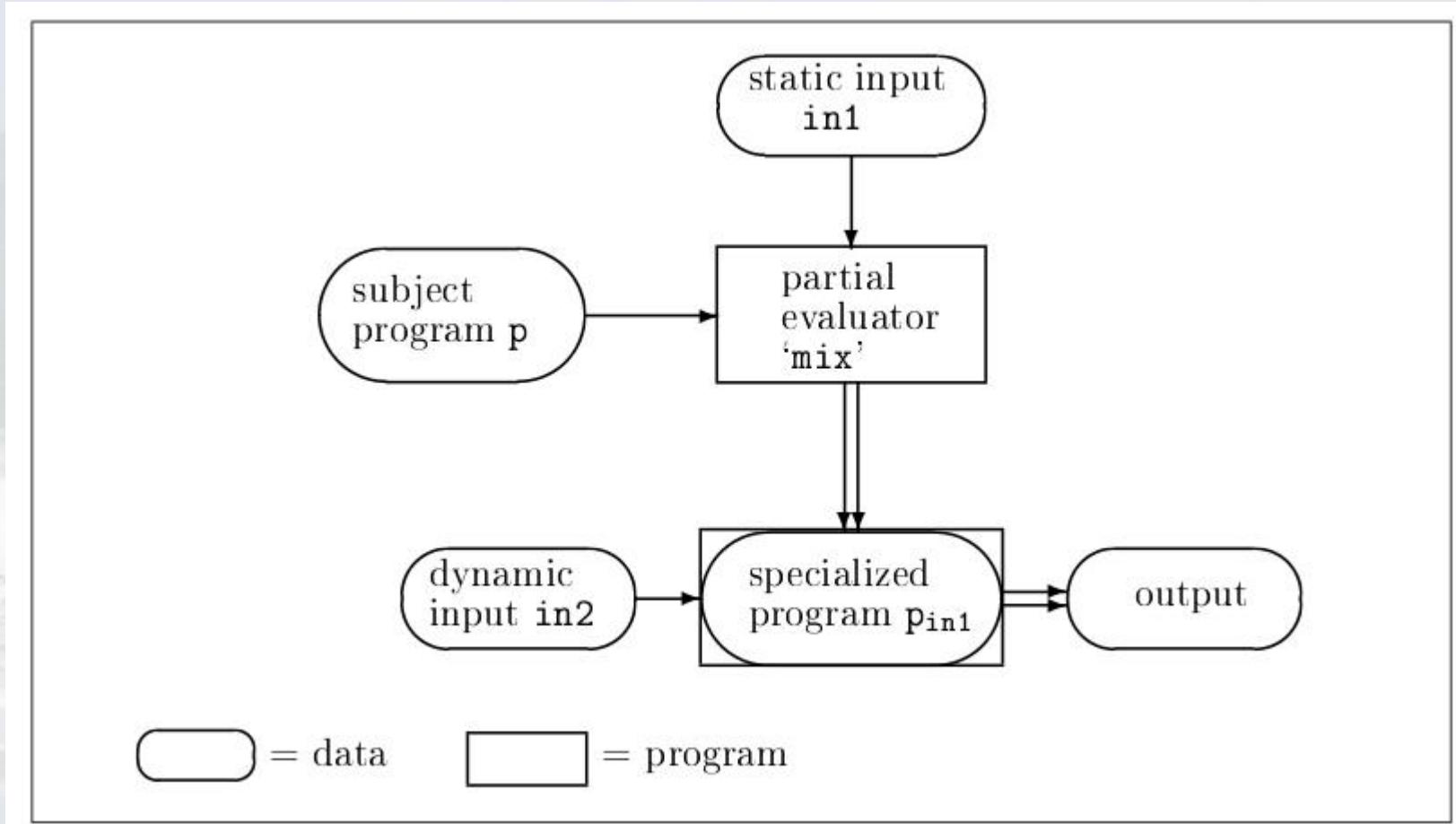
```
//Limbaj cu tipuri slabe  
a = 9  
b = "9"  
c = concatenate(a, b) // rezultat "99"  
d = add(a, b)        // rezultat 18
```

```
//limbaj cu tipuri tari  
a = 9  
b = "9"  
c = concatenate( str(a), b)//Grrr.....  
d = add(a, int(b)) //Mrrr.....
```

Reorganizarea nodurilor în arborele de evaluare



evaluator partial



Specializarea unui program de calcul x^n

A two-
input
program

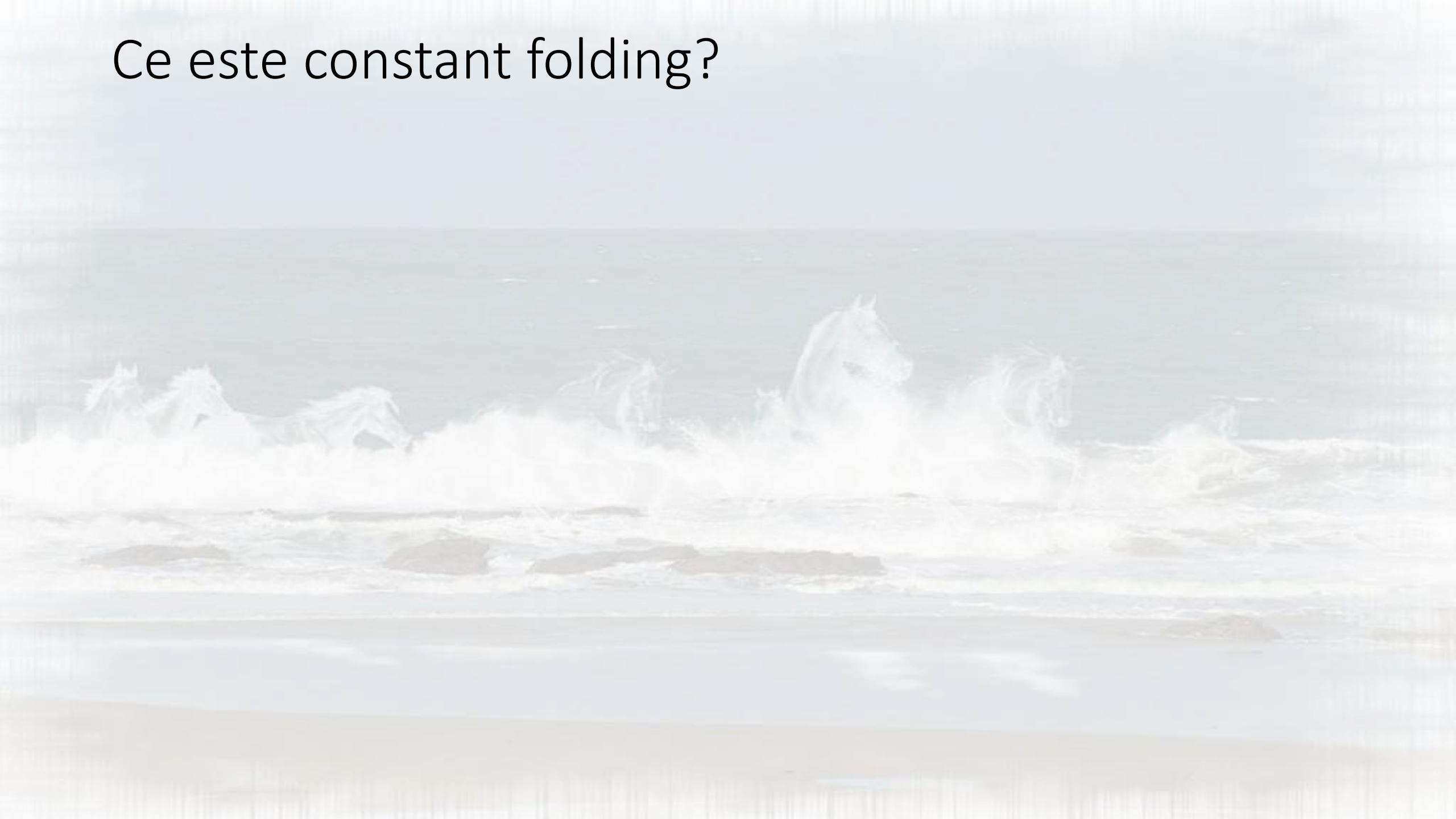
$p =$

```
f(n,x) = if n = 0 then 1
          else if even(n) then f(n/2,x)↑2
          else x * f(n-1,x)
```

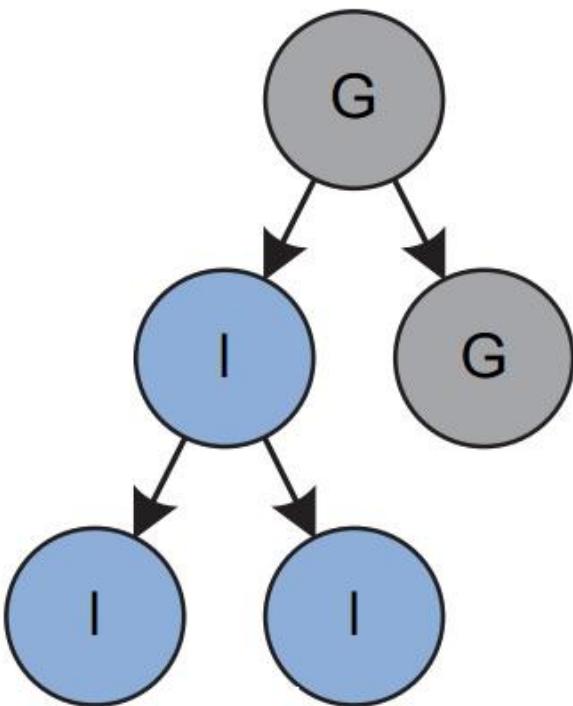
Program p , specialized to static input $n = 5$:

$p_5 = f_5(x) = x * ((x↑2)↑2)$

Ce este constant folding?

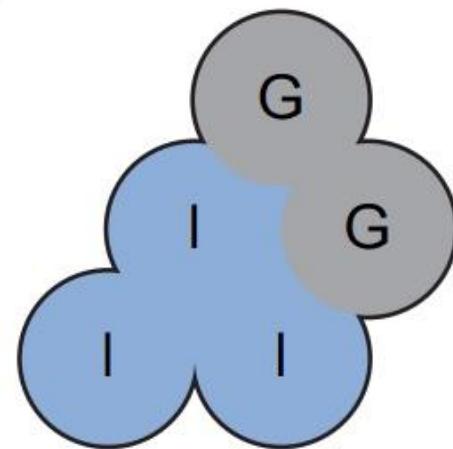


still..... folding



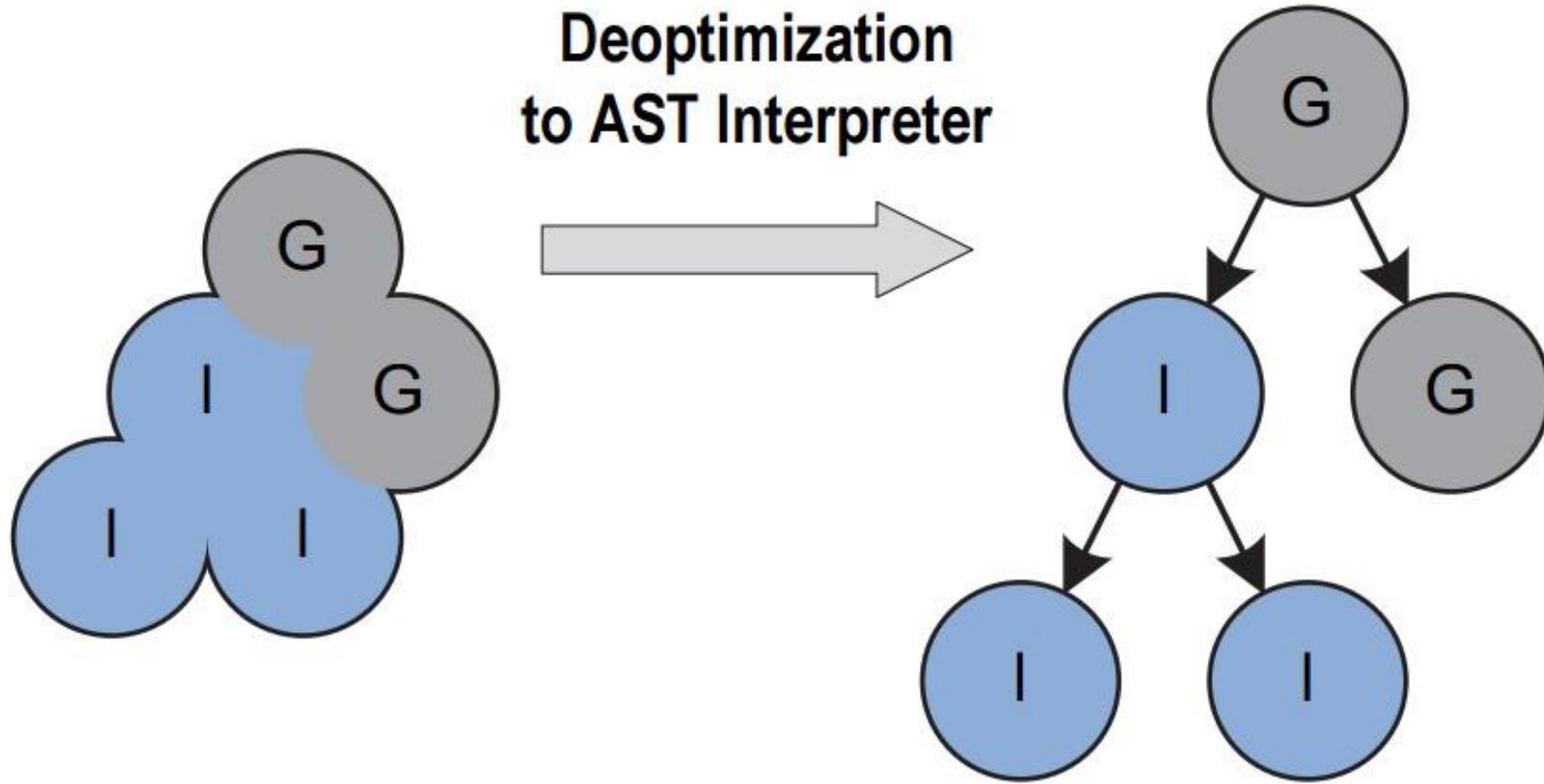
AST Interpreter
Rewritten Nodes

Compilation using
Partial Evaluation



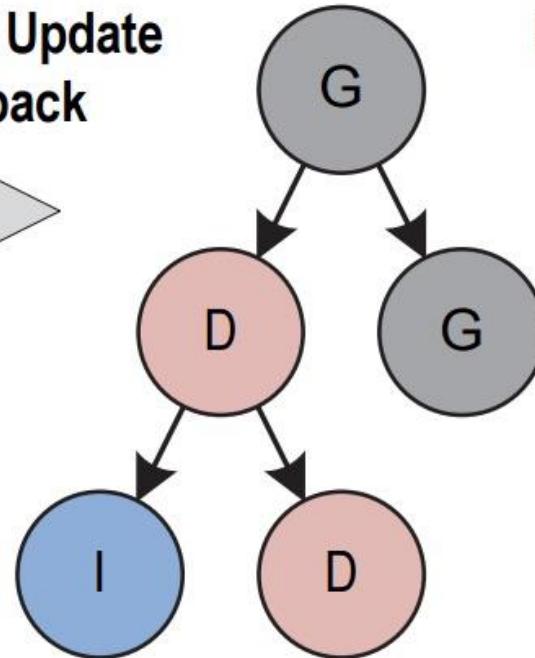
Compiled Code

eliminarea optimizarilor - de-optimizarea (De-optimisation)

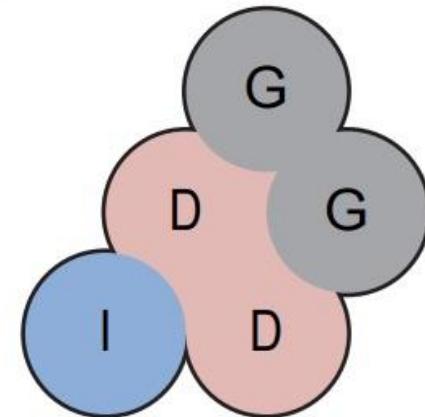


deoptimizarea

**Node Rewriting to Update
Profiling Feedback**



**Recompilation using
Partial Evaluation**



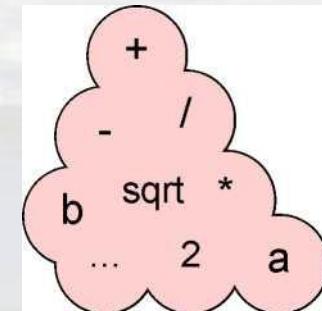
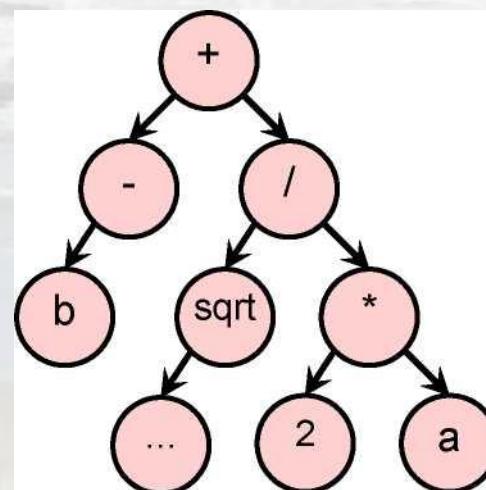
Optimizare performanțe cu Truffle

$-b + (\text{Math.sqrt}(b^{**}2 - 4*a*c)) / 2*a$

- execute b
- check that b is a Float
- check that the negate method in Float has not changed
- calculate negation
- check the result of that is a Float
- execute b
- check that b is a Float
- check that the power method in Float has not changed
- calculate power
- check the result of that is a Float
- execute a
- check that a is a Float
- check that the multiply method in Float has not changed
- calculate multiplication
- check the result of that is a Float
- execute c
- check that c is a Float
- check that the multiply method in Float has not changed
- calculate multiplication
- check the result of that is a Float
- check that Math has not changed
- check that the sqrt method in Math has not changed
- calculate sqrt
- check the result of that is a Float
- execute a
- check that a is a Float
- check that the multiply method in Float has not changed
- calculate multiplication
- check the result of that is a Float
- check that the division method in Float has not changed
- calculate division



- execute b
- check that the negate method in Float has not changed
- calculate negation execute b
- check that the power method in Float has not changed
- calculate power execute a
- check that the multiply method in Float has not changed
- calculate multiplication execute c
- check that the multiply method in Float has not changed
- calculate multiplication
- check that Math has not changed
- check that the sqrt method in Math has not changed
- calculate sqrt execute a
- check that the multiply method in Float has not changed
- calculate multiplication
- check that the division method in Float has not changed
- calculate division



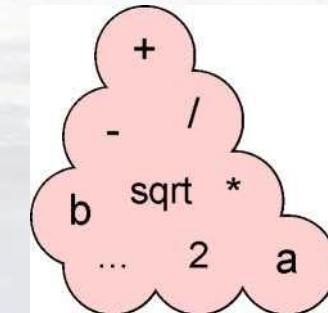
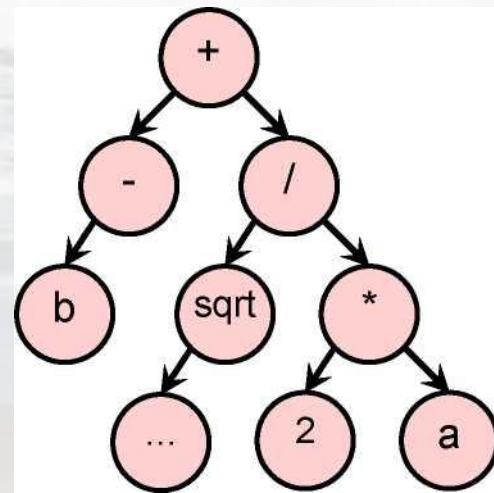
Optimizare performanțe cu Graal

$$-b + (\text{Math.sqrt}(b^{**}2 - 4*a*c)) / 2*a$$

- execute b
- check that the negate method in Float has not changed
- calculate negation execute b
- check that the power method in Float has not changed
- calculate power execute a
- check that the multiply method in Float has not changed
- calculate multiplication execute c
- check that the multiply method in Float has not changed
- calculate multiplication
- check that Math has not changed
- check that the sqrt method in Math has not changed
- calculate sqrt execute a
- check that the multiply method in Float has not changed
- calculate multiplication
- check that the division method in Float has not changed
- calculate division



execute b calculate negation execute b calculate power
execute a
calculate multiplication execute c
calculate multiplication calculate sqrt execute a
calculate multiplication calculate division



Programare în PolyGlot

- import org.graalvm.polyglot.*;
- public class HelloPolyglotWorld {
 - public static void main(String[] args) throws Exception {
 - System.out.println("Hello polyglot world Java!");
 - Context context = Context.create();
 - context.eval("js", "print('Hello polyglot world JavaScript!');");
 - context.eval("ruby", "puts 'Hello polyglot world Ruby!'");
 - context.eval("R", "print('Hello polyglot world R!');");
 - context.eval("python", "print('Hello polyglot world Python!');");
 - }
 - }

Script-uri în paralel

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;
import org.graalvm.polyglot.Context;
public class ParallelEval
{
    public static void main(String[] args) throws InterruptedException, ExecutionException
    {
        ExecutorService service = Executors.newFixedThreadPool(1);
        Context context = Context.create("js");
        Future<Integer> future = service.submit(() -> context.eval("js", "21 + 21").asInt());
        // executa altceva în timp ce scriptul Java se executa
        int result = future.get();
        assert 42 == result;
        context.close();
    }
}
```

detectia unei duree prea mare de executie

```
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

import org.graalvm.polyglot.Context;
import org.graalvm.polyglot.PolyglotException;
import org.graalvm.polyglot.Value;

public class CancelExecution {

    public static void main(String[] args) throws
InterruptedException {
        ExecutorService
serviceExecutors.newFixedThreadPool(1);
        try {
            Context context = Context.create("js");
            // se trimite pt testare script-uri cu buclile infinite
Future<Value> future = service.submit(() -> context.eval("js",
"while(true);"));
            Thread.sleep(1000);
            // se opreste executarea si se inchide contextul
            // se poate face in orice thread in paralel
            // se mai poate utiliza context.close(true) pentru a inchide toate
            context.close(true);
            try { future.get();
            } catch (ExecutionException e)
            {
                PolyglotException polyglotException =
(PolyglotException) e.getCause();
                polyglotException.printStackTrace();
                // dupa oprirea executiei thread va fi generata o exceptie
                // PolyglotException cu flagul cancelled activat.
                assert polyglotException.isCancelled();
            }
            } finally {
                service.shutdown();
            }
        }
    }
```

exemplu Sulong native pipe

```
#include <jni.h>
#include "com_oracle_truffle_llvm_pipe_CaptureNativeOutput.h"

#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>
#include <errno.h>

static bool check_error(JNIEnv *env, int ret) {
    if (ret < 0) {
        char *message = strerror(errno);
        jclass ioex = env->FindClass("java/io/IOException");
        env->ThrowNew(ioex, message);
        return true;
    } else {
        return false;
    }
}

JNIEXPORT jint JNICALL
Java_com_oracle_truffle_llvm_pipe_CaptureNativeOutput_startCapturing(JNIEnv *
*env, jclass self, jint stdFd, jstring filename) {
    const char *path = env->GetStringUTFChars(filename, NULL);
    int fd = open(path, O_WRONLY);
    bool error = check_error(env, fd);
    env->ReleaseStringUTFChars(filename, path);
    if (error) {
        return -1;
    }

    int oldFd = dup(stdFd);
    if (check_error(env, oldFd)) {
        close(fd);
        return -1;
    }

    int result = dup2(fd, stdFd);
    if (check_error(env, result)) {
        close(fd);
        close(oldFd);
        return -1;
    }
    close(fd);
    return oldFd;
}

JNIEXPORT void JNICALL
Java_com_oracle_truffle_llvm_pipe_CaptureNativeOutput_stopCapturing(JNIEnv *
*env, jclass self, jint oldStdOut, jint oldStdErr) {
    if (check_error(env, fflush(stdout))) {
        return;
    }
    if (check_error(env, fflush(stderr))) {
        return;
    }
    if (check_error(env, dup2(oldStdOut,
        com_oracle_truffle_llvm_pipe_CaptureNativeOutput_STDOUT))) {
        return;
    }
    if (check_error(env, dup2(oldStdErr,
        com_oracle_truffle_llvm_pipe_CaptureNativeOutput_STDERR))) {
        return;
    }
    if (check_error(env, close(oldStdOut))) {
        return;
    }
    check_error(env, close(oldStdErr));
}
```

Logging

Ca să facilitate suplimentar [

```
DebugContext debug = ...;
```

```
InstalledCode code = null;
```

```
try (Scope s = debug.scope("CodeInstall", method))
```

```
{
```

```
    code = ...
```

```
    debug.log("installed code for %s", method);
```

```
} catch (Throwable e) {
```

```
    throw debug.handle(e);
```

```
}
```

Metrici

- Fiecare metrică are un nume unic.
- Metricile sunt colectate pentru fiecare compilare
- Pentru a le lista -Dgraal.MetricsFile option.

metrii

```
CounterKey (cantitativ)
// declaratii
private static final CounterKey ByteCodesCompiled =
DebugContext.counter("ByteCodesCompiled");
// utilizare
DebugContext debug = ...;
long compiled = ... ;
ByteCodesCompiled.add(debug, compiled);
```

TimerKey (timp scurs)

```
// declaratie
private static final TimerKey CompilationTime = DebugContext.timer("CompilationTime");
// usage
DebugContext debug = ...;
try (DebugCloseable scope = CompilationTime.start(debug)) {
    ...
}
// timp scurs pentru respectivul scop se va adăuga la `CompilationTime` din `debug`
```

metrici

CompilationTime_Accm=100.0 ms

CompilationTime_Flat=10.0 ms

O cheie MemUseTrackerKey urmărește rezervarea de memorie pentru un scop specific de ex

- // declaratie
- private static final MemUseTrackerKey CompilationMemory =
DebugContext.memUseTracker("CompilationMemory");
- // utilizare
- DebugContext debug = ...;
- try (DebugCloseable a = CompilationMemory.start(debug)) {
- ...
- }
- // octetii folosiți aici vor fi sumati la valoarea debug `CompilationMemory`

metri

ca și TimerKey, o valoare acumulată și flat este raportată de tracker-ul pentru memorie

CompilationMemory_Accm=100000 bytes

CompilationMemory_Flat=1000 bytes

Pentru ambele opțiuni -Dgraal.AggregatedMetricsFile și -Dgraal.MetricsFile ieșirea este determinată de sufixul numelui

.csv produce format cu separare punct și virgulă

Coloanele rezultate din -Dgraal.MetricsFile CSV sunt:

metrici

- **compilable:**
- **compilable_identity:**
- **compilation_nr:**
- **compilation_id:**
- **metric_name:**
- **metric_value:**
- **metric_unit:**

Exemplu metrici (fișier ieșire) -Dgraal.MetricsFile

- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseNodes_PhaseSuite;1
- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseNodes_GraphBuilderPhase;1
- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseCount_GraphBuilderPhase;1
- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseMemUse_GraphBuilderPhase_Accm;896536
- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseMemUse_GraphBuilderPhase_Flat;896536
- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseTime_GraphBuilderPhase_Accm;31095000
- java.lang.String.hashCode()int;1272077530;0;HotSpotCompilation-95;PhaseTime_GraphBuilderPhase_Flat;27724000
- *coloanele din fișierul ieșire CSV al -Dgraal.AggregatedMetricsFile sunt:*
 - metric_name:
 - metric_value:
 - metric_unit:

Dumping

- **HIR graphs** (de ex instance ale Graph) către Ideal Graph Visualizer (IGV), și
- **LIR register allocation si generated code** către C1Visualizer
- opțiunea -Dgraal.Dump

Urmărirea mașinii virtuale pentru o compilare

- cu -XX:+PrintCompilation sau -Dgraal.PrintCompilation=true pentru notificări și informații sumare -XX:+TraceDeoptimization poate fi utilizată pentru a vedea dacă durează prea mult compilarea
 - De exemplu inspectarea compilării pentru o metodă dacă vreau să văd structurile de date utilizate când se compilează Node.updateUsages, se folosește următoarea comandă
 - > mx vm -XX:+UseJVMCICompiler -XX:+BootstrapJVMCI -XX:-TieredCompilation -Dgraal.Dump= -Dgraal.MethodFilter=Node.updateUsages -version
 - Bootstrapping JVMCI....Dumping debug output in /home/bugs/graal/graal/compiler/dumps/1497910458736
 - in 38177 ms (compiled 5206 methods)
 - java version "1.8.0_121"
 - Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
 - Java HotSpot(TM) 64-Bit Server VM (build 25.71-b01-internal-jvmci-0.26, mixed mode)
- > find dumps/1497910458736 -type f
 - dumps/1497910458736/HotSpotCompilation-539[org.graalvm.compiler.graph.Node.updateUsages(Node, Node)].bgv
 - dumps/1497910458736/HotSpotCompilation-539[org.graalvm.compiler.graph.Node.updateUsages(Node, Node)].cfg

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.

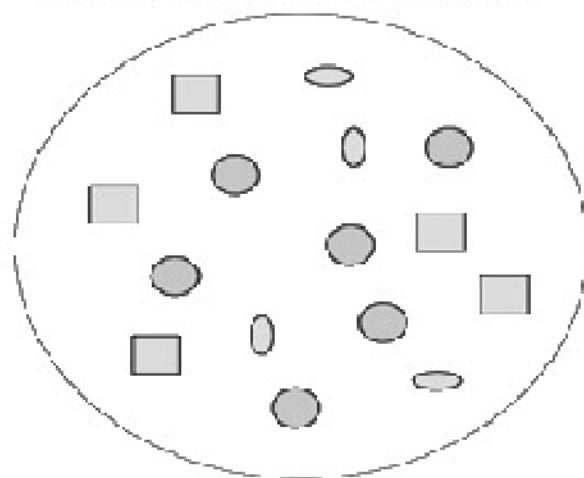
De exemplu un student, o mașină, o dată și o oră

Problemele sunt rezolvate prin:

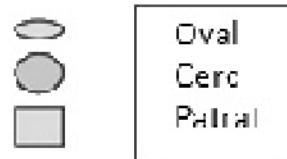
1. Izolarea obiectelor implicate
2. Determinarea proprietăților și acțiunilor (sau responsabilităților) acestora și
3. Descrierea colaborării între obiecte cu scopul rezolvării problemei

Fazele pentru rezolvarea și implementarea problemei în OO

Problema Spatiului de obiecte



De la Abstracțare spre Clase
(descrierea obiectelor)



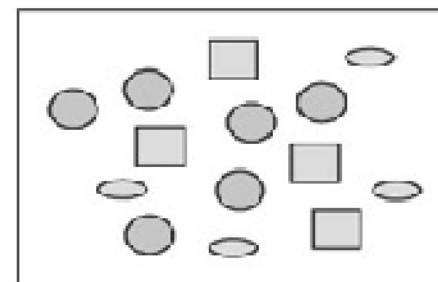
1. Faza de rezolvare a problemei

Definirea tipurilor claselor



2. Faza de implementare

Spatiul program al obiectelor



Variabile var&val

- declarare & initializare variabila RO

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

- declarare & initializare variabila normală

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

Inferența de tip

Desi 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 și 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 implicate?

```
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()

Variabile 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, ', ", \\", \\$.

Şiruri de caractere

- val string = "string with \n new line"
- mai există ceva numit şir 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 ca 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()));
```

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

Clase 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) }
    }
}
```

Clase anónime

```
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()
```

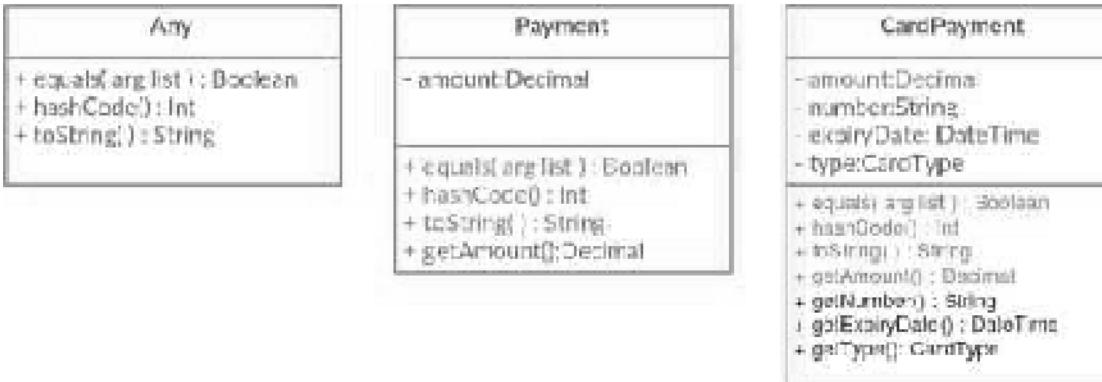
Metode statice și obiecte companion

```
fun showFirstCharacter(input:String):Char //ex1
{
    if(input.isEmpty()) throw IllegalArgumentException()
    return input.first()
}
• iar apelul în cod
showFirstCharacter("Kotlin is cool!")
object Singleton //ex2
{
    private var count = 0
    fun doSomething():Unit {println("Calling a doSomething (${++count} call/-s in total")"}
}
• iar apelul în cod
Singleton.doSomething
```

Interfețe

```
interface Document //ex1
{ val version: Long
  val size: Long
  val name: String get() = "NoName"
  fun save(inputStream)
  fun load(stream: OutputStream)
  fun getDescription(): String
    {return "Document $name has $size byte(-s)"} }
public class MyDocument implements Document //ex2 -Java
{ public long getVersion() { return 0; }
  public long getSize() { return 0; }
  public void save(@NotNull InputStream input) {}
  public void load(@NotNull OutputStream stream) {}
  public String getName() { return null; }
  public String getDescription() { return null; } }
class DocumentImpl : Document //ex3 - Kotlin
{ override val size: Long get() = 0
  override fun load(stream: OutputStream) {}
  override fun save(inputStream: InputStream) {}
  override val version: Long get() = 0 }
```

Moștenirea simplă stop curs 3



enum class CardType

{VISA, MASTERCARD, AMEX}

open class Payment(val amount: BigDecimal)

class CardPayment(amount: BigDecimal, val number: String,
val expiryDate: DateTime, val type: CardType) : Payment(amount)

Moștenire simplă

```
class ChequePayment : Payment
{
    constructor(amount: BigDecimal, name: String, bankId: String) :
        super(amount)
    {
        this.name = name
        this.bankId = bankId
    }
    var name: String
        get() = this.name
    var bankId: String
        get() = this.bankId
}
```

Moștenire Multiplă simulată

```
interface Drivable { fun drive()} //ex1
interface Sailable { fun saill()  }
class MultiRoleCar(val name: String) : Drivable, Sailable
{ override fun drive() { println("Driving...")}
  override fun saill() { println("Sailling...")} }

interface IPersistable { fun save(stream: InputStream) }//ex2
interface IPrintable { fun print() }
abstract class Document(val title: String)
class TextDocument(title: String) : IPersistable, Document(title),
IPrintable
{ override fun save(stream: InputStream)
  {println("Saving to input stream") }
  override fun print()
  { println("Document name:$title")}
}
```

Funcții în Kotlin

```
fun hello(): String = "hello world" // fără listă parametri
```

```
fun hello(name: String, location: String): String = "hello to you  
$name at $location" // cu listă de parametri
```

```
fun print1(str: String): Unit // ex2
```

```
    { println(str) }
```

```
fun print2(str: String)
```

```
    { println(str) }
```

Funcții cu expresie unică

```
fun square(k:Int) = k * k //ex1
```

```
fun square2(k:Int):Int = k * k //ex2
```

```
fun concat1(a: String, b: String) = a + b //ex3
```

```
fun concat2(a: String, b: String): String //descriere standard
```

```
{
```

```
    return a + b
```

```
}
```

Funcții membru

```
val string = "hello" //ex1
val length = string.take(5)
object Rectangle //ex2
{
    fun printArea(width: Int, height: Int): Unit
    {
        val area = calculateArea(width, height)
        println("The area is $area")
    }
    fun calculateArea(width: Int, height: Int): Int
    {
        return width * height
    }
}
```

Functii locale

```
fun printArea(width: Int, height: Int): Unit //ex1
{
    fun calculateArea(width: Int, height: Int): Int = width * height
    val area = calculateArea(width, height)
        println("The area is $area")
}
fun printArea2(width: Int, height: Int): Unit //ex2
{
    fun calculateArea(): Int = width * height
    val area = calculateArea()
        println("The area is $area")
}
```

Funcții top-level

```
fun foo(k: Int) //ex 1
{
    require(k > 10,  { "k should be greater than 10" } )
}
```

Funcții cu parametri denumiți explicit

```
val string = "a kindness of ravens" string.regionMatches(14, "Red
Ravens", 4, 6, true) //ex2
```

```
string.regionMatches(thisOffset = 14, other = "Red Ravens",
otherOffset = 4, length = 6, ignoreCase = true) //ex3
```

Paradigma Orientata Obiect

Cursul nr. 4
Mihai Zaharia

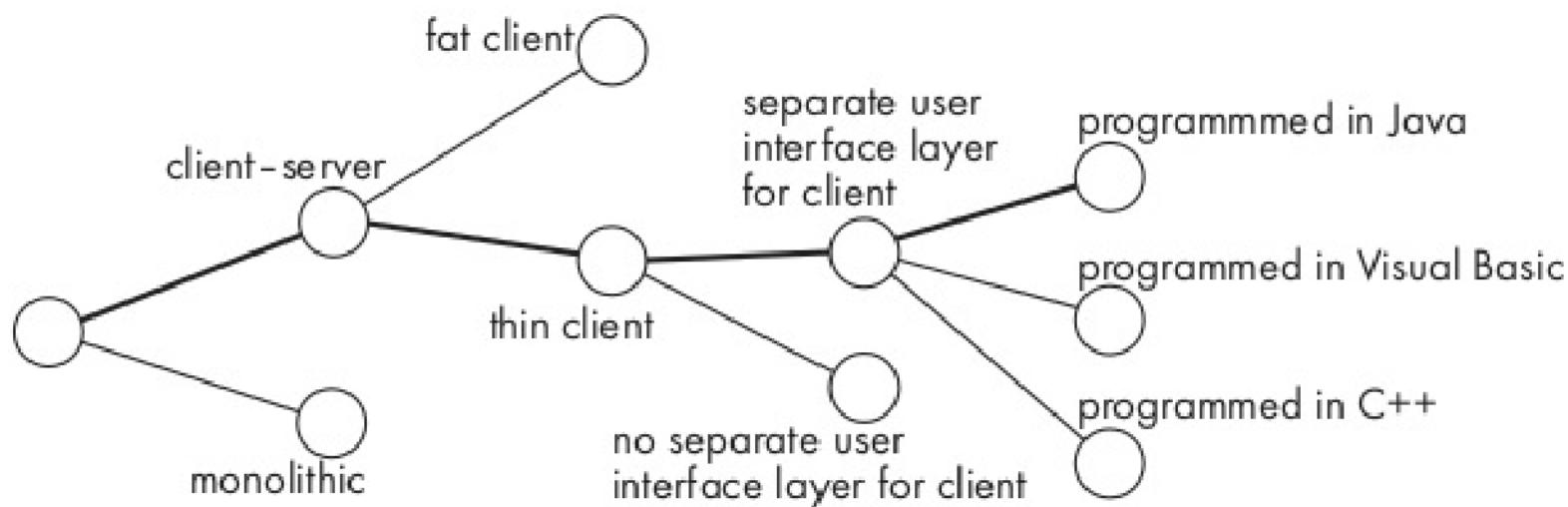
Proiectare? = o serie de decizii

Definiție: Proiectarea, în contextul aplicațiilor software reprezintă un proces de rezolvare a unor probleme, obiectivul fiind SĂ SE GĂSEASCĂ și SĂ SE DESCRIE o cale de a implementa NECESITĂȚILE FUNCȚIONALE ale sistemului tinând cont și de CONSTRÂNGERILE impuse de:

- nivelul de calitate așteptat,
- specificațiile platformei(lor) țintă,
- de necesitățile specifice ale procesului(lor) care trebuie modelat
- și aspectele de business cum ar fi:
 - termenul de realizare și
 - fondurile disponibile pentru respectivul proiect.

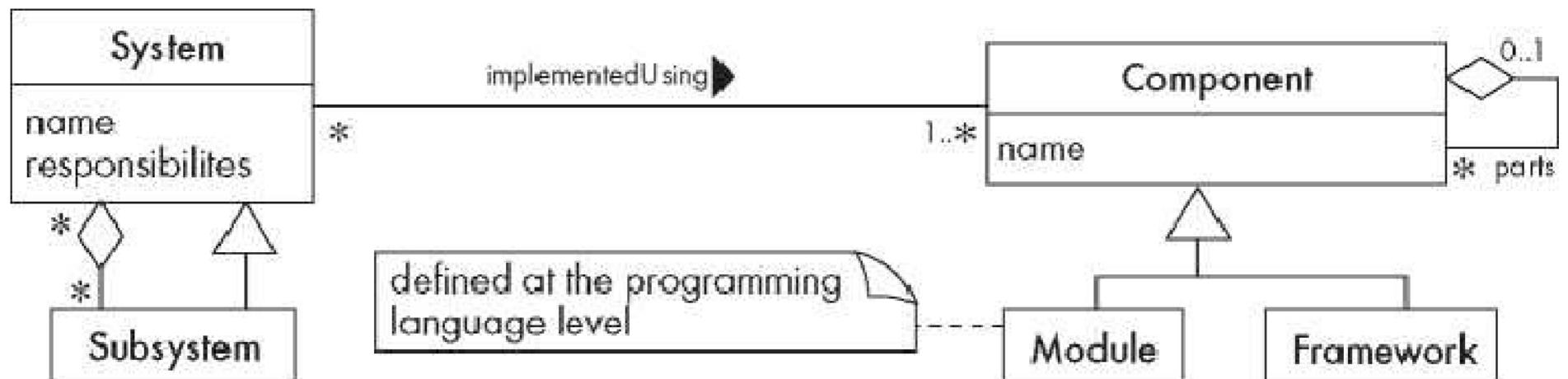
Sistemul

- Sistemul care va fi proiectat este alcătuit din:
 - subsisteme
 - componente și
 - module.



Arbore decizional general specific primei analize după stabilirea specificațiilor
și a constrângerilor

Modelul domeniului



Acesta explică concepțele de sistem, subsistem, componentă și modul

Ce este de fapt UML?

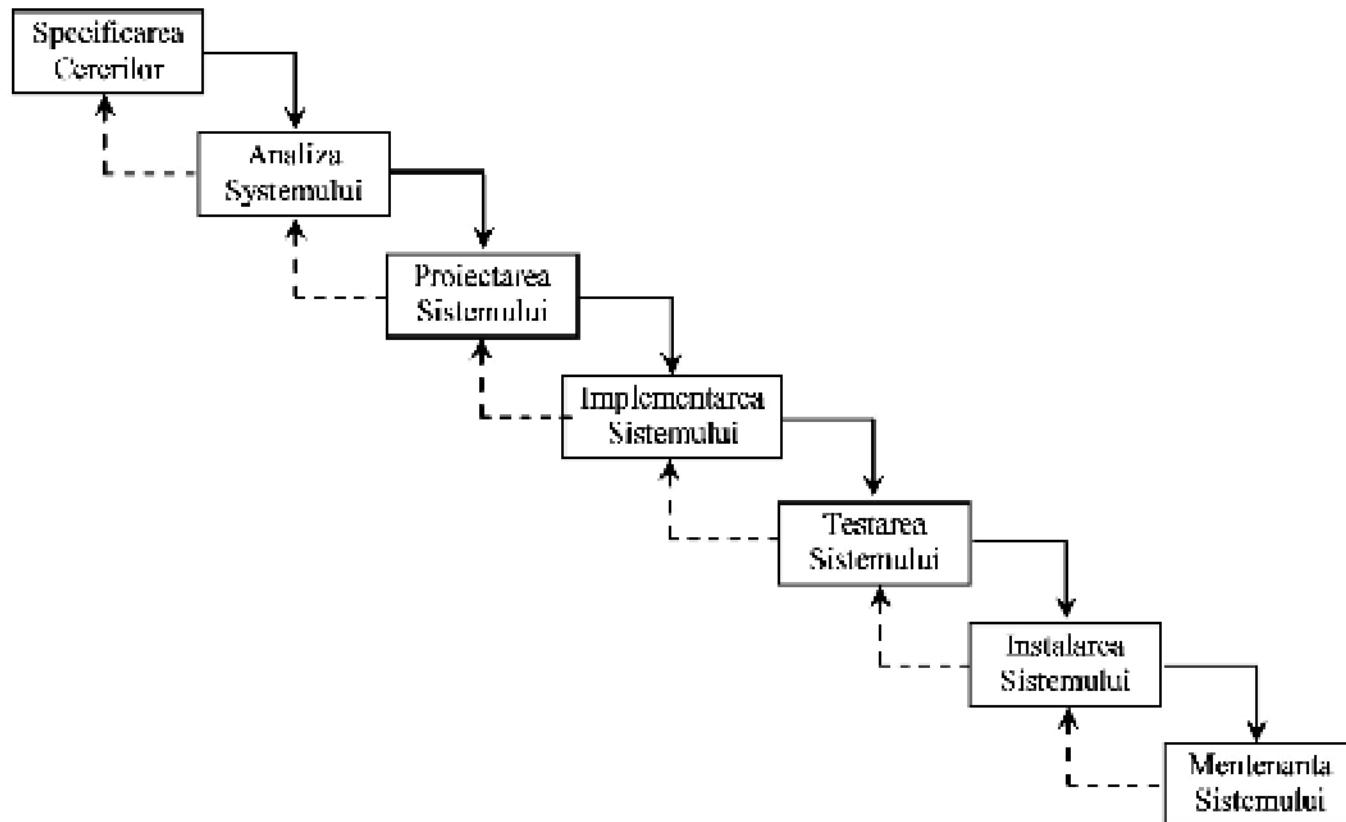
Definiție: (OMG)

- "*Unified Modeling Language (UML) reprezintă un limbaj grafic pentru vizualizarea, specificarea, dezvoltarea și documentarea componentelor unui sistem software de dimensiuni medii sau mari.*"
- "*UML oferă o manieră standard pentru a crea schema unui sistem pornind de la aspecte concrete cum ar fi blocuri de cod, scheme pentru bazele de date, componente reutilizabile și ajungând la aspecte abstracte precum capturarea fluxului de desfăsurare a unei afaceri sau funcții ale sistemului.*"

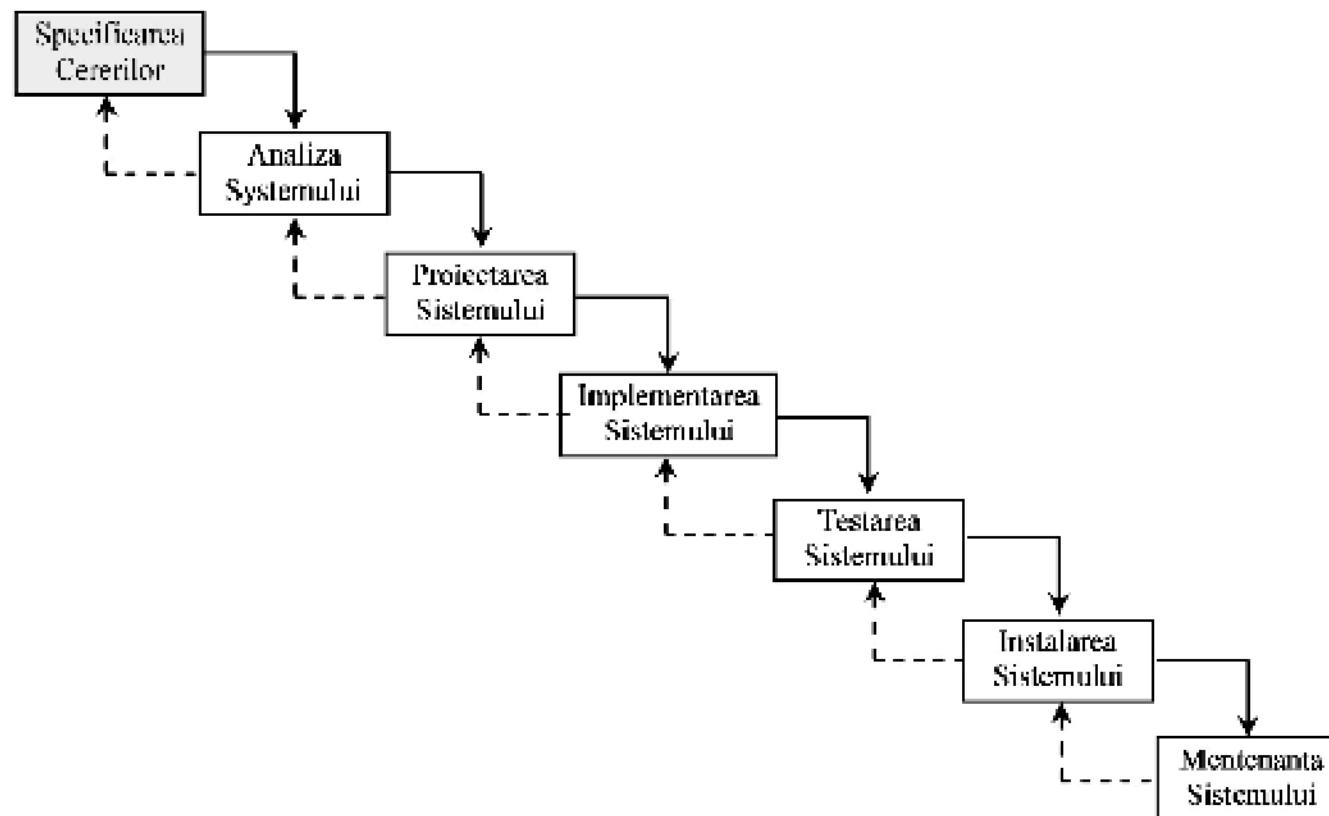
UML oferă semantici și notații pentru

1. User Interaction/Use Case Model
2. Interaction / Communication Model
3. State / Dynamic Model
4. Logical / Class Model
5. Physical Component Model
6. Physical Deployment Model
7. Mai definește și soluții pentru extinderea sau dezvoltarea după nevoi a unor mecanisme existente

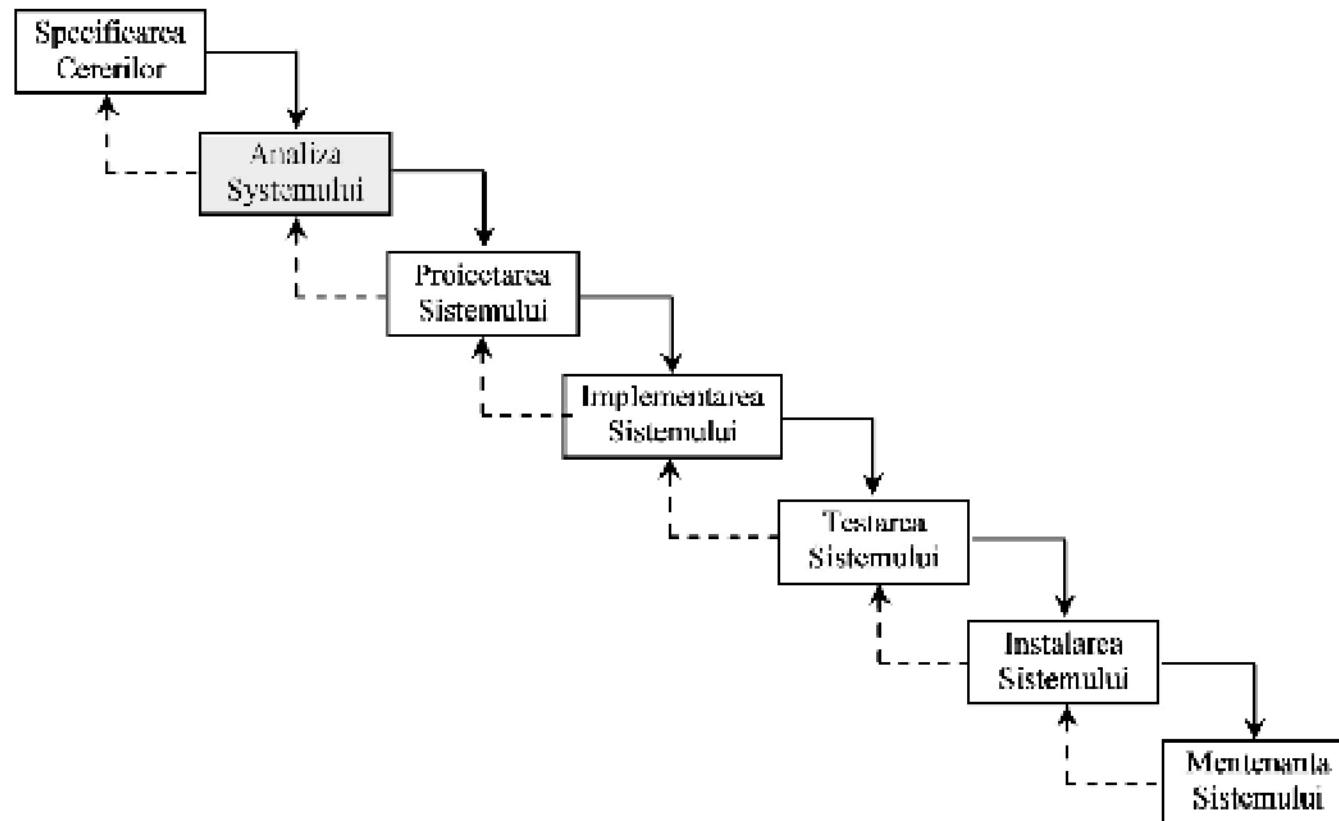
Procesul dezvoltare a unei aplicații



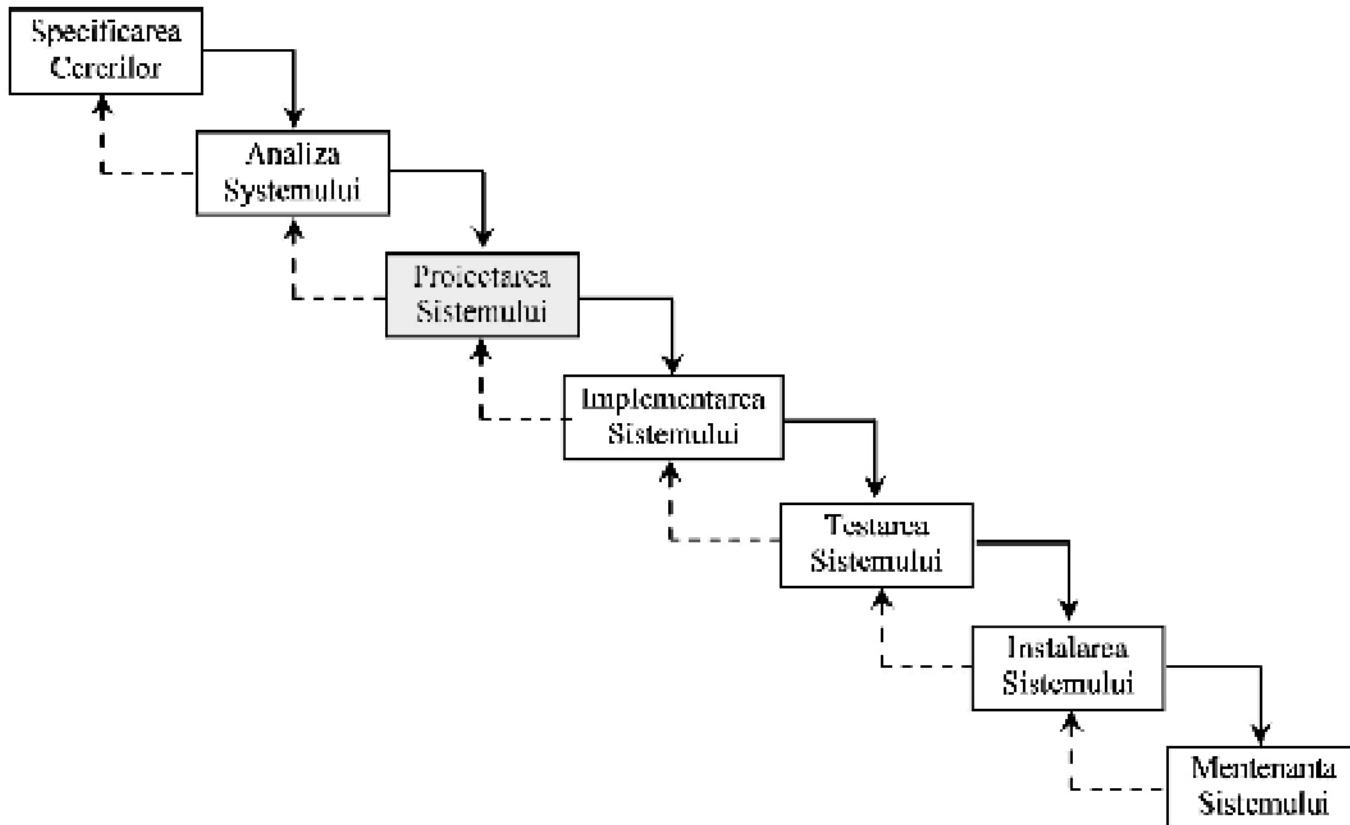
Specificarea Cererilor



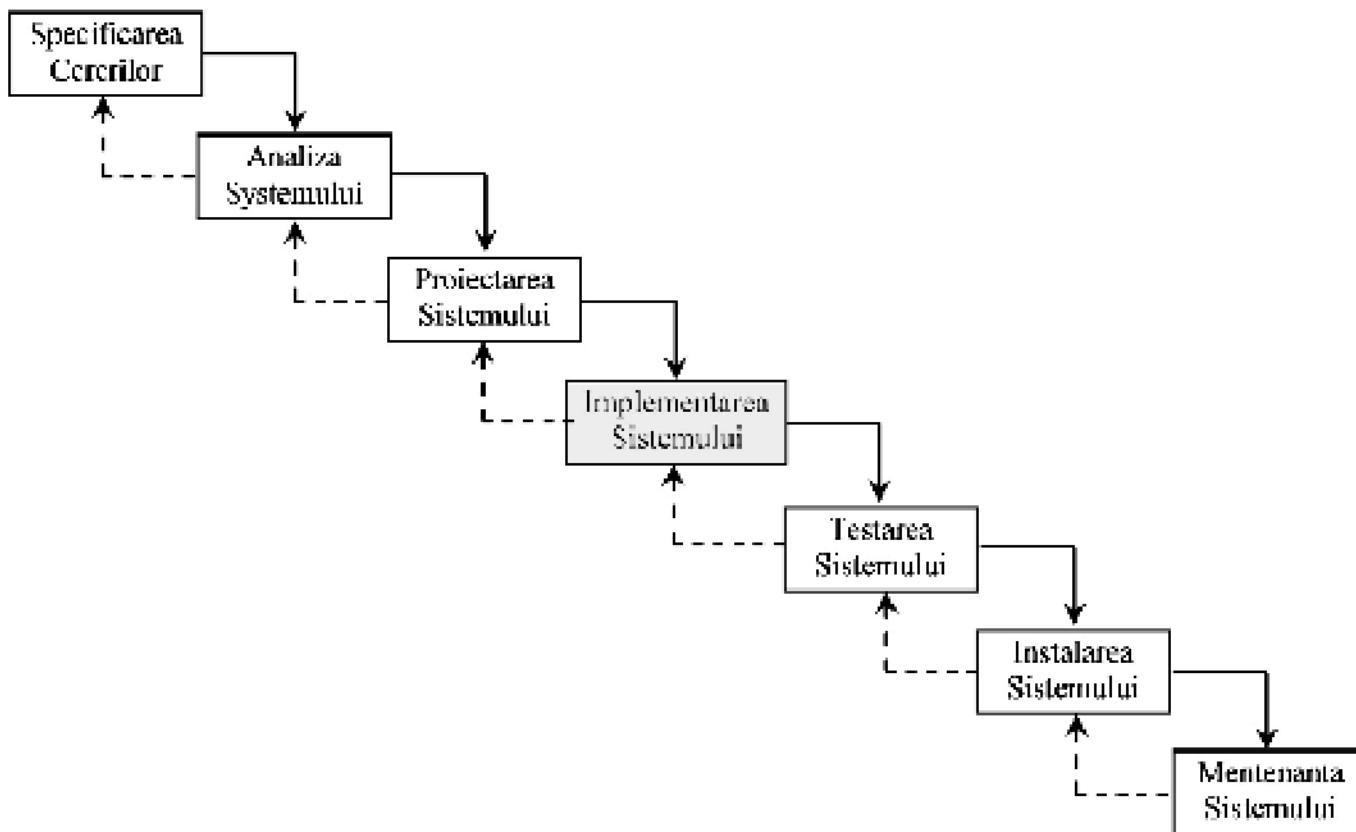
Analiza sistemului



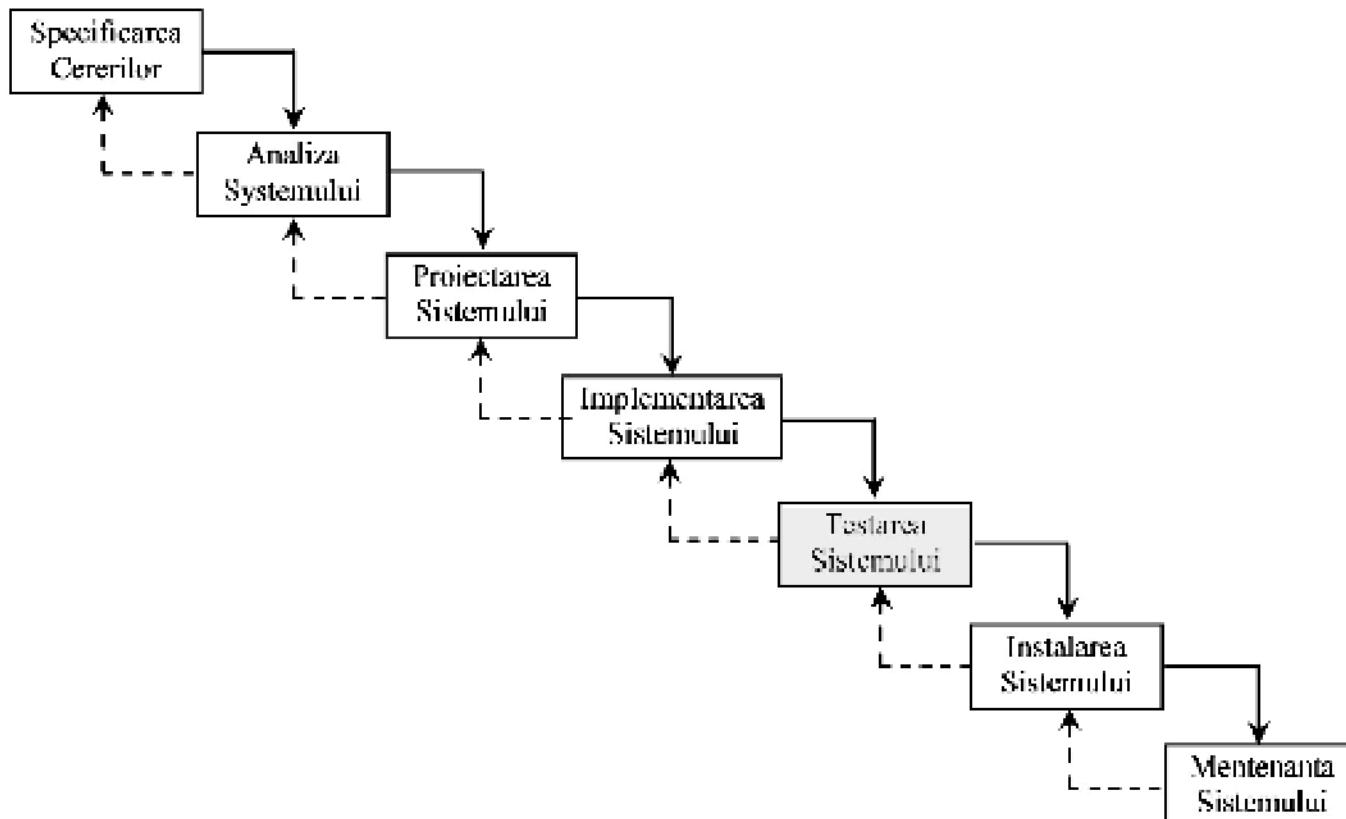
Proiectarea sistemului



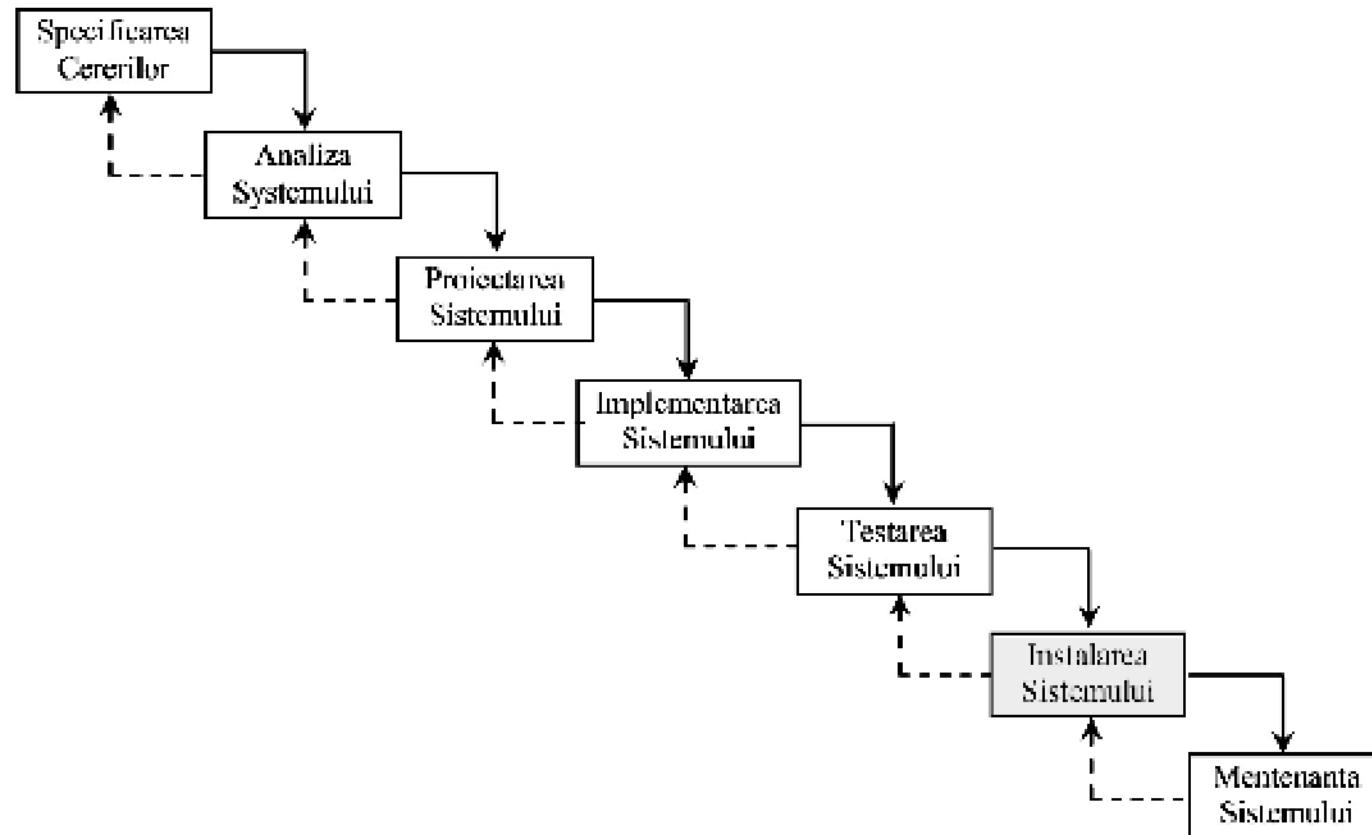
Implementarea sistemului



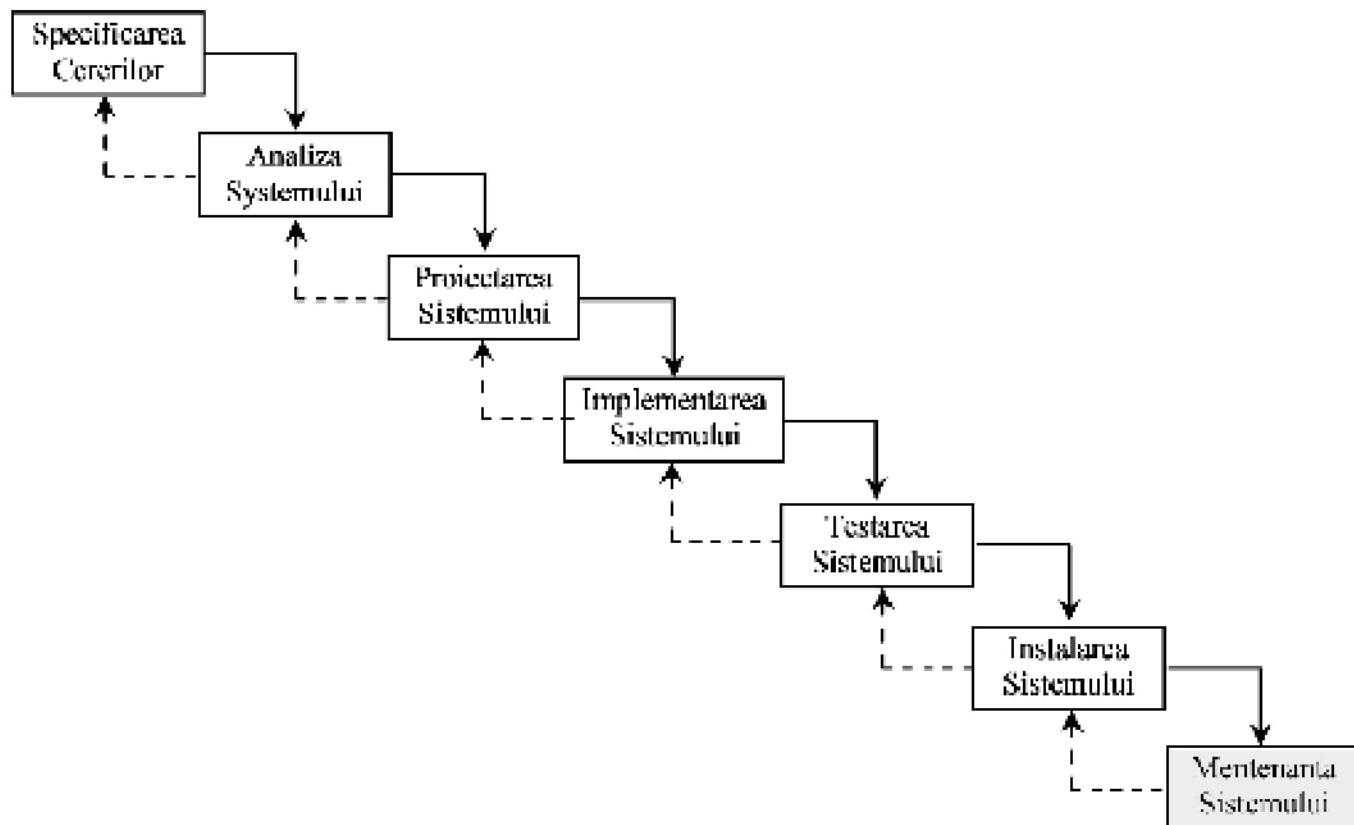
Testarea sistemului



Instalarea sistemului



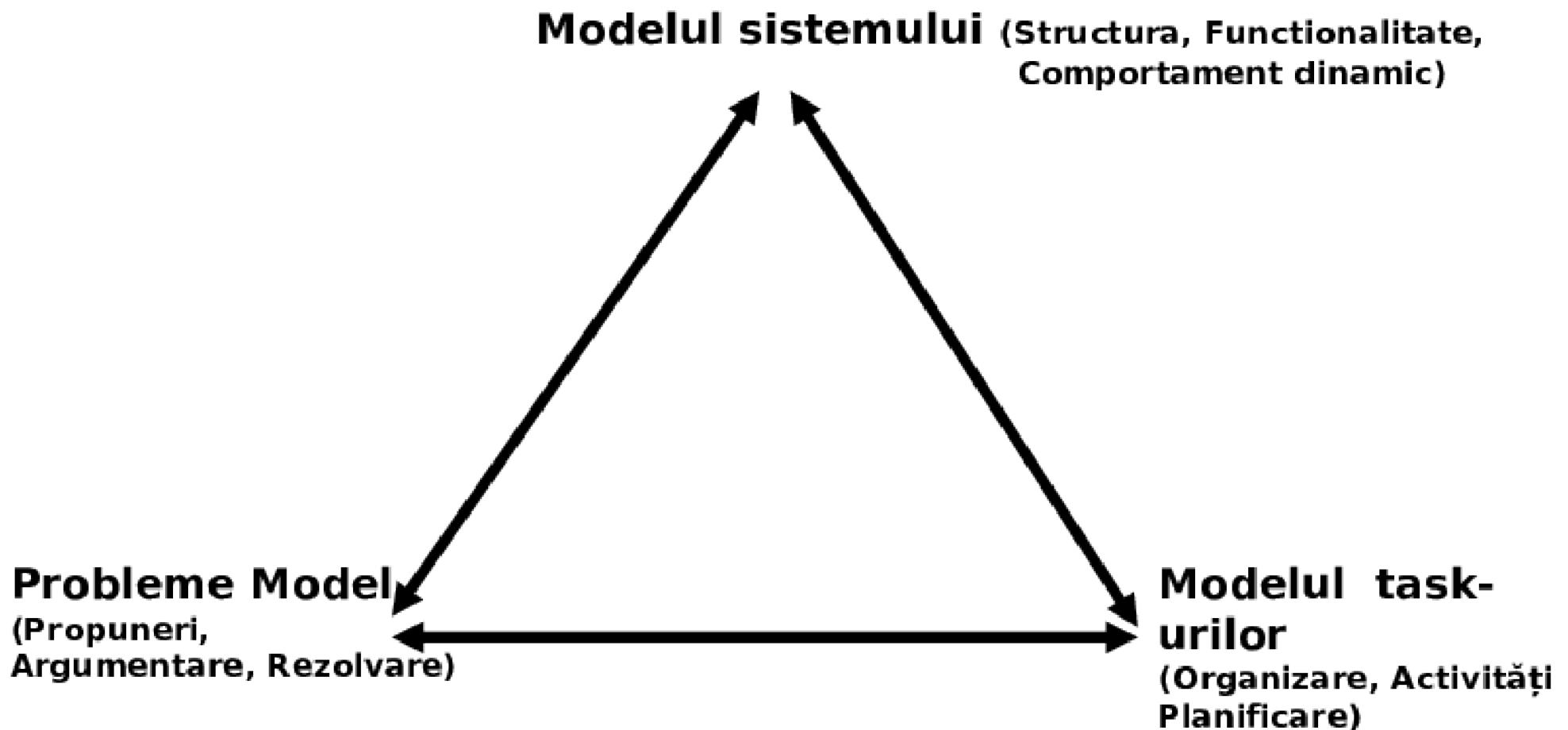
Intreținerea sistemului



Modelele sunt folosite pentru a furniza descrierile abstracte

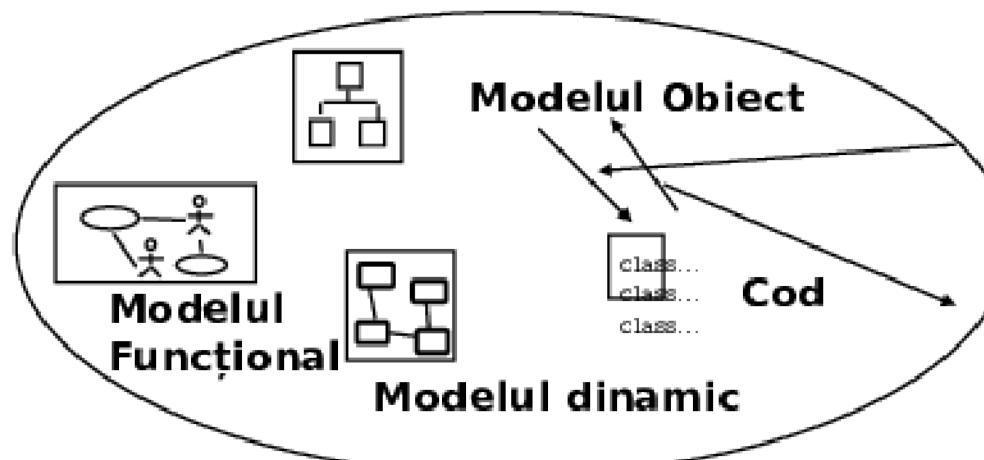
- Modelul sistem:
 - Modelul obiect:
 - Modelul funcțional:
 - Modelul dinamic:
- Modelul task-urilor:
 - Harta PERT: Care sunt legăturile dintre task-uri?
 - Planificarea: Cum poate fi aceasta realizată în durata de timp rezervată?
 - Harta organizațională: Care sunt rolurile în proiect sau organizație?
- Problemele Modelului:

Relațiile modelului

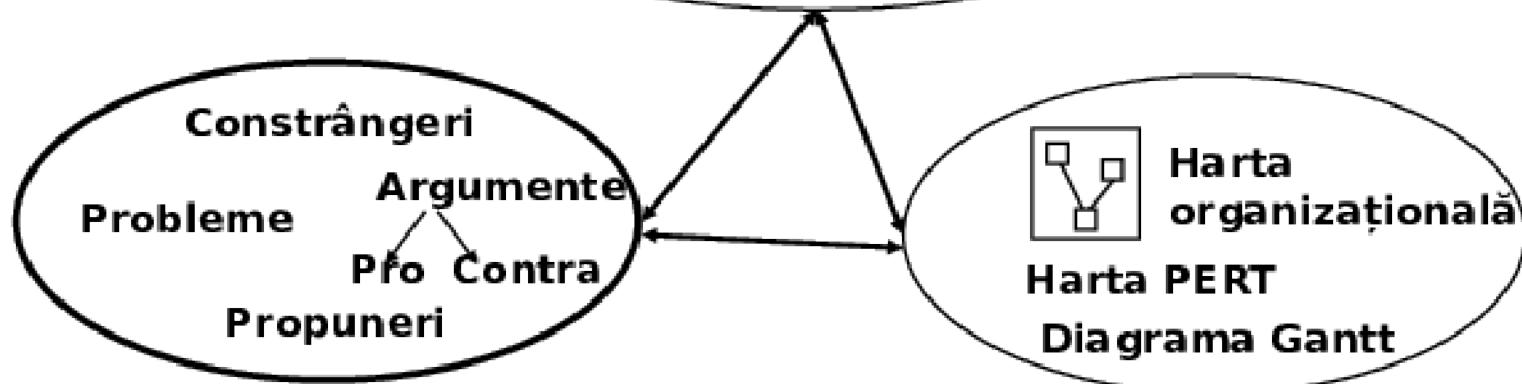


Problema modelării

Modelele Sistemului



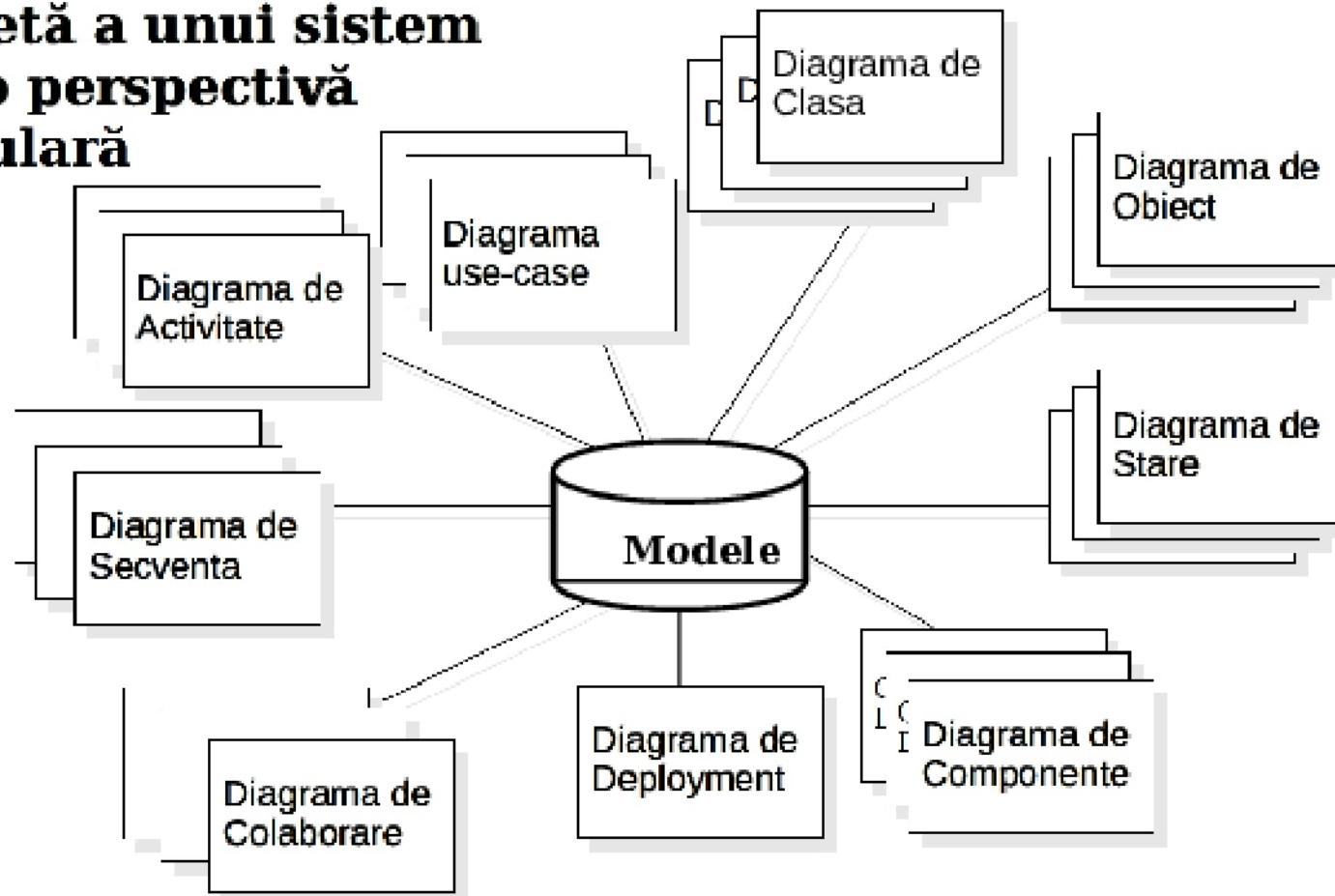
Projecțare normală
↓
Analiza codului Existenter
(Reverse Engineering)



Problemele Modelului

Modele Task-urilor

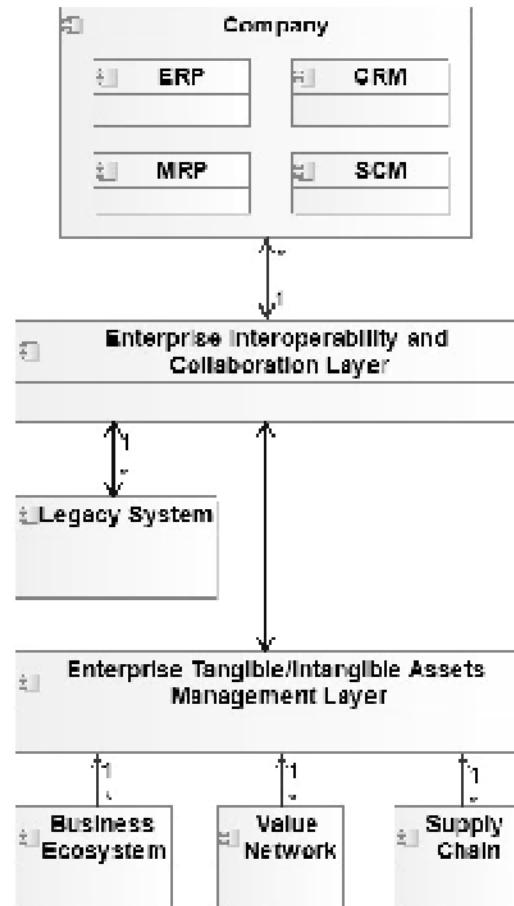
O diagramă este o reprezentare vizuală într-un model
Un model este o descriere
completă a unui sistem
dintr-o perspectivă
particulară



Tipuri de proiectare specifice

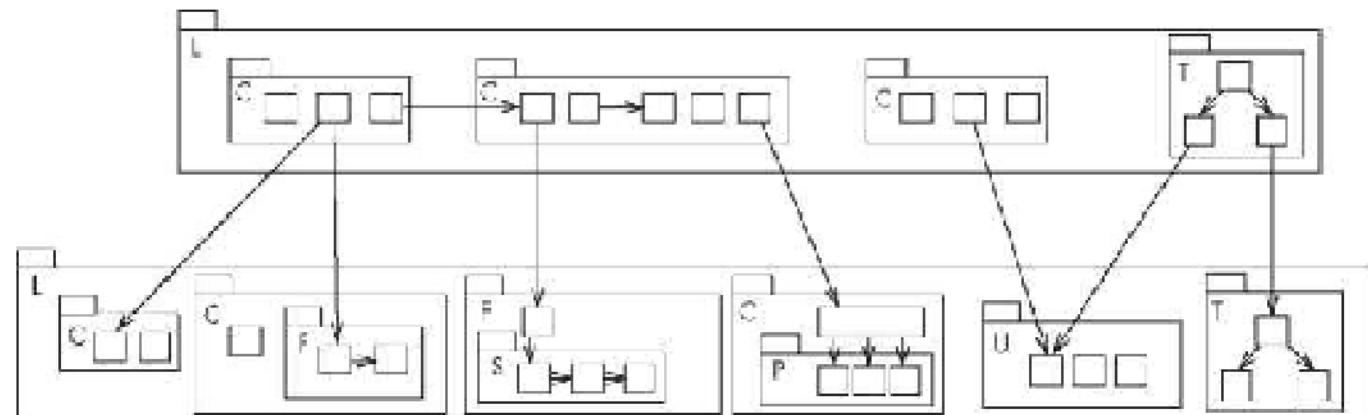
- Proiectare arhitecturală:
- Proiectarea claselor
- proiectarea interfeței vizuale
- proiectarea bazelor de date
- proiectarea algoritmilor
- proiectarea protocoalelor

Bune practici în proiectare(BPP) Divide and conquer

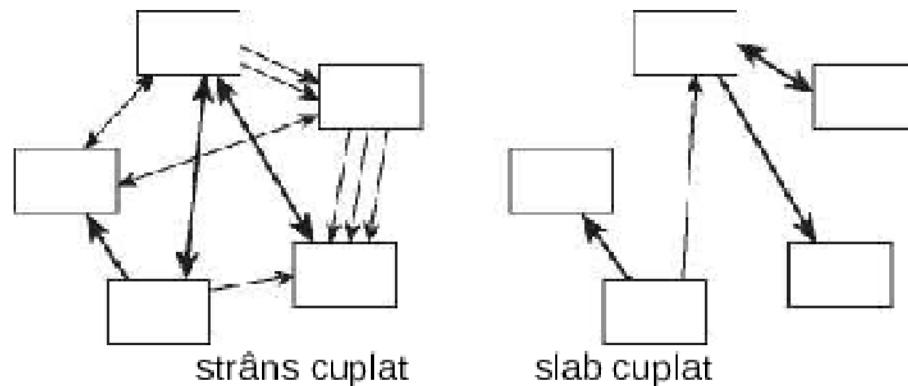


BPP - Increase cohesion where possible

- Coeziunea: "Aplică D&C dar păstrează cât mai "
- Tipuri de coeziune posibile:
 - Funcțională
 - La nivel de strat arhitectural
 - La nivel de comunicații
- Secvențială
- Procedurală
- Temporală



BPP - Reduce coupling where possible



- Cuplarea apare atunci când sunt interdependențe între un modul sau altul
- În figură sunt prezentate cele două cazuri dominante sisteme cu coeziune stransă sau slabă.
- Dacă un sistem are o coeziune mai strânsă (câteodată din cauza unei proiectări deficitare câteodată datorită limitărilor specifice paradigmelor de programare și a bibliotecilor sau facilităților tehnologiei folosite) el este mai greu de înțeles sau și de modificat deci are o mențenanță problematică de-a lungul ciclului de viață.

Tipuri de cuplare:

- Conținut
- Comun
- Control
- Etichetă
- Date
- Apel de funcție
- Utilizare de tip
- Incluziune/Import
- Externă

BPP - Keep the level of abstraction as high as possible

- ascund pădurea ca să nu mă încurc de copaci
- detaliile despre copaci frunze și veverișe pot fi furnizate
 - la o etapă ulterioară pe ciclul de proiectare
 - prin compilator sau la runtime
 - prin valori implicite

BPP - Increase reusability where possible

- Există două principii complementare legate de reutilizare.
- “proiectează pentru reutilizare”
- “proiectează prin reutilizare”

BPP - Reuse existing designs and code where possible

BPP - Design for flexibility

BPP - Anticipate obsolescence

BPP - Design for portability

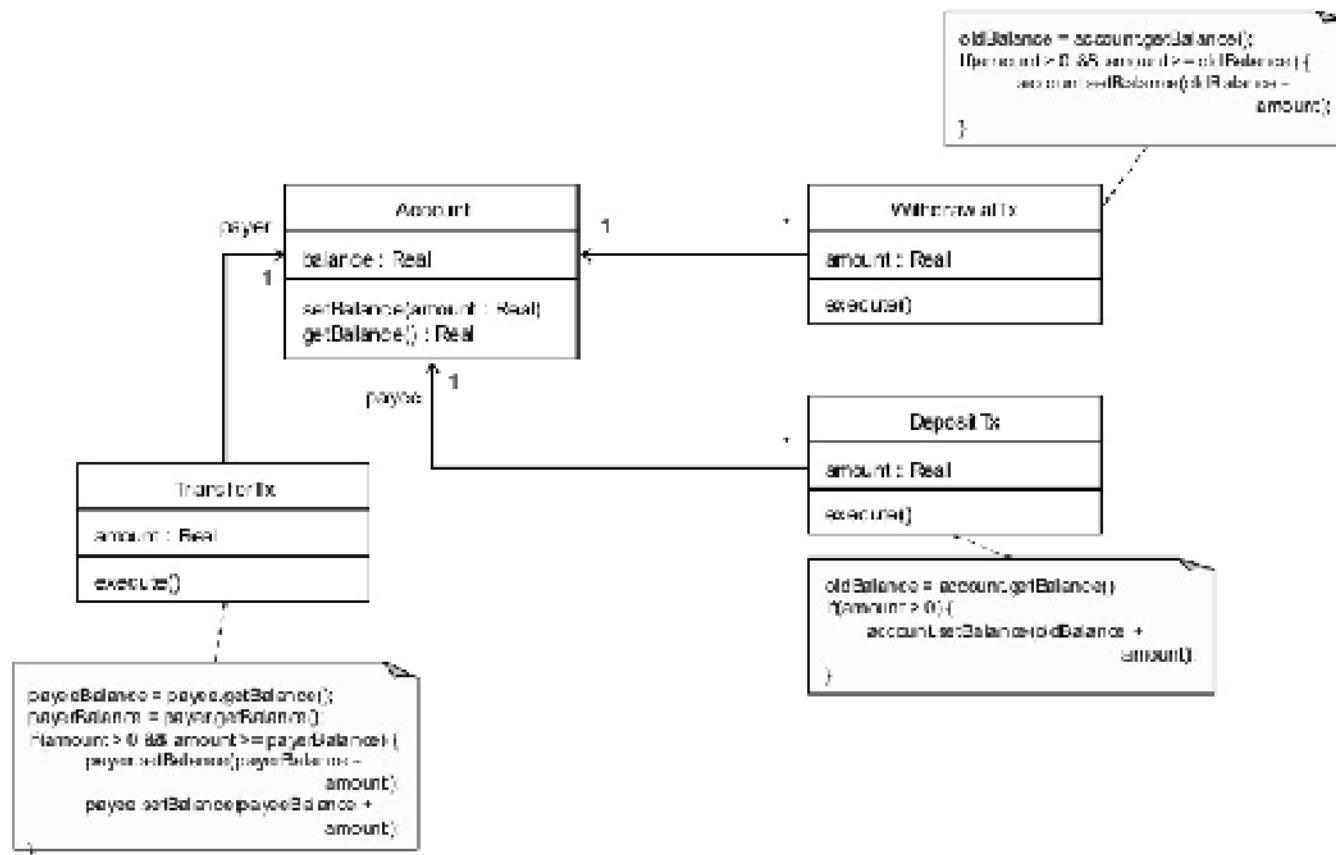
BPP - Design for testability

Scopurile proiectării OOP

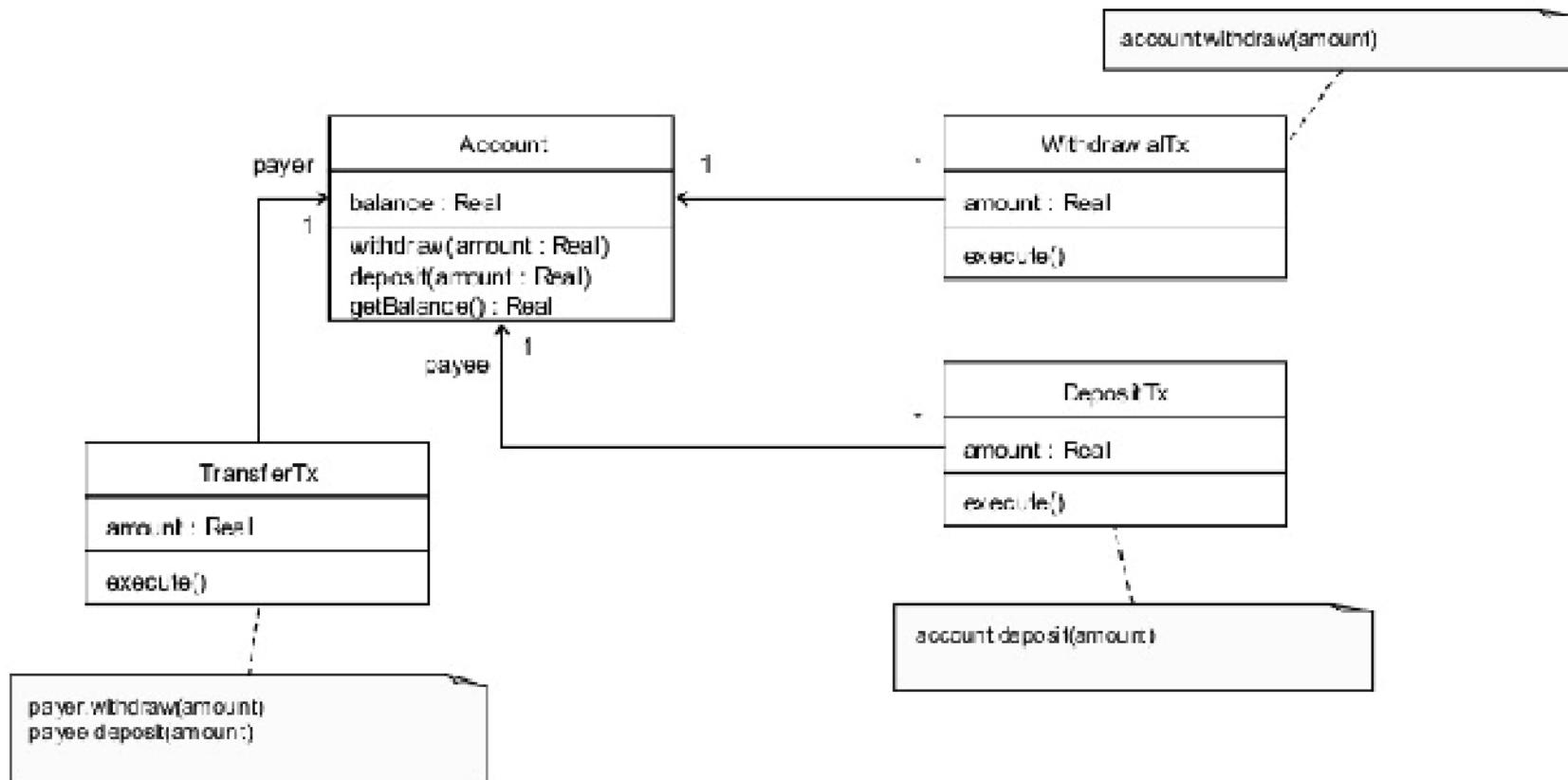
Principii OOP pentru proiectarea claselor

- Coeziunea Claselor
- Închisă vs. Deschisă
- Răspundere Unică
- Separarea Interfețelor
- Dependența Inversă
- Substituția Liskov
- Legea lui Demeter
- Reutilizarea Abstracțiilor

Coeziunea Claselor - Înainte..

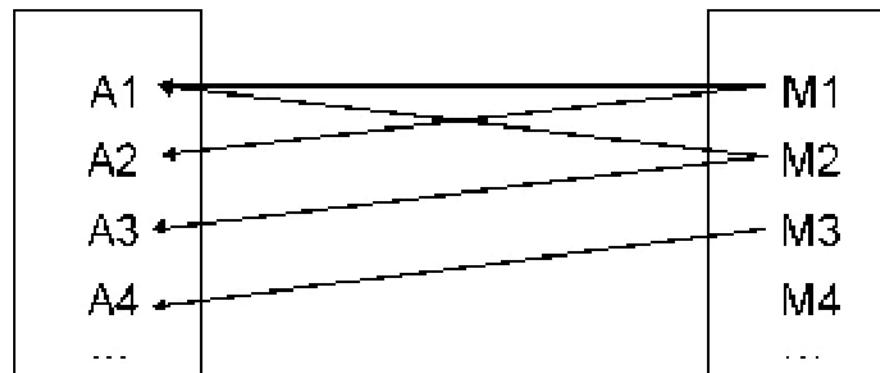


Coeziunea Claselor - și după refactorizare



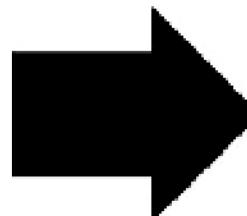
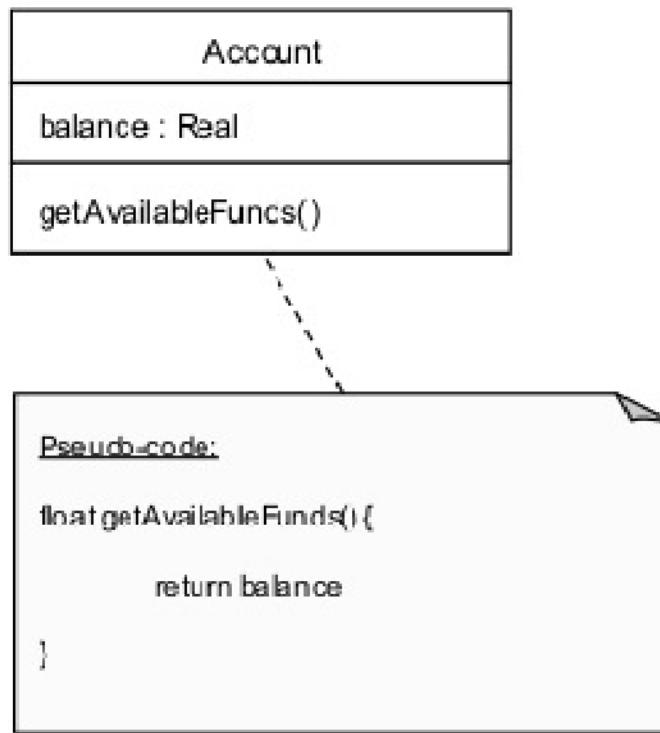
Metrici pentru măsurarea coeziunii

- Clasa are un număr de A - attribute
- Clasa are un număr de M - metode
- Fiecare atribut A_i este accesat de $R(A)$ metode

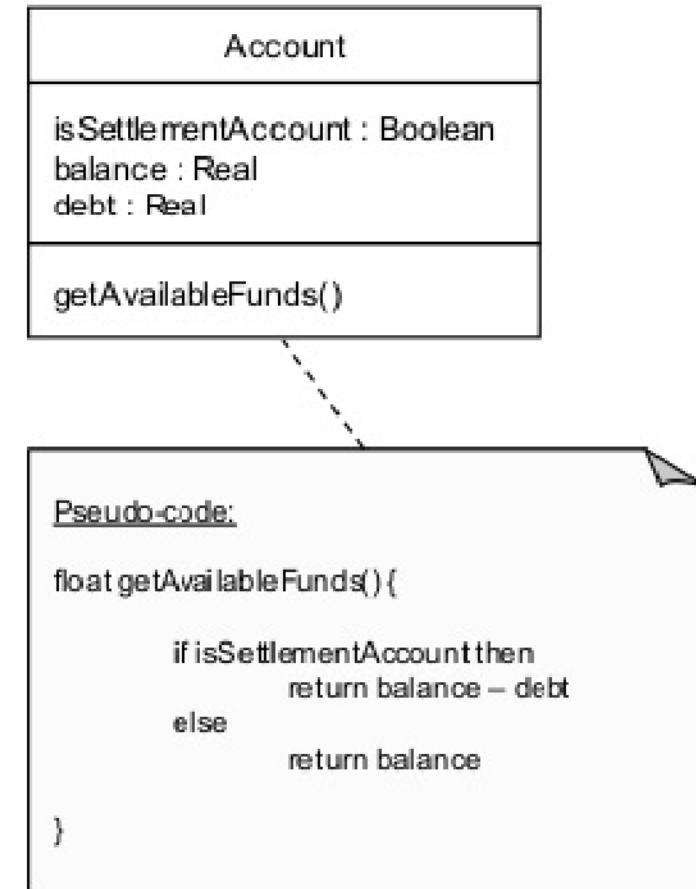


$$LCM = \frac{\sum_{i=0}^{A-1} R_i(A)}{A} - M$$
$$\frac{1-M}{1-M}$$

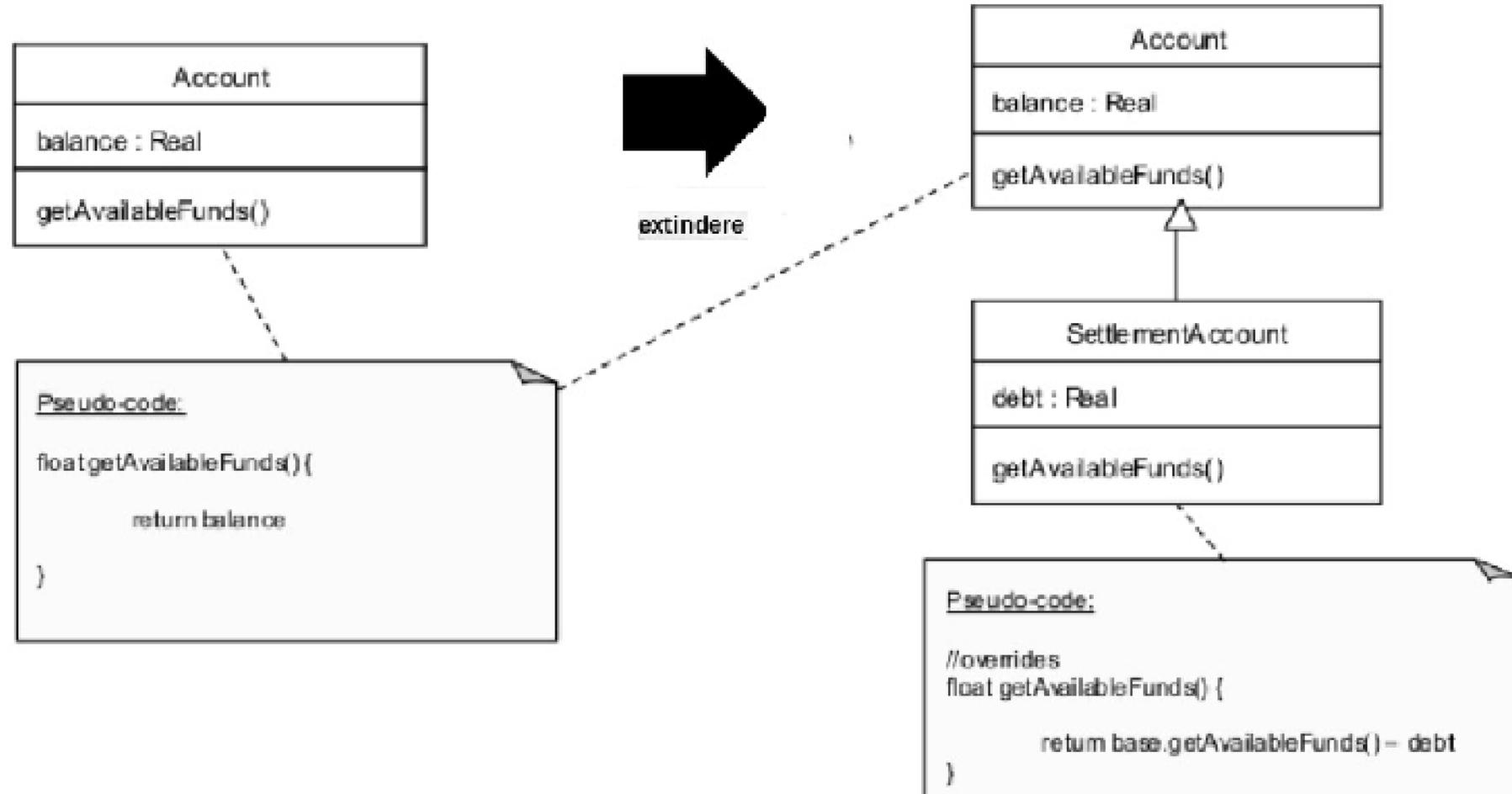
Închisă vs. Deschisă? - Înainte



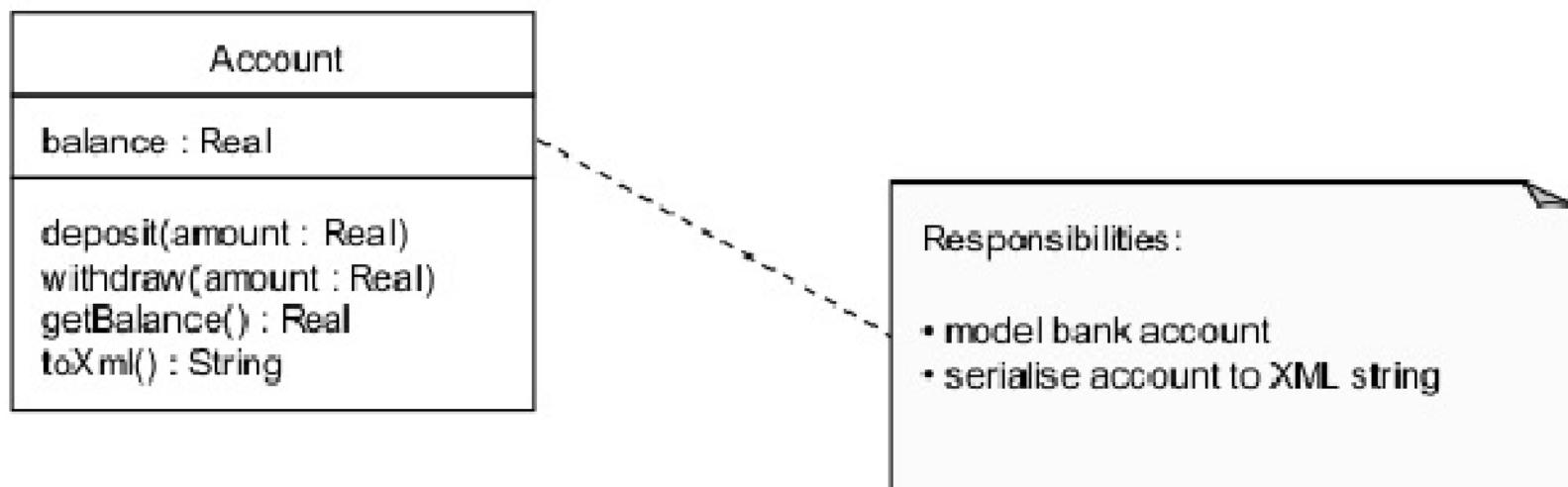
modificare



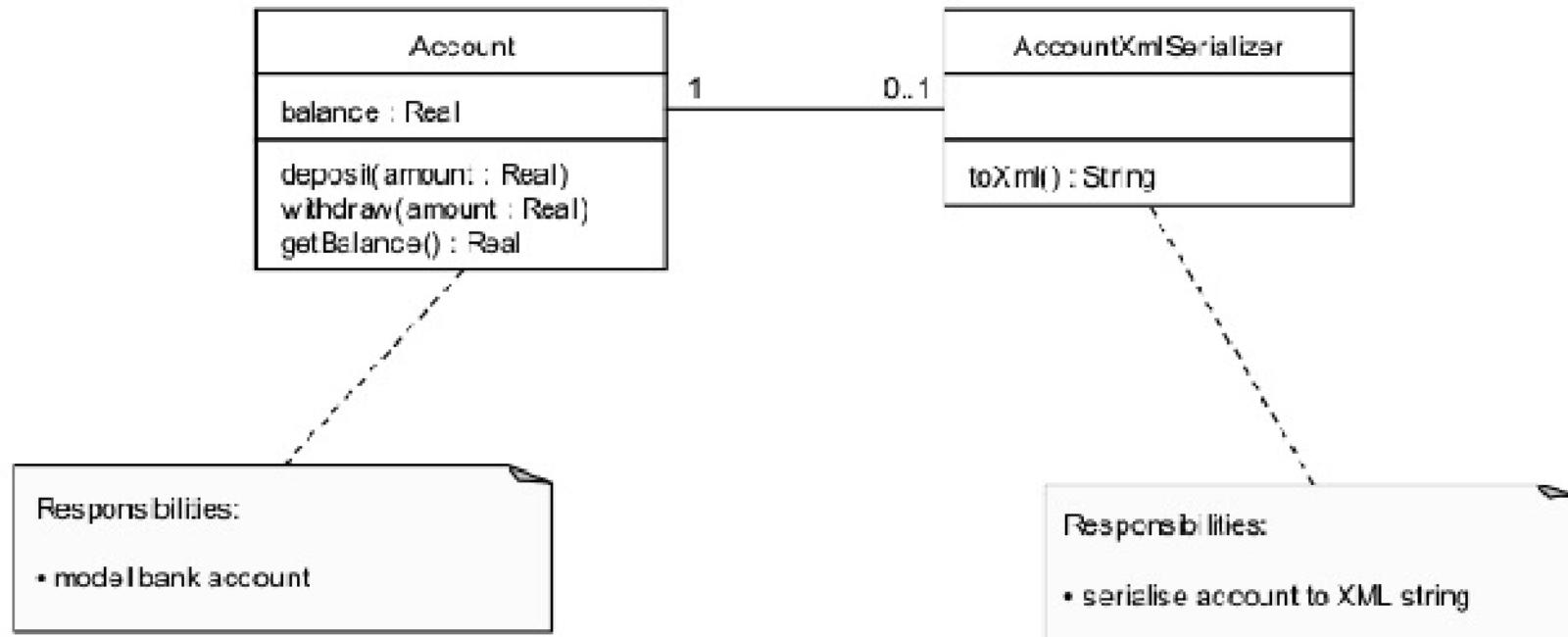
Închisă vs. Deschisă? - după refactorizare



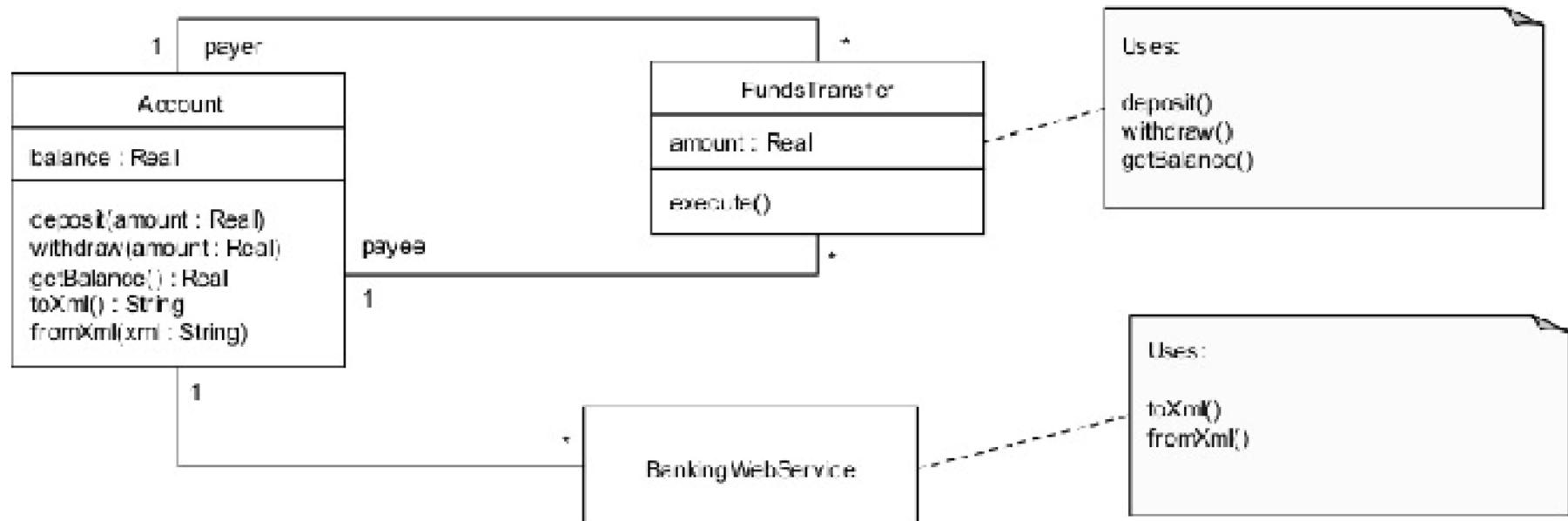
Răspundere Unică - înainte...



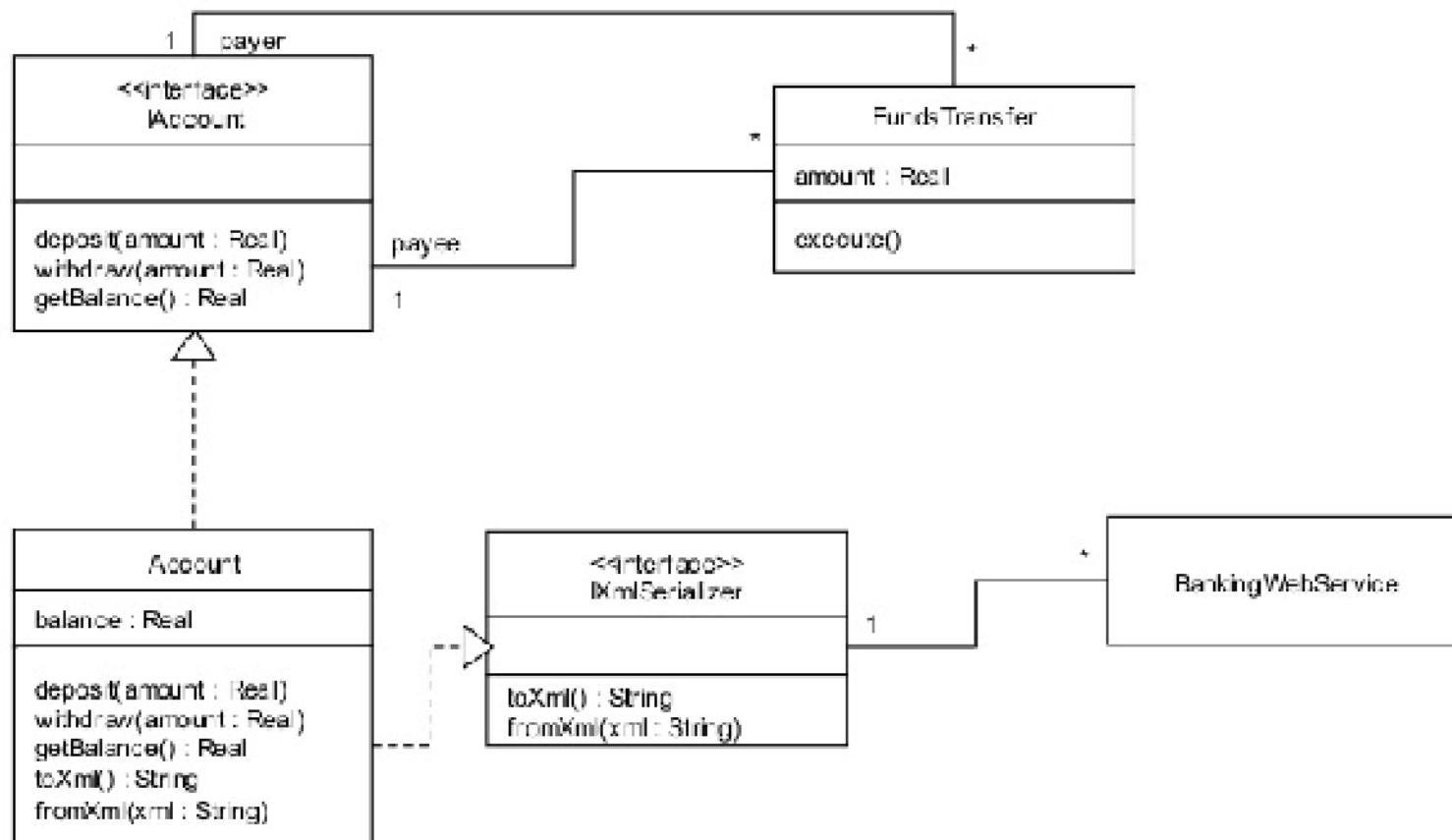
Răspundere Unică - după refactorizare



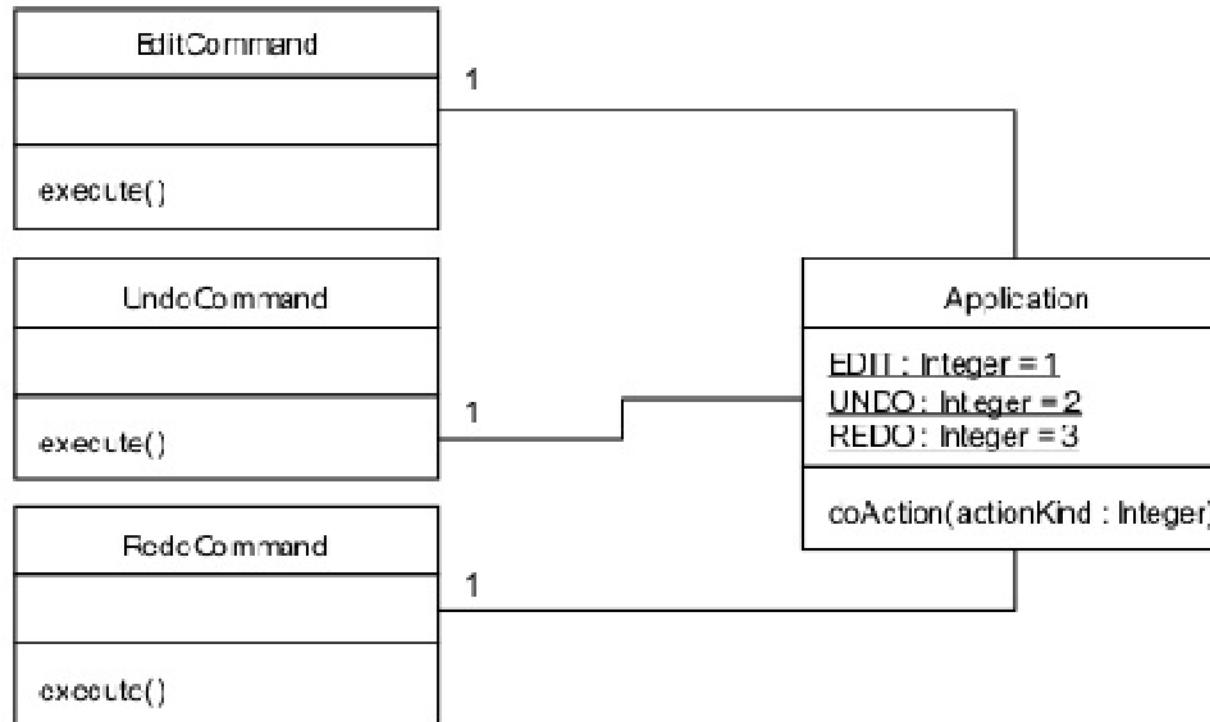
Separarea Interfețelor - Înainte...



Separarea Interfețelor - după refactorizare



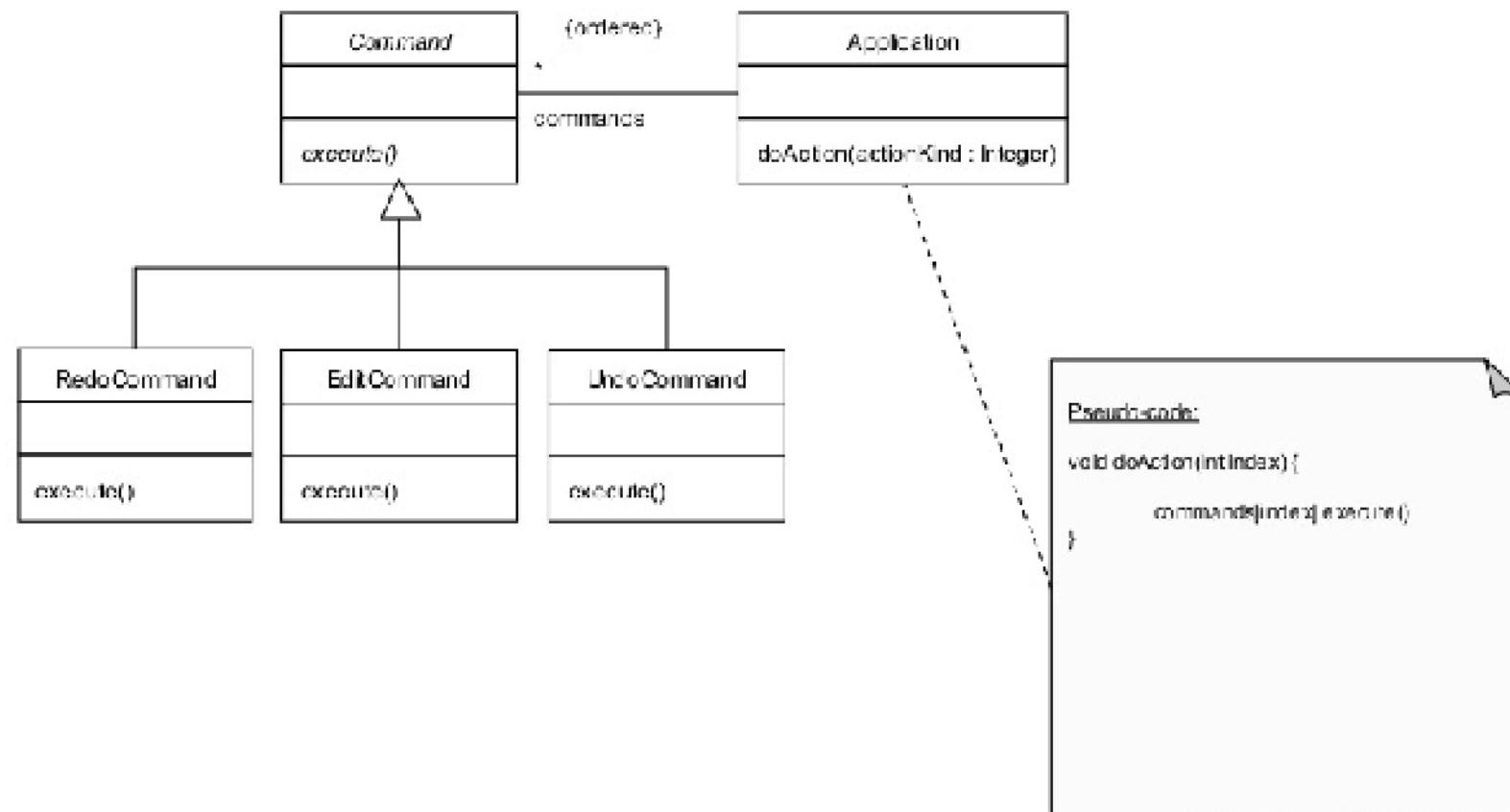
Dependență/Relația Inversă - înainte ...



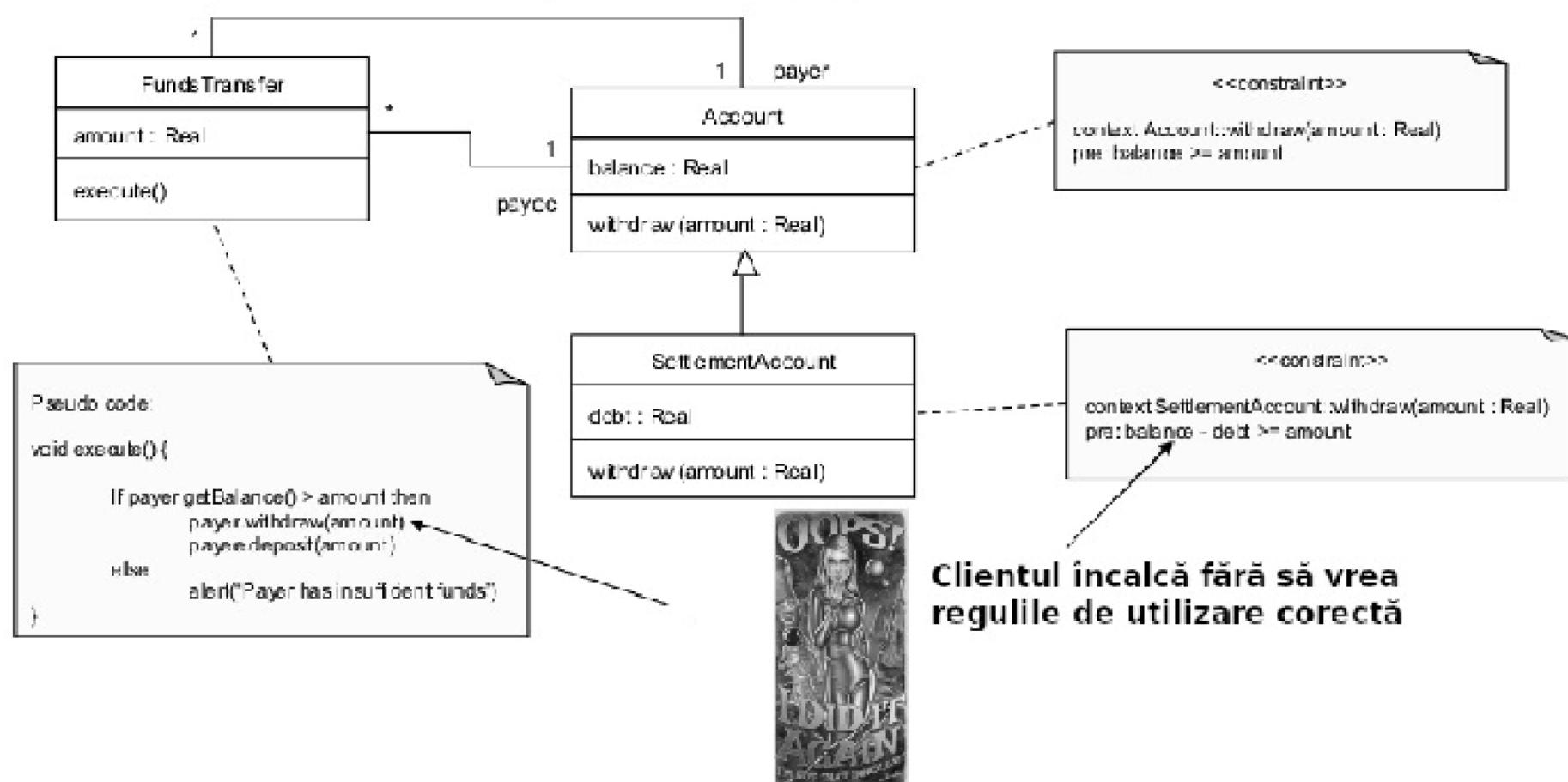
Pseudo-code:

```
void doAction(n:actionKind) {
    switch actionKind {
        case EDIT:
            editCommand.execute()
        case UNDO:
            undoCommand.execute()
        case REDO:
            redoCommand.execute()
    }
}
```

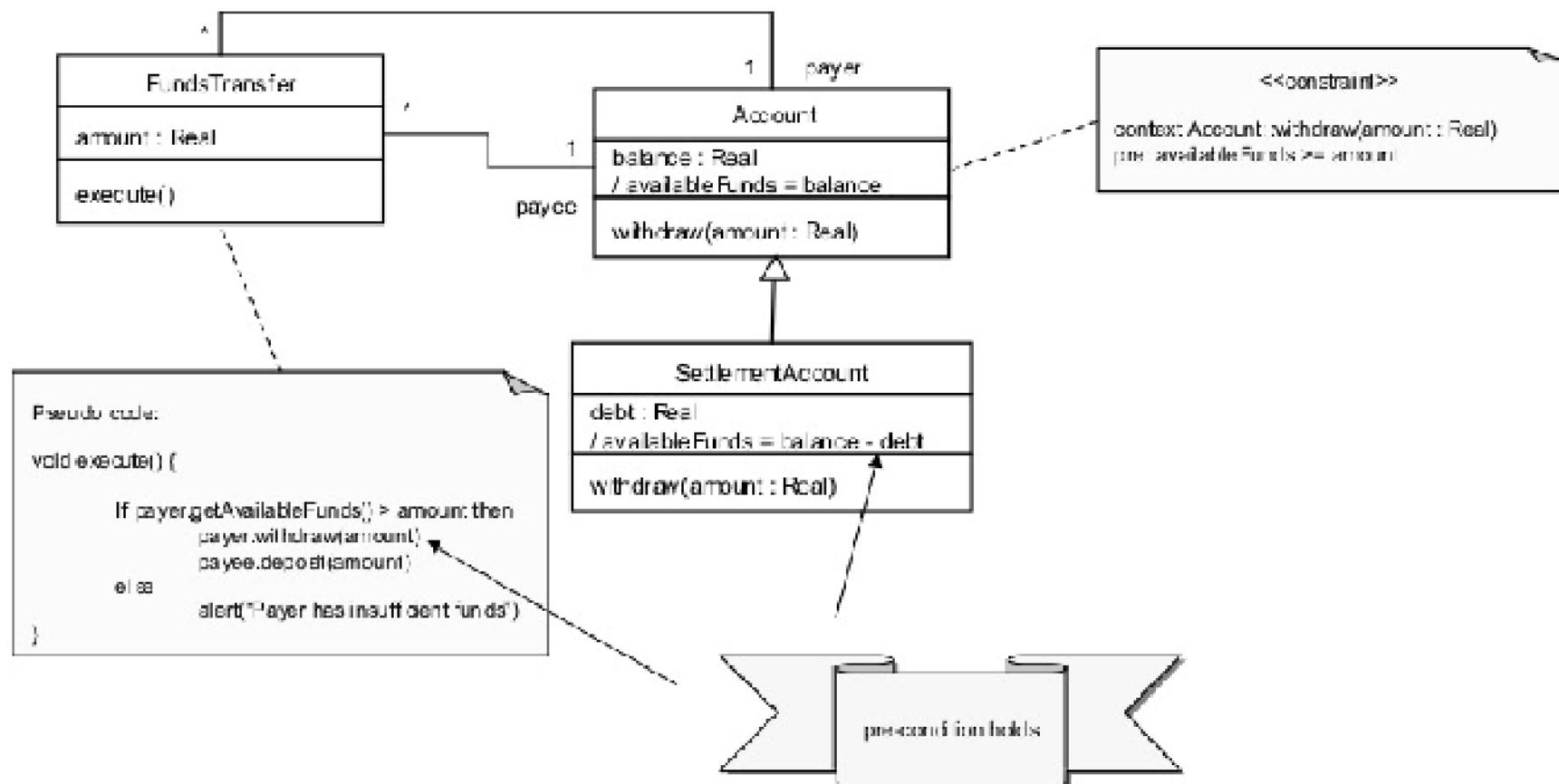
Dependență Inversă - după Factorizare



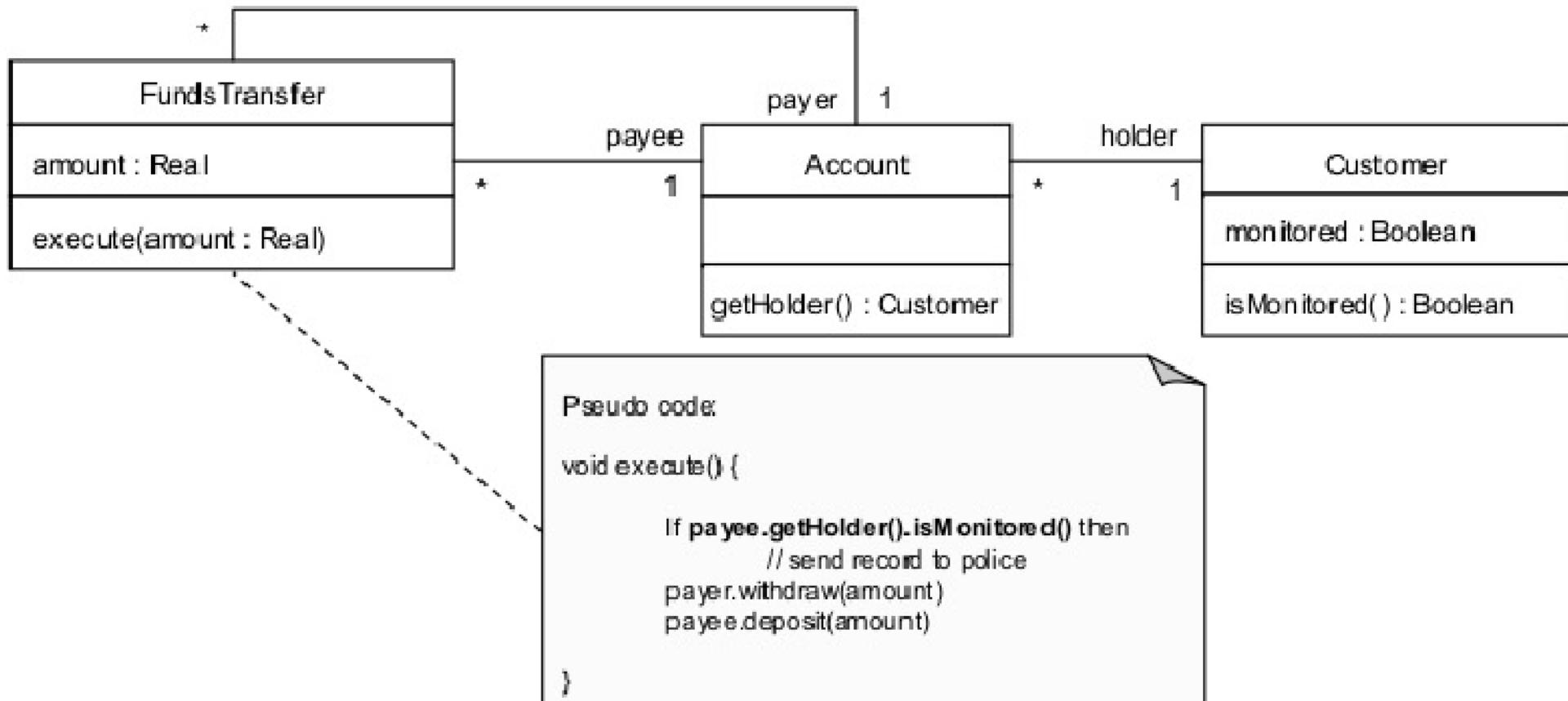
Substituția Liskov - înainte...



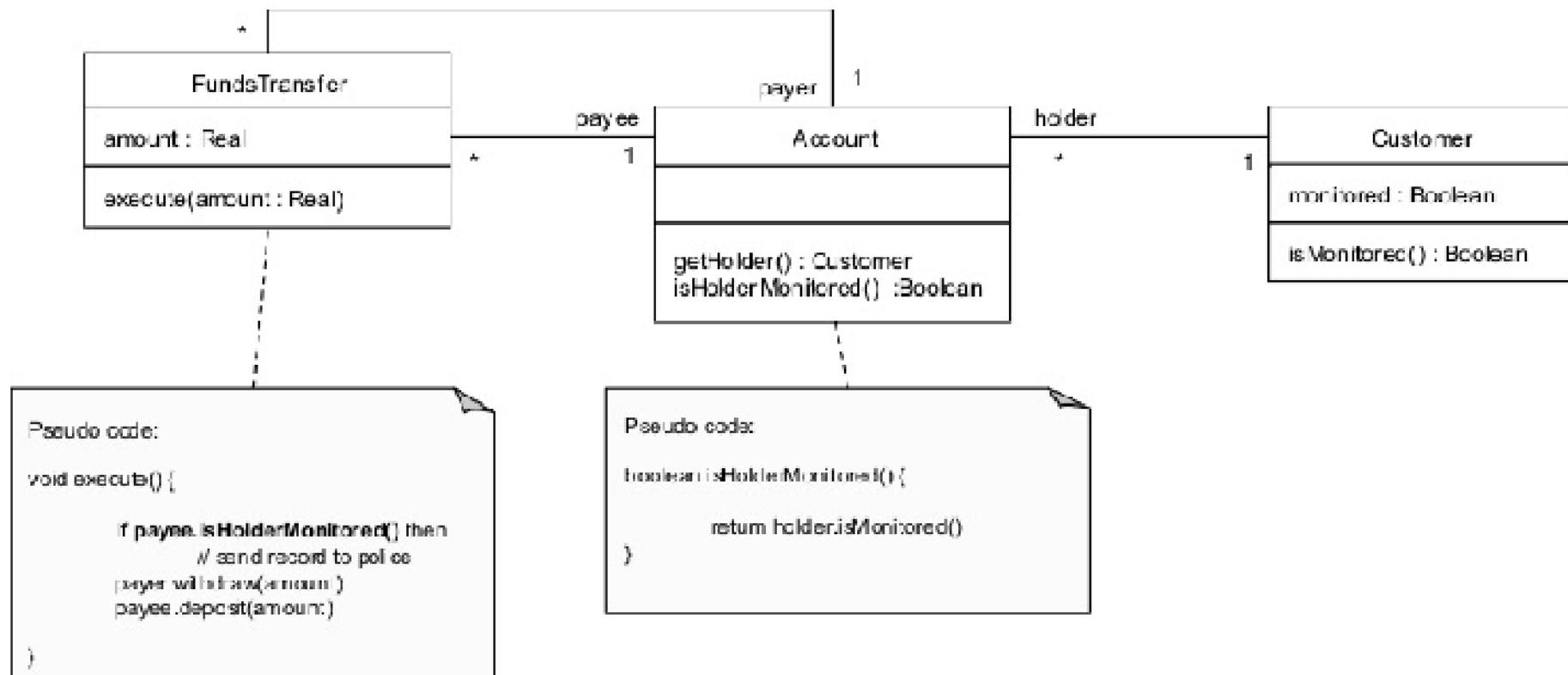
Substituția Liskov - după Refactorizare un curs cu use case și ce mai este aici



Legea lui Demeter - înainte...

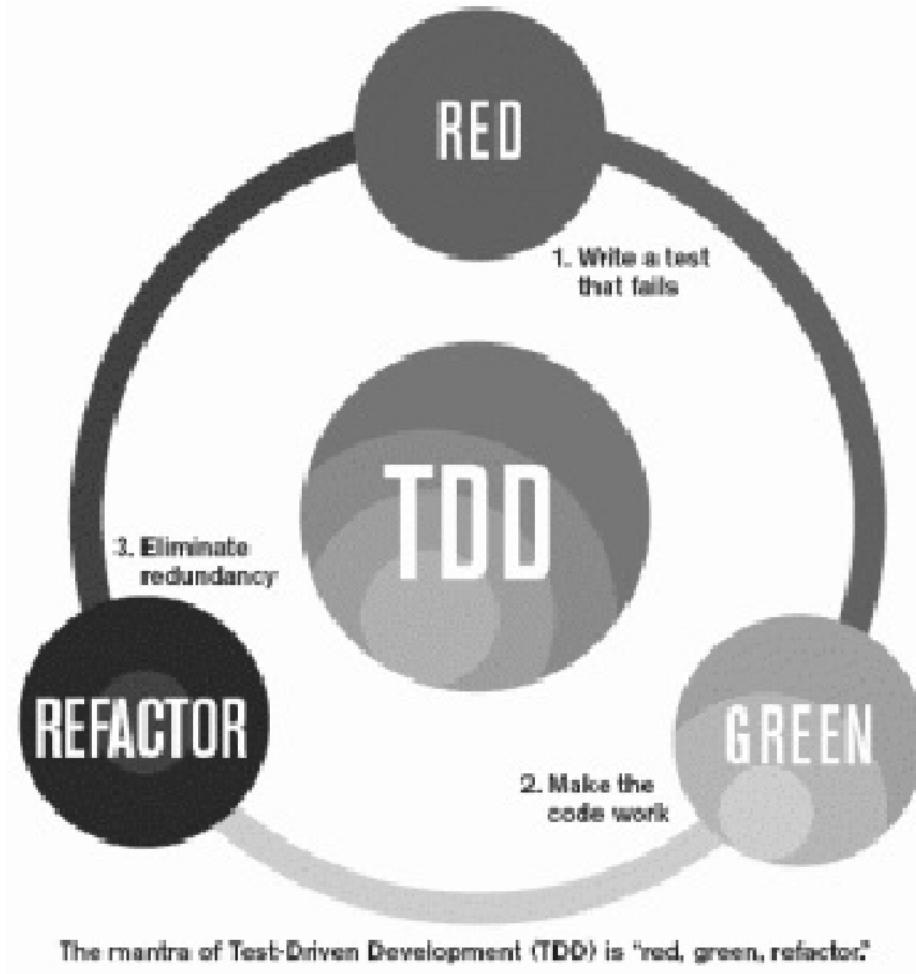


Legea lui Demeter - după Refactorizare



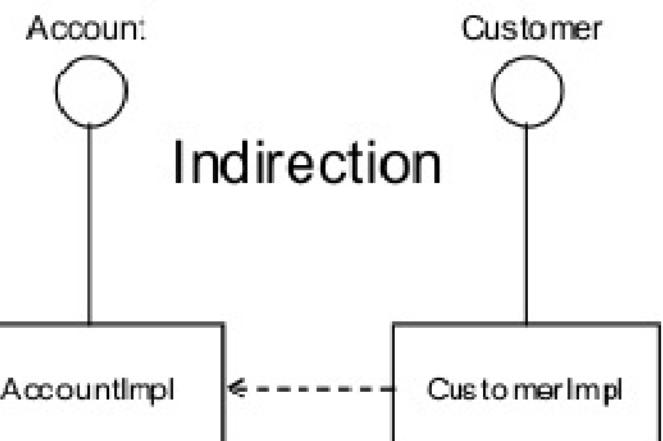
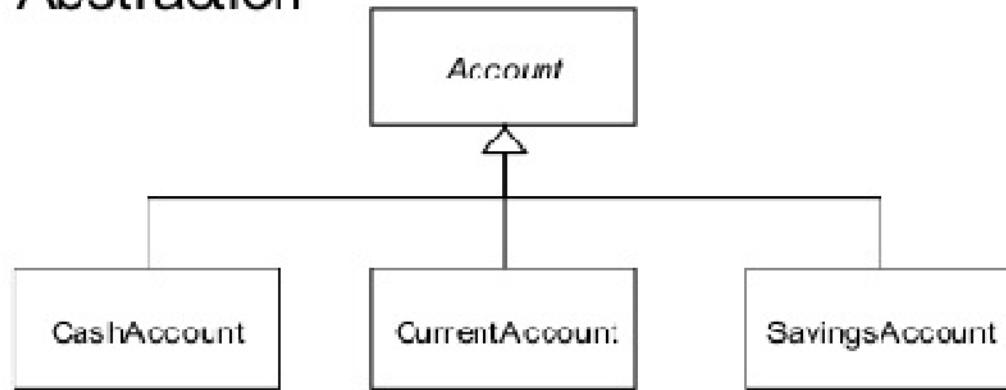
Test-Driven Development (TDD)

```
while(!OutOfBeer())
{
    scrie(un test)
    execută(toate testele)
    scrie(codul întă)
    execută(testele)
    refactorizeaza(codul)
}
```

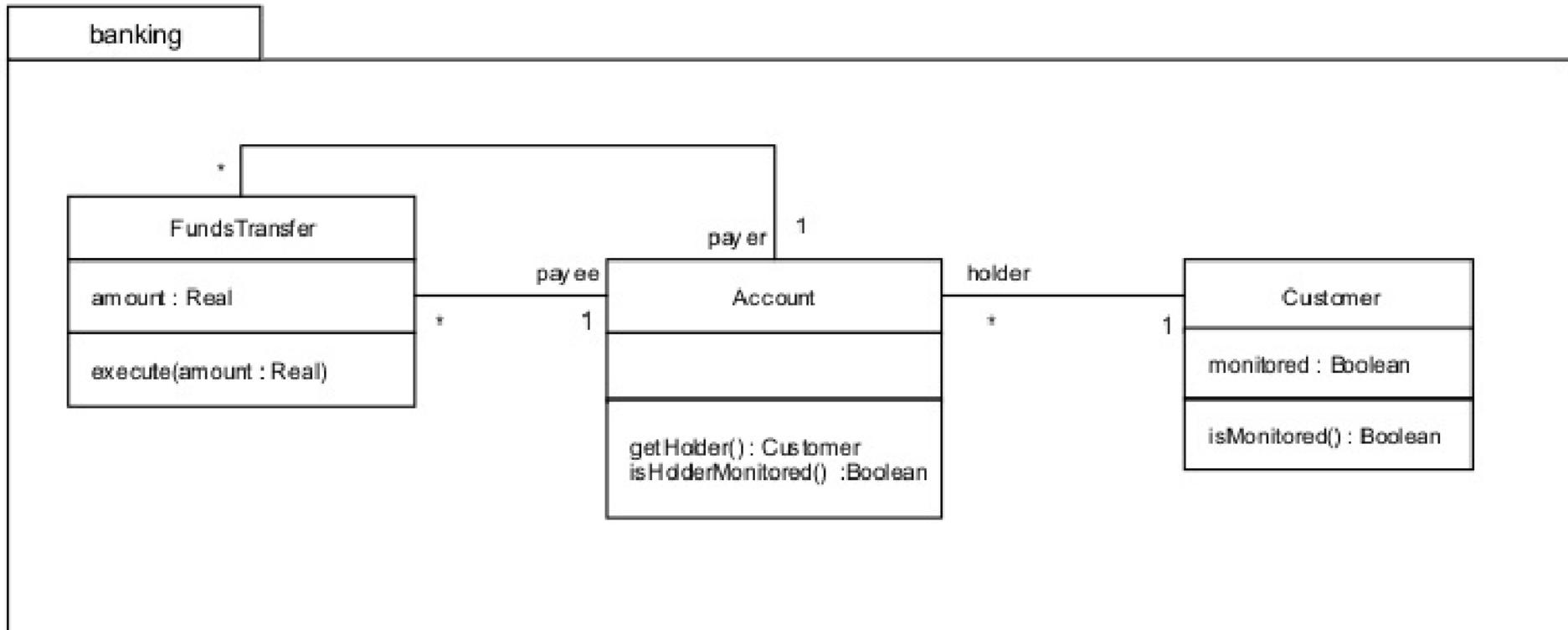


Reutilizarea Abstracțiilor

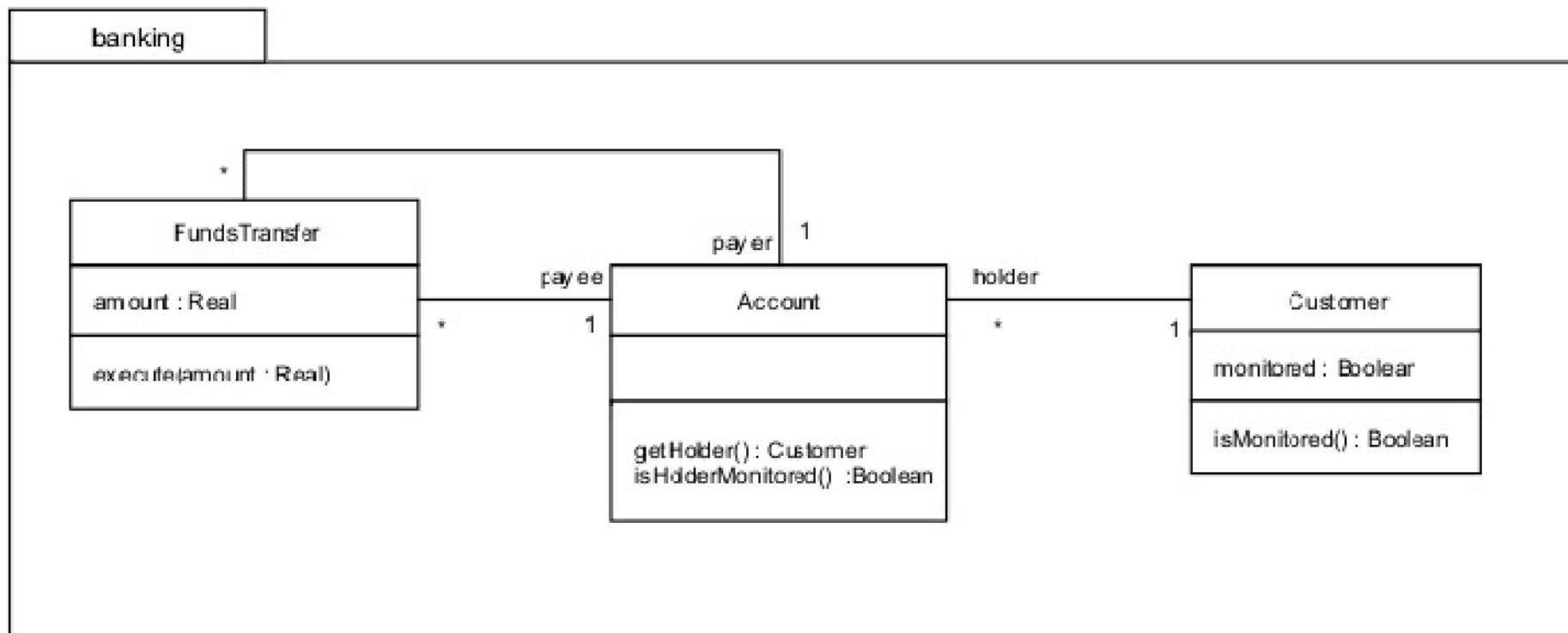
Abstraction



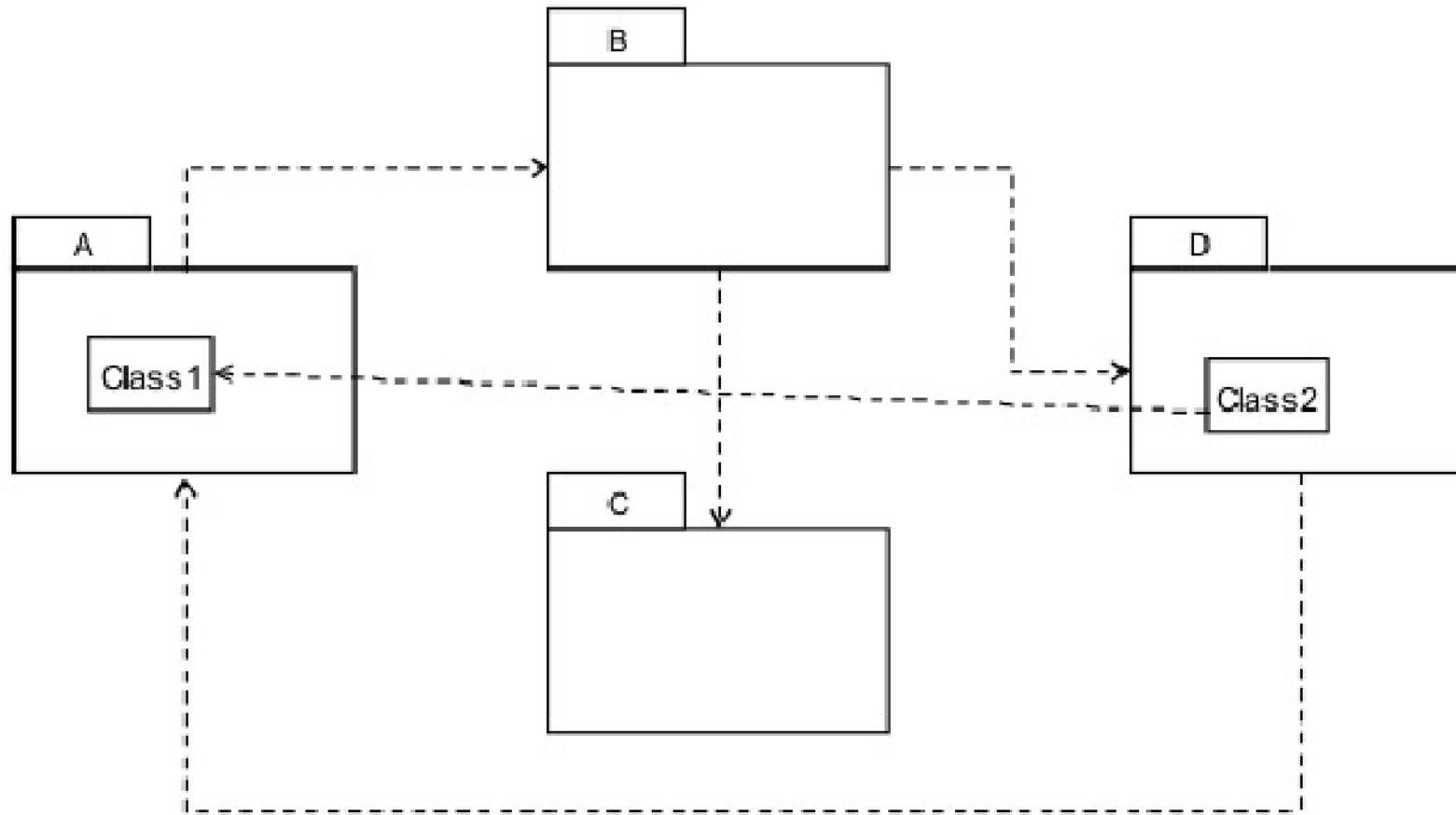
Common Closure



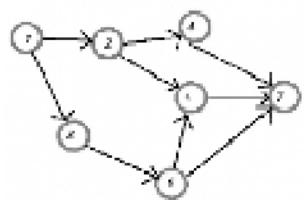
Common Reuse



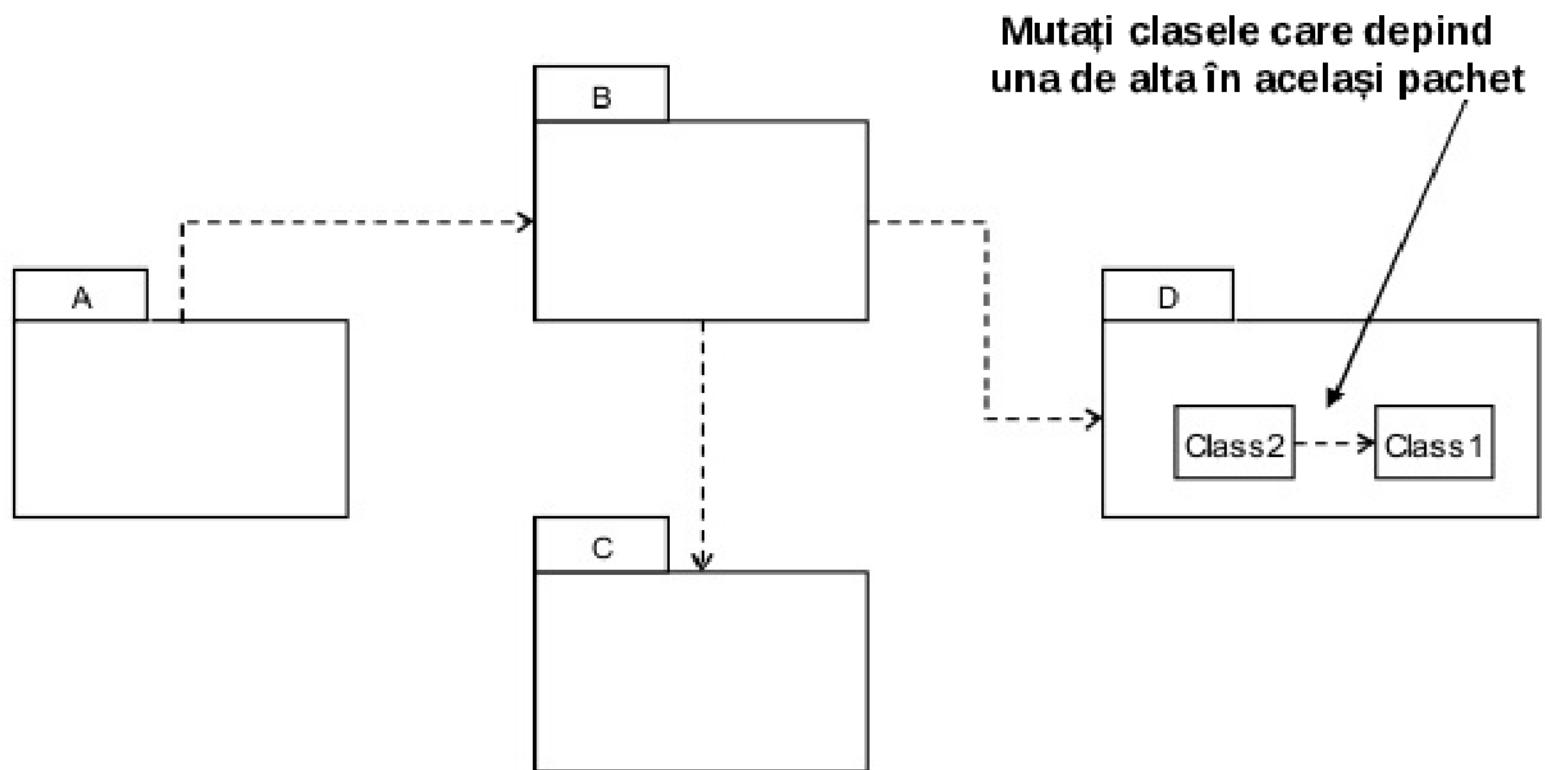
Dependență aciclică



Dependenta aciclică - Refactorizare



DGA

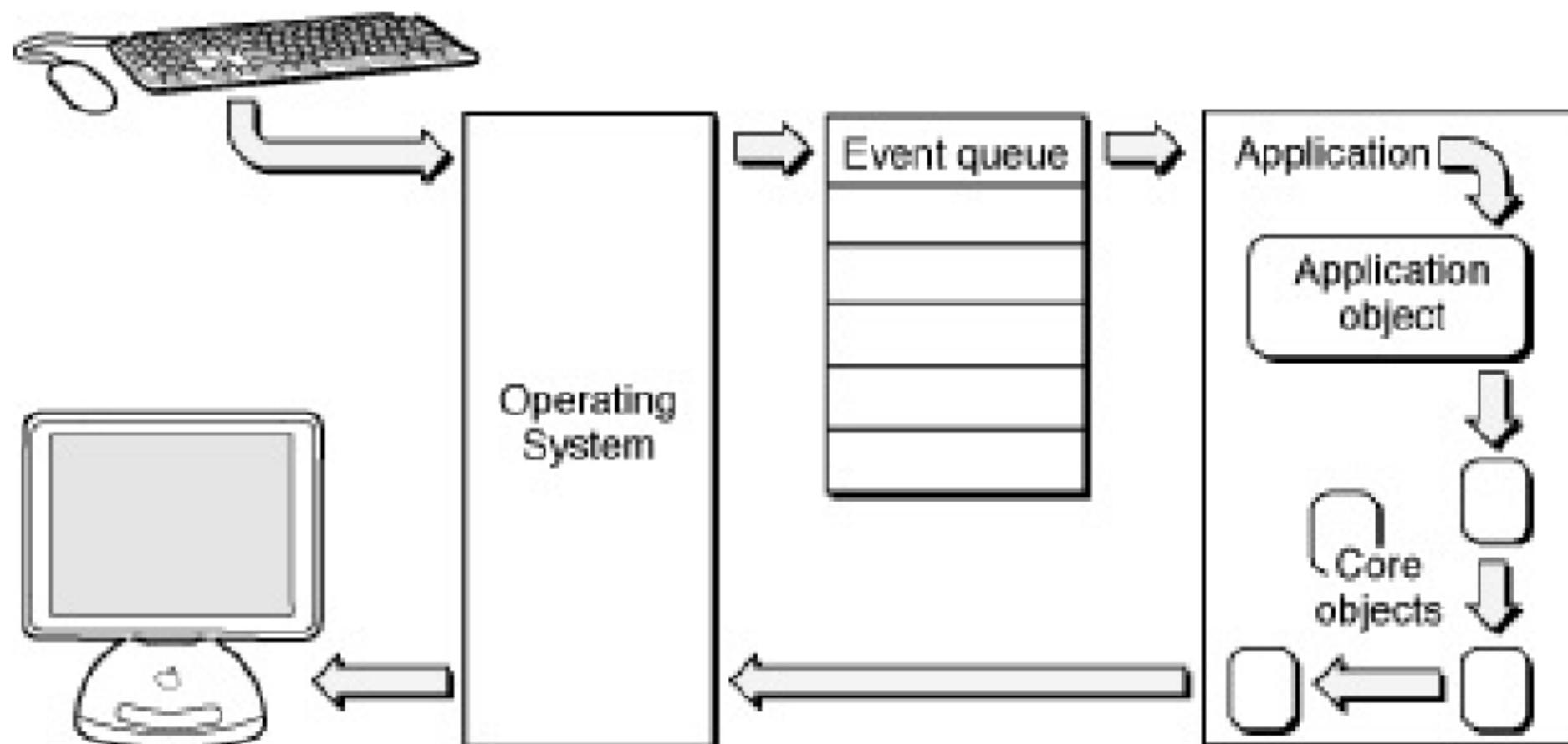


Paradigma Orientata Eveniment

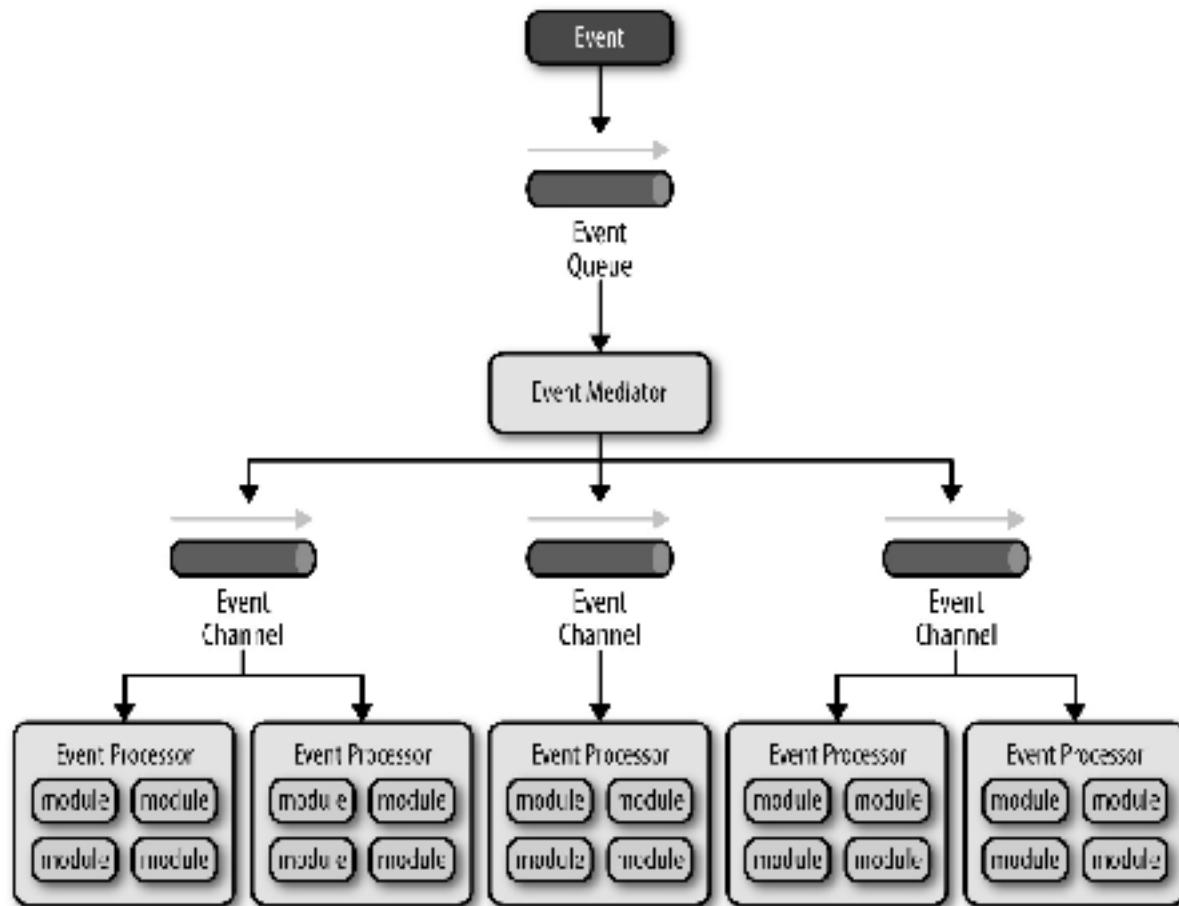
Cursul nr. 5
Mihai Zaharia

Ce este un eveniment?

O manieră de gestionare/tratare



Abordarea modernă de proiectare arhitecturală



Tratare evenimente asincrone

Polling

- **Interrupt-driven**

Event-driven

Paradigma orientată eveniment

Metoda 1 - Polling

Interacțiunea este guvernată de o buclă infinită:

Loop forever:

```
{ i=1..n  
    read input i  
    answer to input i  
    inc i }
```

Metoda 2 - Interrupt-driven

1. Activează dispozitivul, apoi
2. Începe procesarea de bază (instalează sistemul de gestiune a evenimentelor/întreruperi)
3. Așteaptă apariția unei întreruperi
4. La apariția unei întreruperi
 1. Salvează starea curentă (schimbare context)
 2. Încarcă și execută metoda de tratare a întreruperii
 3. Restaurează contextul anterior
 4. Go to #1

Metoda #3: Event-driven

Interacțiunea este din nou guvernată de o buclă:

main()

{

...initializează structurile de date ale aplicației ...

...initializează și lansează în execuție GUI....

// intră în bucla de eveniment

while(true)

{

Event e = **get_event();**//primește evenimentul

process_event(e); // tratează evenimentul

}

}

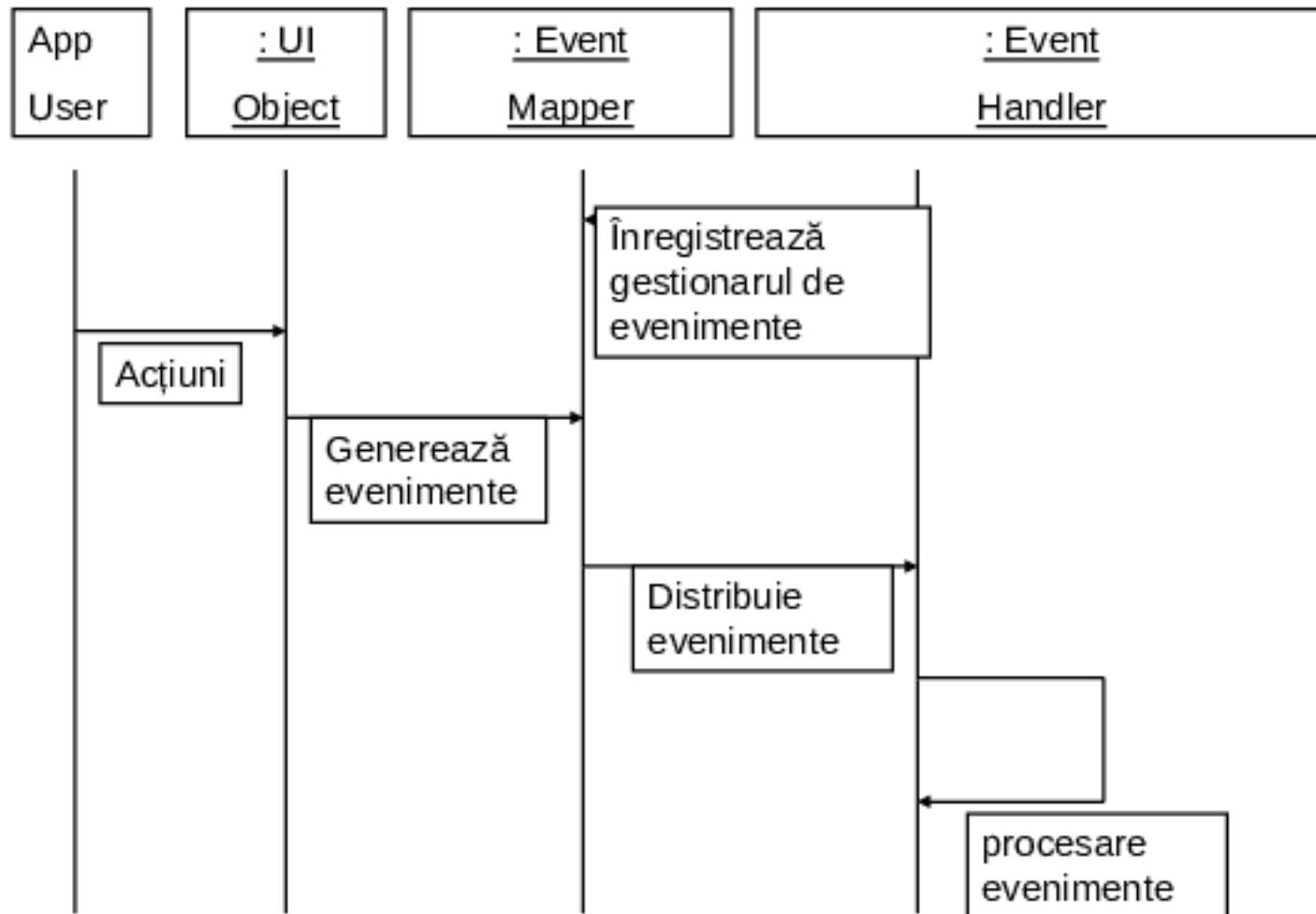
Avantajele procesării orientate eveniment

- Sunt mai portabile
- Permit tratarea mai rapidă
- Se pot folosi în time-slicing)
- Încurajează reutilizarea codului
- se potrivesc cu oop

Componentele unui program simplu bazat pe EDP

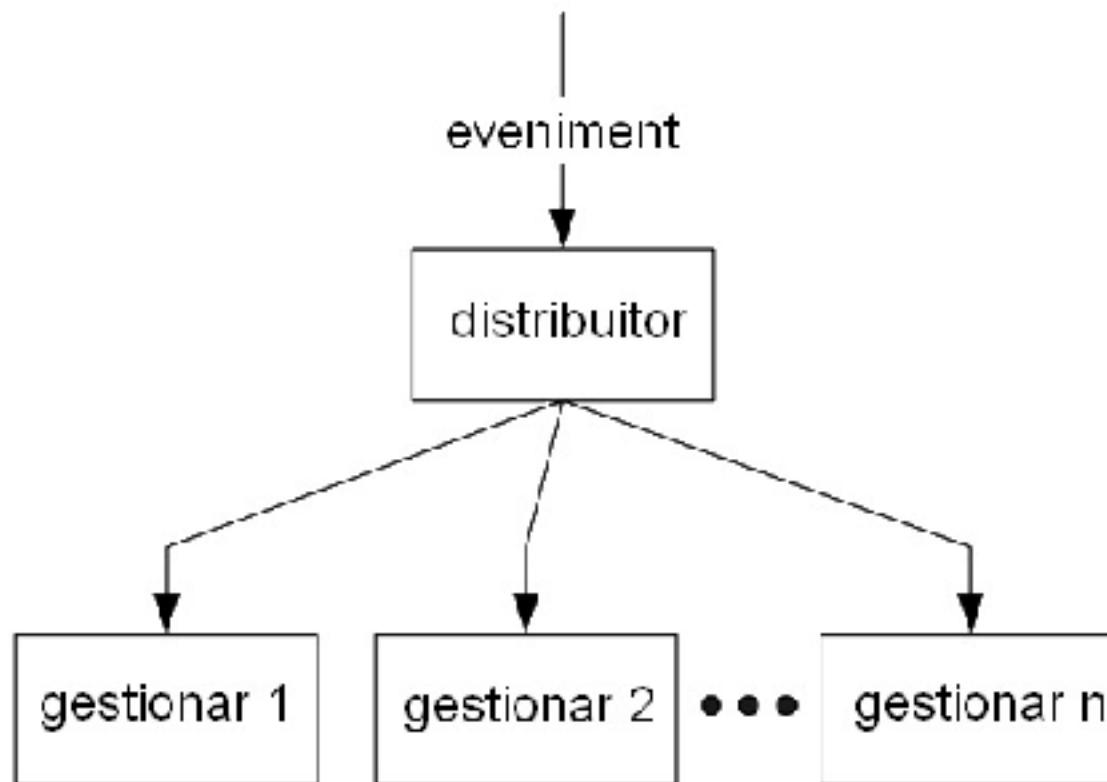
1. Generatoare de evenimente
2. Sursa evenimentelor
3. Bucla de evenimente
4. Gestionari de evenimente
5. Event mapper
6. Înregistrarea evenimentelor

Diagrama de secvență pentru EDP



Tratarea evenimentelor

Dispatcher/Distribuitor/Mapper:



Programarea orientată pe eveniment

EDP – aplicații consolă

```
#include <iostream>
using namespace std;
int value; // globală
int main() { // Inițializare
    char s = '+';
    value = 0;
    while(1) { // buclă tratare eveniment
        cout << "Selectați operația (+,-,q): \n";
        cin >> s; // aștept evenimentul de la utilizator
        switch(s)
        { //event mapper
            case '+': //event registration
                add(); break; //event handler
            case '-': //înregistrare eveniment
                sub(); break; //gestionare eveniment
            case 'q': //înregistrare eveniment
                exit(1); //gestionare eveniment
        }
    }
    return(1);
}
```

EDP – aplicații consolă

```
// gestionare pentru evenimente
void add ()
{ // gestionar eveniment "+"
int in;
cout << "Introduceți un întreg: \n";
cin >> in;
value += in;
cout << "Valoarea curentă este: " << value << "\n";
}
void sub ()
{ // "-" event handler
int in;
cout << "Introduceți un întreg: \n";
cin >> in;
value -= in;
cout << "Valoarea curentă este: " << value << "\n";
}
```

EDP – aplicații GUI

	Secvențial	EDP
Text	puțin	modest
GUI	inutil	majoritar

Proiectarea unei aplicații EDP - GUI

- Stabilirea interacțiunii vizuale și a evenimentelor: **Look & Feel**

GUI proiectată corect

- O interfață este bună dacă are următoarele caracteristici:
 - **Eleganță**
 - Îl ghidează pe looser
 - Oferă **informații ajutătoare**
 - Folosește o **ierarhie** de interfețe
 - Permite ca utilizatorul să facă **greșeli**

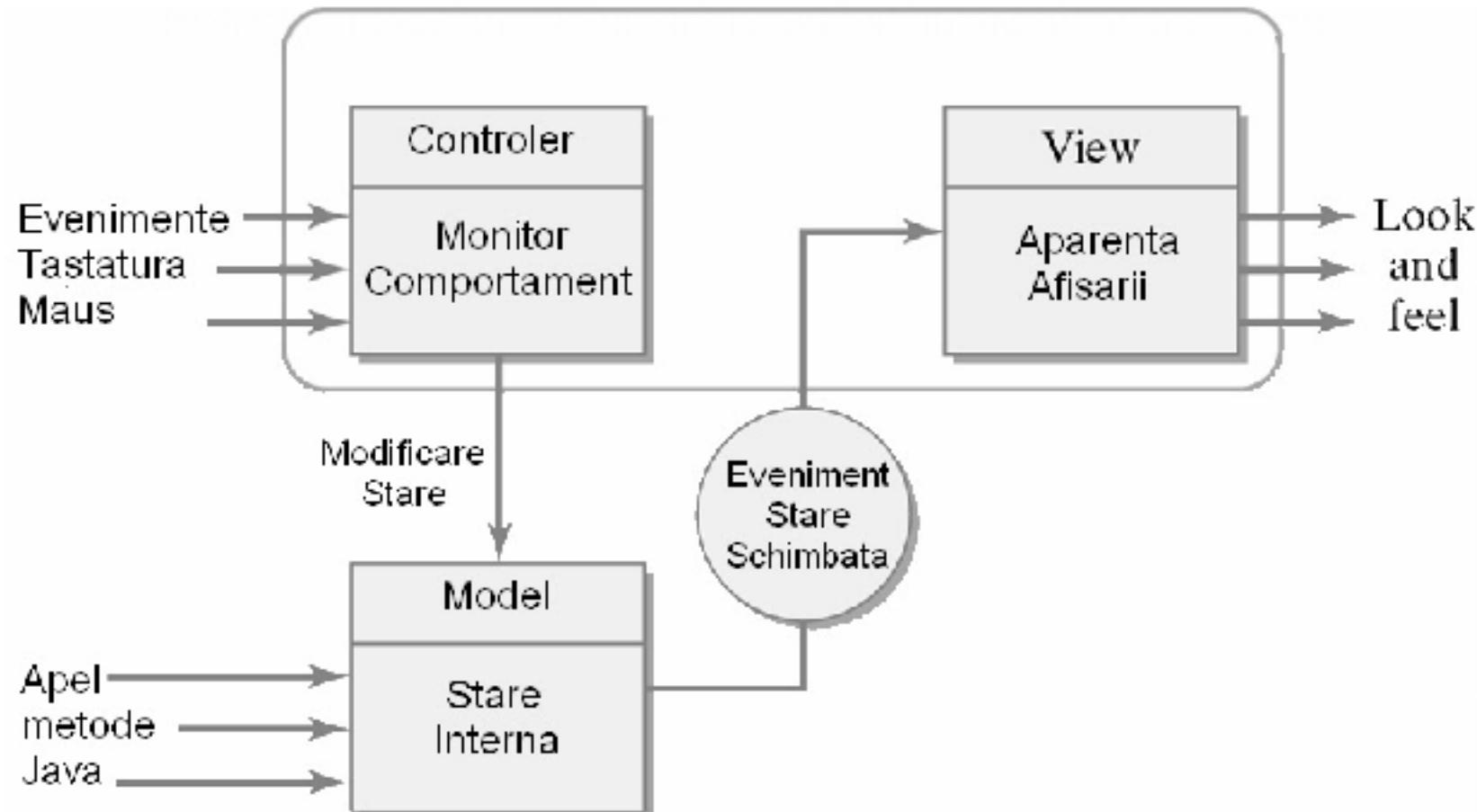
Pick Correlation

- Procesul de selecție a unei ferestre sau aplicații care trebuie să trateze un eveniment oarecare (deoarece le aparține) se numește corelația de selecție (pick correlation)

Ce sunt widgets?

- Sunt obiectele din cadrul unui GUI orientat obiect.

Model-View-Controller (MVC)



Ce este SDL?

- SDL = Simple DirectMedia Layer
- unde = www.libsdl.org
- alte surse
 - <http://tlahoda.github.io/sdlpp/index.html>
 - <https://wiki.libsdl.org/>
 - <https://docs.sdl.com/LiveContent/content/en-US/SDL%20Web-v5>

Componente SDL

Componentă	Descriere	Echivalent DirectX
Video	Ascunde accesul nativ la ecran	DirectDraw
Gestiunea evenimentelor	Ascunde accesul nativ la evenimente	DirectInput
Joystick	Ascunde accesul nativ la maneta de joc	DirectInput
Audio	Ascunde accesul nativ la placa audio	DirectSound
CD-ROM	Accesează direct CD audio	Fără Echivalent (F/E)
Fire de execuție	Funcții helper peste cele native de gestiune a firelor de execuție	F/E
Timere	Funcții helper peste cele native de gestiune a timerelor	F/E

Funcții pentru inițializare	Componenta care va fi inițializată
SDL_Init	inițializează una sau mai multe subsisteme SDL
SDL_InitSubSystem	inițializează numai un subsistem anume. Poate fi utilizată numai după anterioara.
SDL_Quit	închide toate subsistemele SDL
SDL_Quit_SubSystem	închide numai un subsistem anume
SDL_WasInit	verifică și anunță care subsisteme sunt active
SDL_GetError	raportează ultima eroare internă generată de SDL
Identifier	Componenta care va fi inițializată
SDL_INIT_TIMER	acces la timere
SDL_INIT_AUDIO	acces la audio
SDL_INIT_VIDEO	acces la video
SDL_INIT_CDROM	acces la CDROM
SDL_INIT_JOYSTICK	acces la Joystick
SDL_INIT_EVERYTHING	activează întreg framework-ul

SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO); //exemplu de utilizare

Crearea unei ferestre

```
#include<SDL.h>
SDL_Window* g_pWindow = 0;
SDL_Renderer* g_pRenderer = 0;
int main(int argc, char* args[])
{
    if(SDL_Init(SDL_INIT_EVERYTHING) >= 0)
        { SDL_SetVideoMode(800, 600, 32, SDL_FULLSCREEN);
            g_pWindow = SDL_CreateWindow("Testing", SDL_WINDOWPOS_CENTERED,
                                         SDL_WINDOWPOS_CENTERED, 640, 480, SDL_WINDOW_SHOWN);
            if(g_pWindow != 0)
                { g_pRenderer = SDL_CreateRenderer(g_pWindow, -1, 0); }
        }
    else
        { return 1; // eroare }
    SDL_SetRenderDrawColor(g_pRenderer, 0, 0, 0, 255);
    SDL_RenderClear(g_pRenderer);
    SDL_RenderPresent(g_pRenderer);
    // astept 5 secunde
    SDL_Delay(5000);
    // iesire în mod corect
    SDL_DestroyWindow(m_pWindow);
    SDL_DestroyRenderer(m_pRenderer);
    SDL_Quit();
    return 0;
}
```

Setări posibile pentru o fereastră SDL

Flag	Scop
SDL_WINDOW_FULLSCREEN	fereastra va ocupa tot ecranul
SDL_WINDOW_OPENGL	Fereastra poate fi folosită cu un context OpenGL
SDL_WINDOW_SHOWN	Fă vizibilă fereastra
SDL_WINDOW_HIDDEN	Ascunde Fereastra
SDL_WINDOW_BORDERLESS	Elimină marginea implicită a ferestrei
SDL_WINDOW_RESIZABLE	Permite redimensionarea ferestrei
SDL_WINDOW_MINIMIZED	Minimizează fereastra
SDL_WINDOW_MAXIMIZED	Maximizează fereastra
SDL_WINDOW_INPUT_GRABBED	Fereastra preia focus-ul
SDL_WINDOW_INPUT_FOCUS	fereastra are focus-ul de intrare
SDL_WINDOW_MOUSE_FOCUS	fereastra are focus de la șobolan
SDL_WINDOW_FOREIGN	fereastra nu a fost creată cu SDL

Gestiunea evenimentelor în SDL

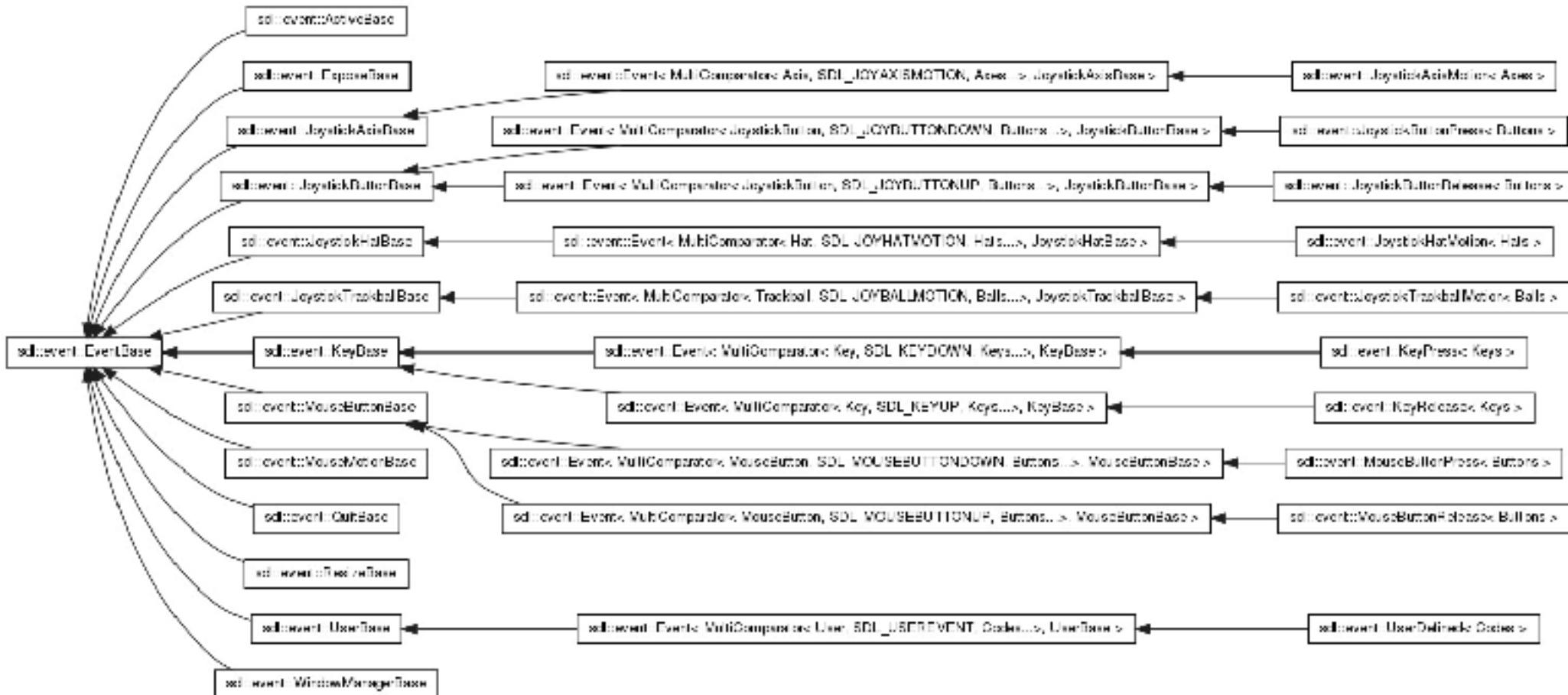
```
#include "SDL.h"
class InputHandler {
public:
    static InputHandler* Instanced
    {
        if(s_pInstance == 0)
            { s_pInstance = new InputHandler(); }
        return s_pInstance;
    }
    void update();
    void clean();
private:
    InputHandler();
    ~InputHandler() {}
    static InputHandler* s_pInstance;
};
typedef InputHandler TheInputHandler;
```

Gestiunea şobolanului - evenimente

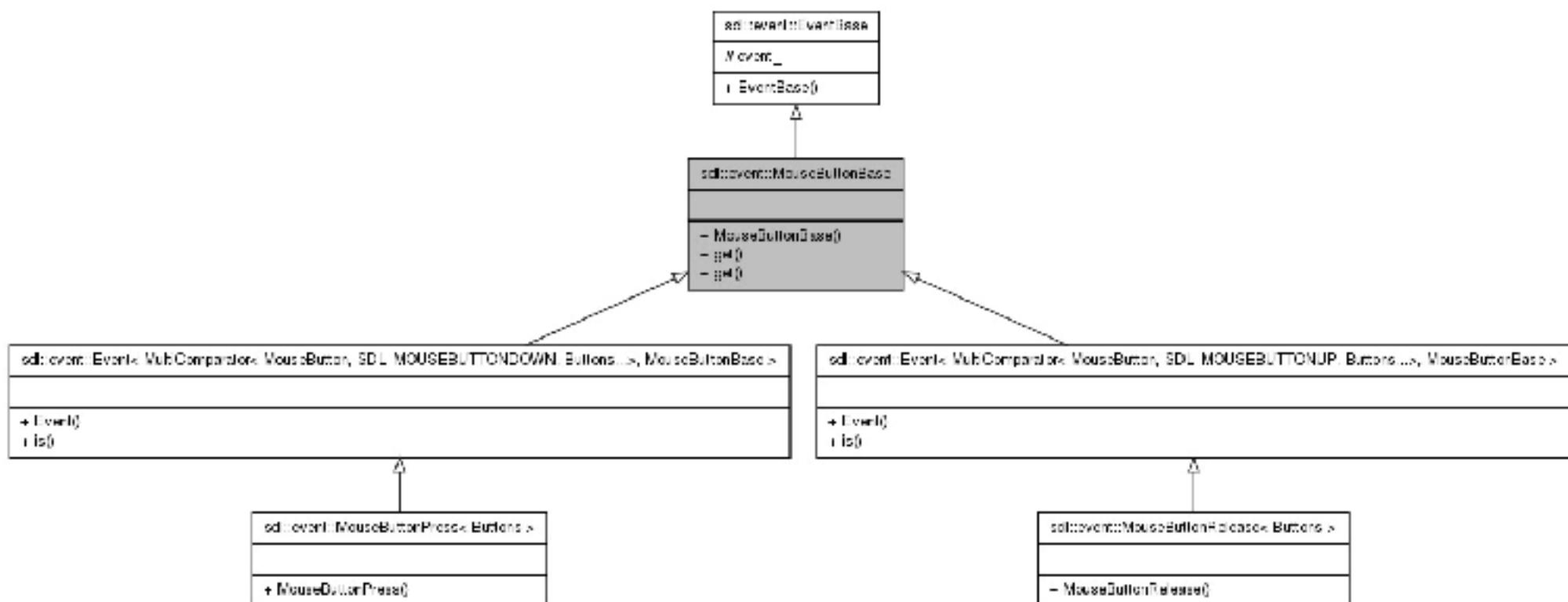
Tip Eveniment SDL	Scop
SDL_MouseButtonEvent	un buton apăsat/elibera
SDL_MouseMotionEvent	s-a mişcat şobolanul
SDL_MouseWheelEvent	a fost mişcată rotiţa guzganului

Tip Eveniment SDL	Valoare posibilă
SDL_MouseButtonEvent	SDL_MOUSEBUTTONDOWN sau SDL_MOUSEBUTTONUP
SDL_MouseMotionEvent	SDL_MOUSEMOTION
SDL_MouseWheelEvent	SDL_MOUSEWHEEL

Eventos sdl



Şoricel - Clase



Gestiunea şobolanului - evenimente

```
std::vector<bool> m_mouseButtonStates;  
for(int i = 0; i < 3; i++)  
    { m_mouseButtonStates.push_back(false); }  
apoi  
enum mouse_buttons {  
LEFT = 0,  
MIDDLE = 1,  
RIGHT = 2  
};
```

Tratare evenimente urechi și obiecte

```
if(event.type == SDL_MOUSEBUTTONDOWN)
{ if(event.button.button == SDL_BUTTON_LEFT)
  { m_mouseButtonStates[LEFT] = true; }
if(event.button.button == SDL_BUTTON_MIDDLE)
  { m_mouseButtonStates[MIDDLE] = true; }
if(event.button.button == SDL_BUTTON_RIGHT)
  { m_mouseButtonStates[RIGHT] = true; } } //și ...
if(event.type == SDL_MOUSEBUTTONUP)
{
if(event.button.button == SDL_BUTTON_LEFT)
  { m_mouseButtonStates[LEFT] = false; }
if(event.button.button == SDL_BUTTON_MIDDLE)
  { m_mouseButtonStates[MIDDLE] = false; }
if(event.button.button == SDL_BUTTON_RIGHT)
  { m_mouseButtonStates[RIGHT] = false; }
} //și...
bool getMouseButtonState(int buttonNumber)
{ return m_mouseButtonStates[buttonNumber]; } //și...
if(TheInputHandler::Instance()->getMouseButtonState(LEFT))
{ //acțiune în program }
```

Tratarea fugii... după brânză

```
Vector2D* m_mousePosition;//apoi  
Vector2D* getMousePosition()  
{ return m_mousePosition; } //acum ...  
if(event.type == SDL_MOUSEMOTION)  
{  
    m_mousePosition->setX(event.motion.x);  
    m_mousePosition->setY(event.motion.y);  
} //și  
Vector2D* vec = TheInputHandler::Instance()->getMousePosition();  
m_velocity = (*vec - m_position) / 100; // un calcul de viteză
```

Tratarea tastaturii

```
SDL_GetKeyboardState(int* numkeys) //și
Uint8* m_keystate; //iar apoi
m_keystates = SDL_GetKeyboardState(0); //ne mai...
bool InputHandler::isKeyDown(SDL_ScanCode key)
{
    if(m_keystates != 0)
    {
        if(m_keystates[key] == 1)
            { return true; }
        else
            {return false; }
    }
    return false;
}// si...

if(TheInputHandler::Instance()->isKeyDown(SDL_SCANCODE_RIGHT))
{ m_velocity.setX(2); }
```

Exemplu combinat

```
cvoid InputHandler::update()
{
    SDL_Event event; while(SDL_PollEvent(&event))
    {switch (event.type)
    { case SDL_QUIT: TheGame::Instance()->quit(); break;
        case SDL_MOUSEMOTION: onMouseMove(event); break;
        case SDL_MOUSEBUTTONDOWN: onMouseButtonDown(event); break;
        case SDL_MOUSEBUTTONUP: onMouseButtonUp(event); break;
        case SDL_KEYDOWN: onKeyDown(); break;
        case SDL_KEYUP: onKeyUp(); break;
        default: break; } }
} //si ...
void InputHandler::onMouseButtonDown(SDL_Event& event)
{ if(event.button.button == SDL_BUTTON_LEFT)
    { m_mouseButtonStates[LEFT] = true; }
    if(event.button.button == SDL_BUTTON_MIDDLE)
    { m_mouseButtonStates[MIDDLE] = true; }
    if(event.button.button == SDL_BUTTON_RIGHT)
    { m_mouseButtonStates[RIGHT] = true; } }
```

Grafică în Python - TKInter

Primul Buton

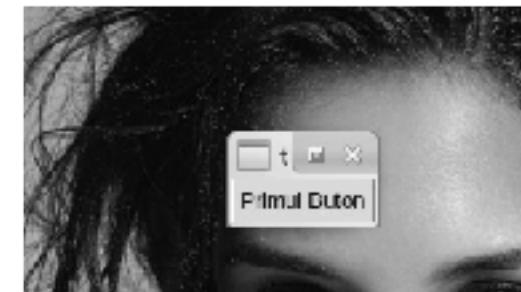
```
from tkinter import *
from tkinter import ttk
baza = Tk()
ttk.Button(baza, text="Primul Buton").grid()
baza.mainloop()
```

Pt disperati

instalați pycharm exact ca și intelij
apoi din terminal

sudo apt-get install python3-tk

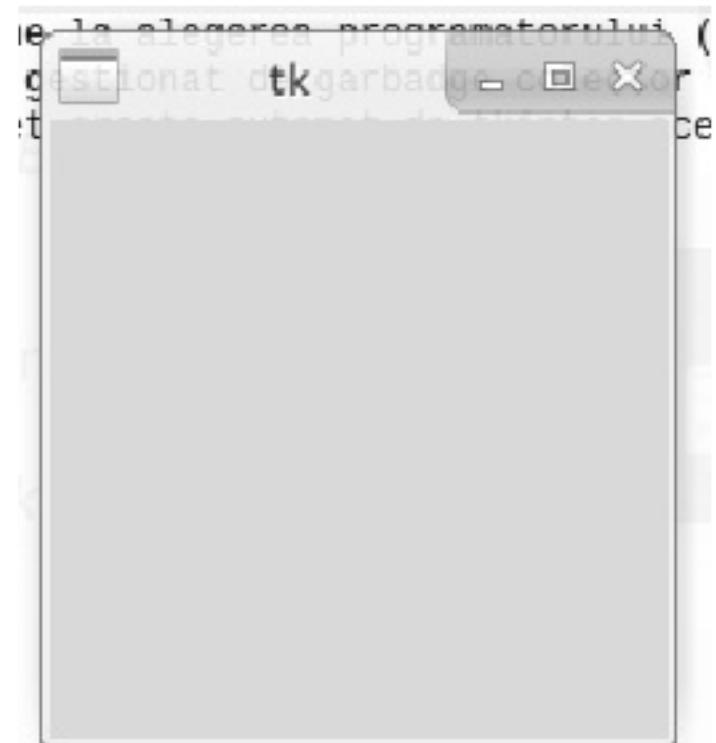
repornește pycharm
execuția și restul ca la intelij
rezultatul --> pe fruntea fetei



Primul Widget

```
from tkinter import *
from tkinter import ttk
```

```
root = Tk()
content = ttk.Frame(root)
button = ttk.Button(content)
root.mainloop()
```



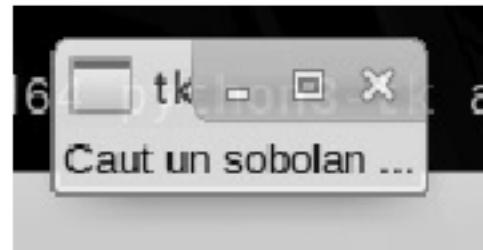
Asocierea unor evenimente

```
from tkinter import *
from tkinter import ttk

root = Tk()

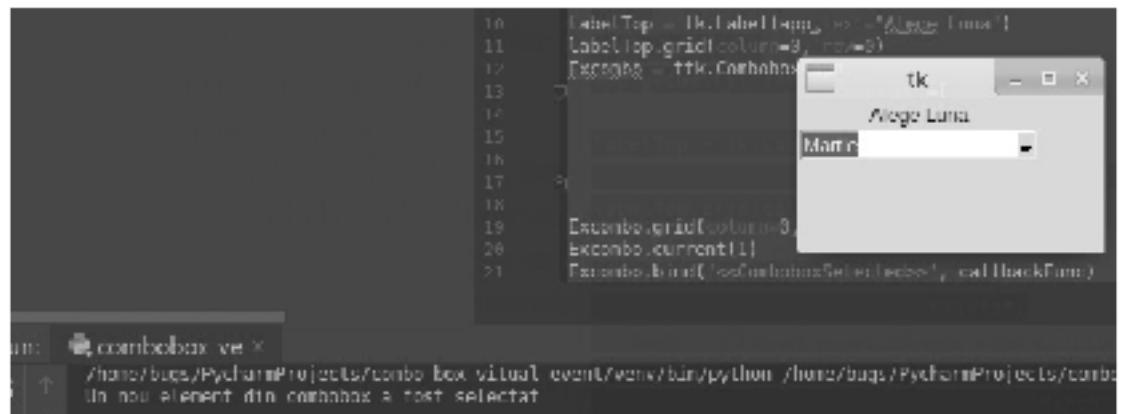
l =ttk.Label(root, text="Caut un sobolan ...")
l.grid()

l.bind('<Enter>', lambda e: l.configure(text='Sobolanul este in interior'))
l.bind('<Leave>', lambda e: l.configure(text='Sobolanul a fugit din zona'))
l.bind('<1>', lambda e: l.configure(text='Sobolanul a miscat din urechea stanga'))
l.bind('<Double-1>', lambda e: l.configure(text='L-am tras de doua ori la rand de
urechi pe sobolan'))
l.bind('<B3-Motion>', lambda e: l.configure(text='Urechea dreapta a fost folosita
pentru o incercare de mutare la %d,%d' %(e.x, e.y)))
root.mainloop()
```



Evenimente virtuale

```
import tkinter as tk
from tkinter import ttk
def callbackFunc(event):
    print("Un nou element din combobox a fost selectat")
app = tk.Tk()
app.geometry('200x100')
labelTop = tk.Label(app, text="Alege Luna")
labelTop.grid(column=0, row=0)
Excombo = ttk.Combobox(app,
                      values=[
                          "Ianuarie",
                          "Februarie",
                          "Martie",
                          "Aprilie"])
Excombo.grid(column=0, row=1)
Excombo.current(1)
Excombo.bind("<<ComboboxSelected>>", callbackFunc)
app.mainloop()
```



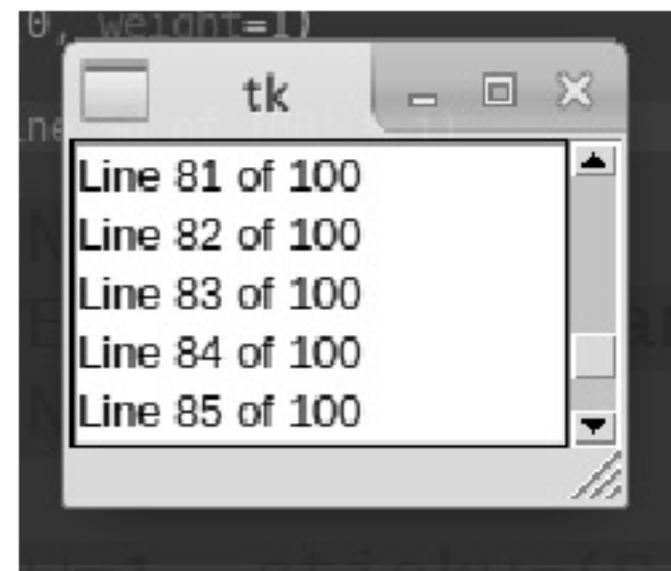
Organizarea matricială a widgets

```
from tkinter import *
from tkinter import ttk
root = Tk()
content = ttk.Frame(root)
frame = ttk.Frame(content, borderwidth=5, relief="sunken", width=200, height=100)
namelbl = ttk.Label(content, text="Name")
name = ttk.Entry(content)
onevar = BooleanVar()
twovar = BooleanVar()
threevar = BooleanVar()
onevar.set(True)
twovar.set(False)
threevar.set(True)
one = ttk.Checkbutton(content, text="One", variable=onevar, onvalue=True)
two = ttk.Checkbutton(content, text="Two", variable=twovar, onvalue=True)
three = ttk.Checkbutton(content, text="Three", variable=threevar, onvalue=True)
ok = ttk.Button(content, text="Okay")
cancel = ttk.Button(content, text="Cancel")
content.grid(column=0, row=0)
frame.grid(column=0, row=0, columnspan=3, rowspan=2)
namelbl.grid(column=3, row=0, columnspan=2)
name.grid(column=3, row=1, columnspan=2)
one.grid(column=0, row=3)
two.grid(column=1, row=3)
three.grid(column=2, row=3)
ok.grid(column=3, row=3)
cancel.grid(column=4, row=3)
root.mainloop()
```



O bară pentru derulare

```
from tkinter import *
from tkinter import ttk
root = Tk()
l = Listbox(root, height=5)
l.grid(column=0, row=0, sticky=(N,W,E,S))
s = ttk.Scrollbar(root, orient=VERTICAL, command=l.yview)
s.grid(column=1, row=0, sticky=(N,S))
l['yscrollcommand'] = s.set
ttk.Sizegrip().grid(column=1, row=1, sticky=(S,E))
root.grid_columnconfigure(0, weight=1)
root.grid_rowconfigure(0, weight=1)
for i in range(1,101):
    l.insert('end', 'Line %d of 100' % i)
root.mainloop()
```



Creare Menu-uri simple cu comenzi directe

```
import tkinter as tk  
  
win = tk.Tk()  
  
win.geometry('400x300')  
win.title("Ceva cu menu")  
menu = tk.Menu(win)  
  
menu.add_command(label='Reintoarcere la dimensiunea normală',  
command=lambda: win.geometry('400x300') + win.title("Dimensiune  
400x300"))  
  
menu.add_command(label='Maresc Fereastra', command=lambda:  
win.geometry('600x600') + win.title("Dimensiune 600x600"))  
  
win.configure(menu=menu)  
win.mainloop()
```



Creare Menu-uri ierarhizate

```
from tkinter import *
def donothing():
    filewin = Toplevel(root)
    button = Button(filewin, text="Do nothing button")
    button.pack()
root = Tk()
menubar = Menu(root)
filemenu = Menu(menubar, tearoff=0)
filemenu.add_command(label="New", command=donothing)
filemenu.add_command(label="Open", command=donothing)
filemenu.add_separator()
filemenu.add_command(label="Exit", command=root.quit)
menubar.add_cascade(label="File", menu=filemenu) ##
editmenu = Menu(menubar, tearoff=0)
editmenu.add_command(label="Undo", command=donothing)
editmenu.add_separator()
editmenu.add_command(label="Cut", command=donothing)
editmenu.add_command(label="Copy", command=donothing)
menubar.add_cascade(label="Edit", menu=editmenu) ##
helpmenu = Menu(menubar, tearoff=0)
helpmenu.add_command(label="Help Index", command=donothing)
helpmenu.add_command(label="About...", command=donothing)
menubar.add_cascade(label="Help", menu=helpmenu)##
root.config(menu=menubar)
root.mainloop()
```

Creare Ferestre cu Ferestre

```
from tkinter import *
m1 = PanedWindow()
m1.pack(fill = BOTH, expand = 1)

left = Entry(m1, bd = 5)
m1.add(left)

m2 = PanedWindow(m1, orient = VERTICAL)
m1.add(m2)

top = Scale( m2, orient = HORIZONTAL)
m2.add(top)

bottom = Button(m2, text = "OK")
m2.add(bottom)

mainloop()
```



Eveniment la Frame

```
from tkinter import Tk, Label, Button, StringVar
from tkinter import ttk
import tkinter as tk
class UnGUI:
    LABEL_TEXT = [
        "Primul GUI!", "Dzeu cu mila.",
        "chiar merge?...", "...eticheta asta interactiva.",
        "apasa poate merge.", ]
    def __init__(self, master):
        self.master = master
        master.title("Ceva simplu")
        self.label_index = 0
        self.label_text = StringVar()
        self.label_text.set(self.LABEL_TEXT[self.label_index])
        self.label = Label(master, textvariable=self.label_text)
        self.label.bind("<Button-1>", self.cycle_label_text)
        self.label.pack()

        self.greet_button = Button(master, text="Fa ceva",
        command=self.validateButton)

        self.greet_button.pack()
```

```
        self.close_button = Button(master, text="Close",
        command=master.quit)
        self.close_button.pack()
    def cycle_label_text(self, event):
        self.label_index += 1
        self.label_index %= len(self.LABEL_TEXT)
        self.label_text.set(self.LABEL_TEXT[self.label_index])
    def validateButton(self):
        win = tk.Toplevel()
        win.wm_title('Window')
        l = tk.Label(win, text="Apasa-l")
        l.grid(row=0, column=0)
        b = ttk.Button(win, text="Am inteles!",
        command=win.destroy)
        b.grid(row=1, column=0)
    root = tk.Tk()
    my_gui = UnGUI(root)
    root.mainloop()
```

Un exemplu pentru disperați

```
from tkinter import *
```

```
from tkinter import messagebox
```

```
top = Tk()
```

```
top.geometry("300x100")
```

```
def hello():
```

```
    messagebox.showinfo("Zii", "Zii Ceva")
```

```
B1 = Button(top, text = "Zii Inca Ceva", command = hello)
```

```
B1.place(x = 35,y = 50)
```

```
top.mainloop()
```

O altă versiune cu butoane

```
import tkinter as tk
app = tk.Tk()
labelExample = tk.Button(app, text="0")

def change_label_number():
    counter = int(str(labelExample['text']))
    counter += 1
    labelExample.config(text=str(counter))
buttonExample = tk.Button(app, text="Increase", width=30,
command=change_label_number)
buttonExample.pack()
labelExample.pack()
app.mainloop()
```

UN desen simplu

```
from tkinter import *
```

```
top = Tk()
```

```
C = Canvas(top, bg = "blue", height = 250, width = 300)
```

```
coord = 10, 50, 240, 210
```

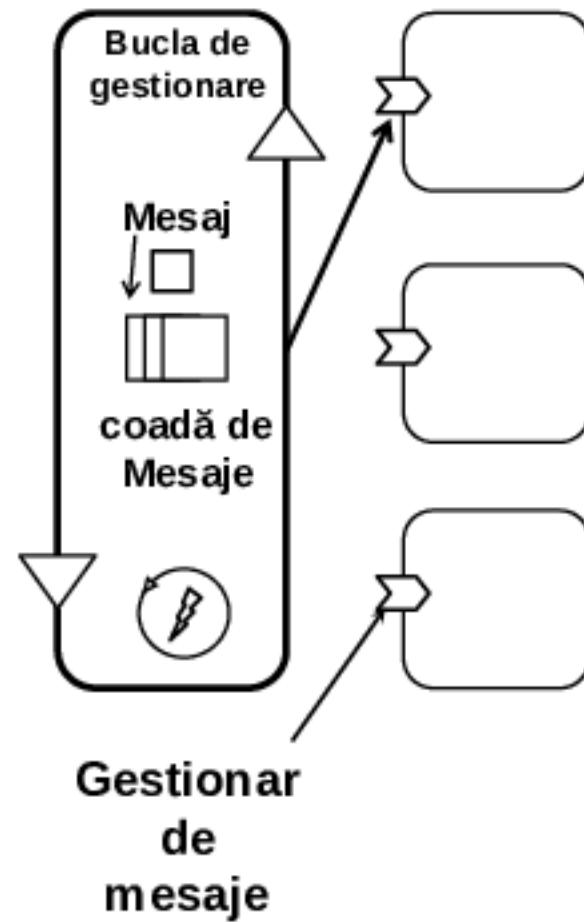
```
arc = C.create_arc(coord, start = 0, extent = 150, fill = "red")
```

```
line = C.create_line(10,10,200,200,fill = 'white')
```

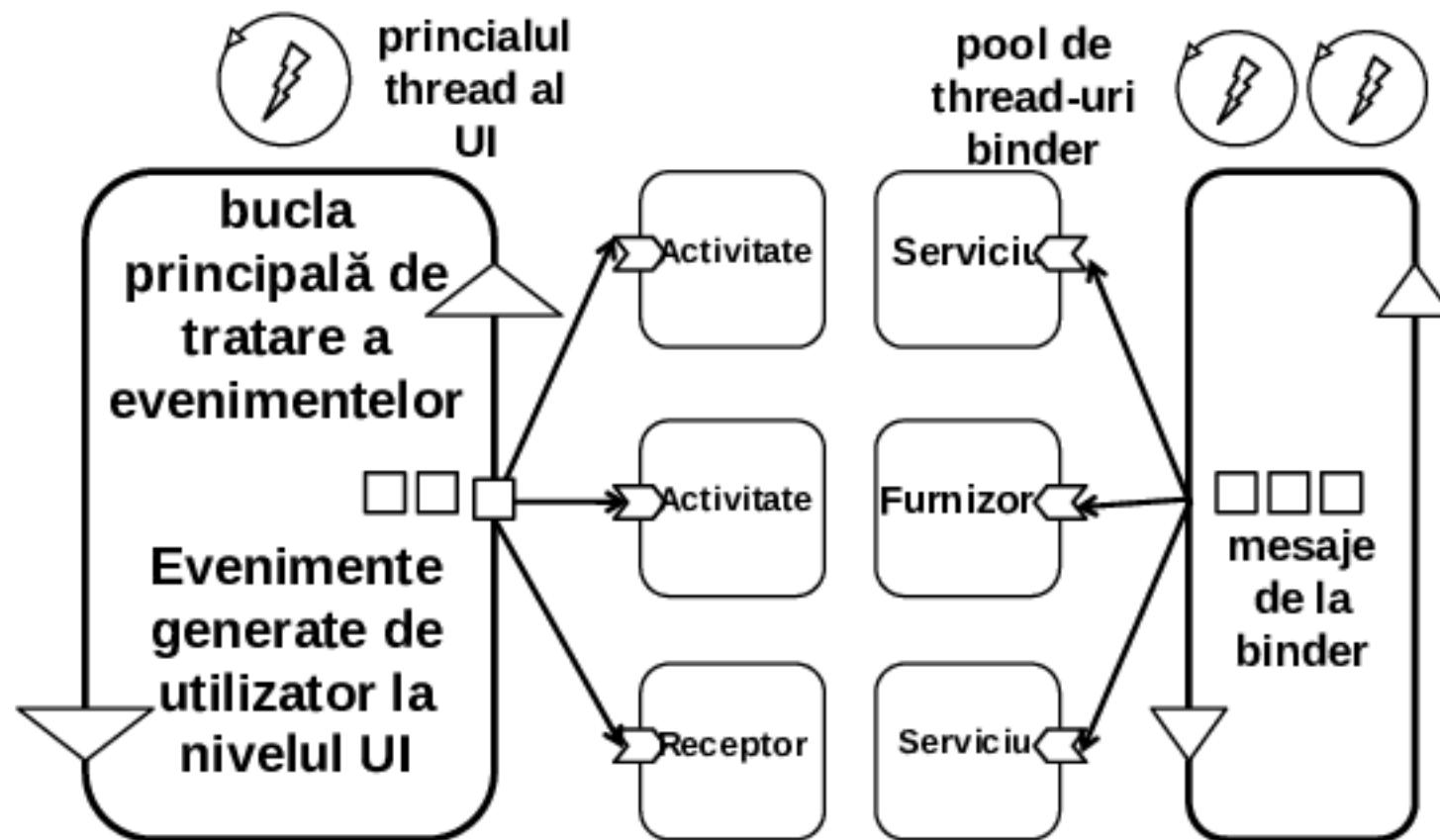
```
C.pack()
```

```
top.mainloop()
```

Clase eveniment pentru Android



Adăugare servicii în Android (versiunea pentru disperați)



API pentru gestiunea ideală a evenimentelor

Gestionarul (Poll)

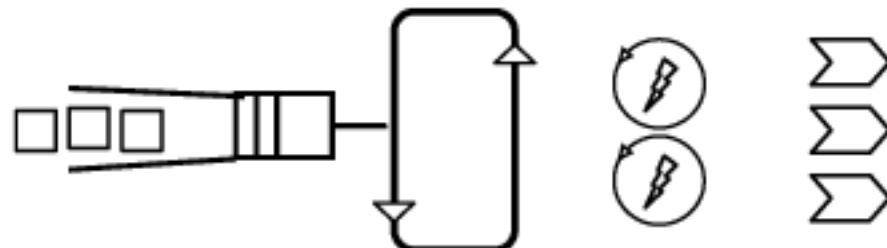
1. Livrează:

2. În pauze:

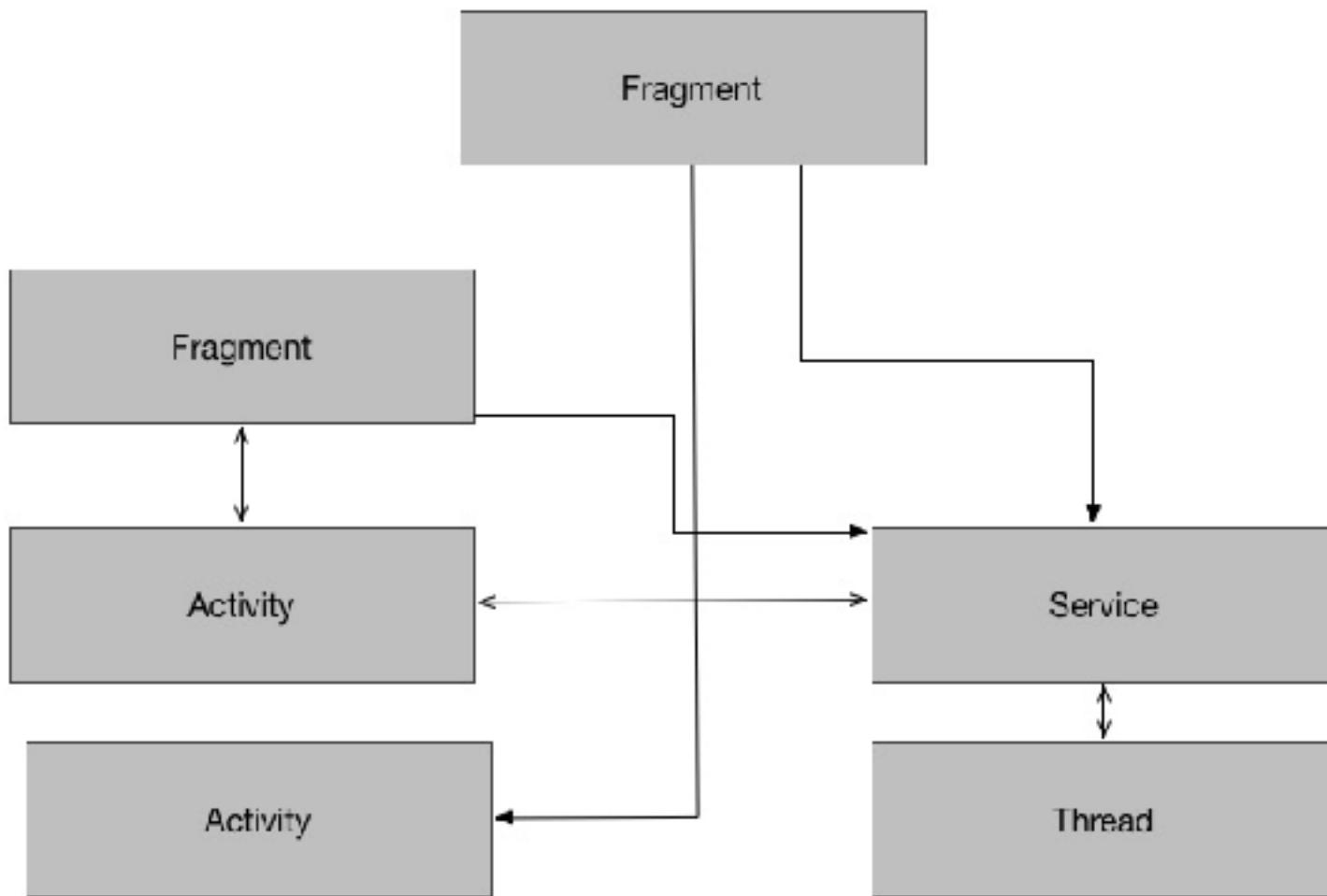
3. Consumă:

4. Combină:

5. Synchronizare:

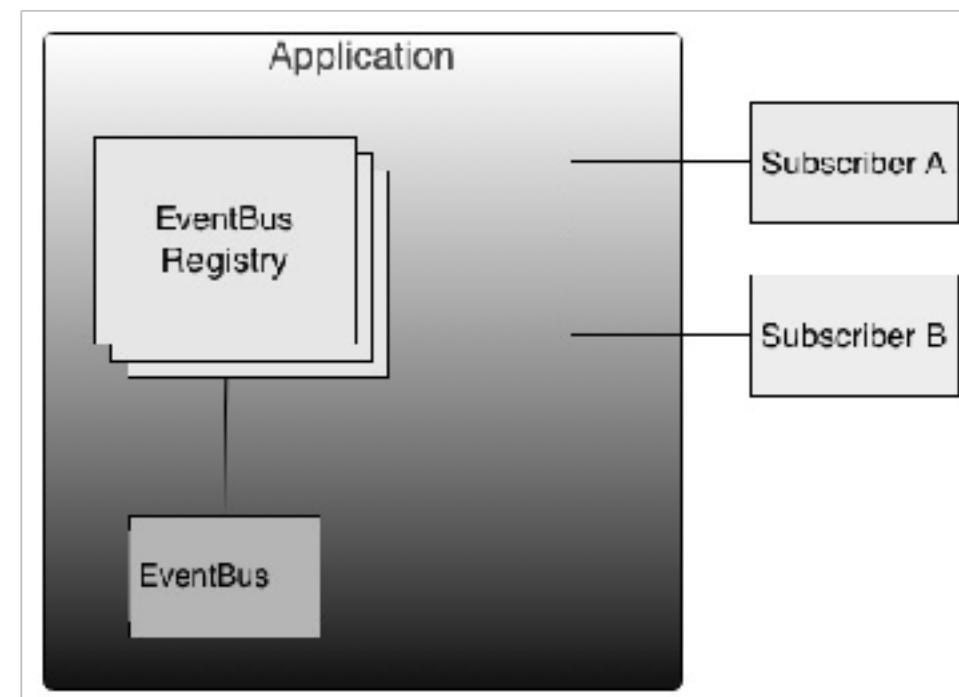
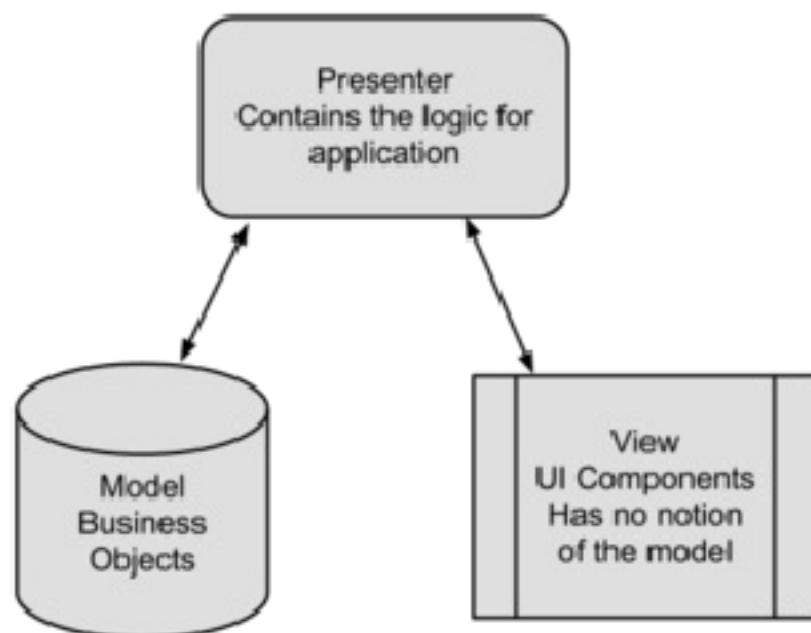


O posibilă manieră de abordare a proiectării unei aplicații



Ce se poate face?

- Să utilizăm Model View Presenter!



Paradigma Modulară

Cursul nr. 6
Mihai Zaharia

Ce se poate face cu Python?

ORICE:

- aplicații WEB
- frontend (inclusiv pt cloud)
- backend
- hardware low level (suportat nativ pe unele microcontrolere)
- arhitecturi complexe bazate pe microservicii
- inteligență artificială
- securitate (pentest (mai mult automatizare), criptografie dar NU treburi serioase)

Paradigme de programare suportate de PyThon

- Funcțională

```
import functools
my_list = [1, 2, 3, 4, 5]
def add_it(x, y):
    return (x + y)
sum = functools.reduce(add_it,
my_list)
print(sum)
```

- Orientată obiect

```
class ChangeList(object):
    def __init__(self, any_list):
        self.any_list = any_list
    def do_add(self):
        self.sum = sum(self.any_list)
create_sum = ChangeList(my_list)
create_sum.do_add()
print(create_sum.sum)
```

- Imperativă

```
sum = 0
for x in my_list:
    sum += x
print(sum)
```

- Procedurală

```
def do_add(any_list):
    sum = 0
    for x in any_list:
        sum += x
    return sum
print(do_add(my_list))
```

Structurarea unui program

Sintaxa

Instrucțiuni multi-linie

Exemplu 1:

```
total = item1 + \
        item2 + \
        item3
```

Exemplu 2:

```
ZileLucrat = ['Luni','Marti','Miercuri',
               'Joi','Vineri']
```

Cuvinte cheie

And	exec	not	assert	finally	break or continue
For	pass	class	from	print	del
global	raise	def	if	return	else
Import	try	elif	in	while	
Is	with	except	lambda	yield	

Conversii explicite ale tipurilor de date

Sunt suportate: *integer*, *floating point*, *long* și *complex integer* (valoare), *long(valoare)*, *complex(real,imaginar)* cu caz particular *limaginare = 0 complex(real)*

Câteva funcții matematice (suportă mult mai multe)

sqrt(val), *pow(baza,exp)*, minorare la întreg - *ceil(val)*, majorare la întreg - *floor(val)*, *log(val)*, *min(val1,val2,val3,...)* și *max(val1,val2,val3,...)*

și trigonometrice:

acos, *asin*, *atan*, *atan2*, *hypot*, *cos*, *sin*, *tan*, *degrees*, *radian*

și constante precum *pi* și *e*

Control execuției

Decizie unică (simplă)

```
if <condition>:  
    <body>
```

Decizie binară/duală

```
if <condition>:  
    <statements>  
else:  
    <statements>
```

În calculul rădăcinii ecuației de gradul II

Calculează discriminantul (Δ)

when < 0 : tratează cazul fără rădăcini reale

when $= 0$: tratează cazul particular al unui singur rezultat

when > 0 : tratează cazul general al celor două soluții

Decizie Multiplă

```
if <condition1>:  
    <case1 statements>  
elif <condition2>:  
    <case2 statements>  
elif <condition3>:  
    <case3 statements>  
...  
else:  
    <instructiuni pt default>
```

Similară cu *switch-case-default*

Excepții

```
import math

print("Rezolvare ecuație de grad II\n")
try:
    a, b, c = [int(x) for x in input("Introdu coeficienții a b c: ").split()]
    discRoot = math.sqrt(b * b - 4 * a * c)
    root1 = (-b + discRoot) / (2 * a)
    root2 = (-b - discRoot) / (2 * a)
    print("\nSoluțiile sunt:", root1, root2)
except ValueError as excObj:
    if str(excObj) == "math domain error":
        print("Nu are rădăcini reale")
    else:
        print("Număr încorect de valori")
except NameError:
    print("\nNu ai introdus trei valori")
except TypeError:
    print("\nNu toate valorile introduse sunt numere")
except SyntaxError:
    print("\nFormatul datelor încorect - utilizati spatiu")
except:
    print("\nEroare necunoscută")
```

Instrucțiunea try:

```
try:  
    <body>  
except <ErrorType>:  
    <handler>
```

Bucle cu număr de pași cunoscut - For

```
for <var> in <sequence>:  
    <body>  
  
for i in range(11):  
    print(i)
```

Bucle cu număr de pași necunoscut - While

```
while <condition>:  
    <body>  
  
i = 0  
while i <= 10:  
    print(i)  
    i = i + 1
```

Instrucțiunea pass

- **Exemplu:**

```
for litera in 'Python':  
    if litera == "h":  
        pass  
        print("Acesta este efectul lui pass")  
    print('litera curenta:', litera)  
print( 'Pa Pa!')
```

Rezultate:

```
litera curenta: P  
litera curenta: y  
litera curenta: t  
Acesta este efectul lui pass  
litera curenta: h  
litera curenta: o  
litera curenta: n  
Pa Pa!
```

Operatori

Ordinea de precedență este *not, and, or* deci se consideră că
 $a \text{ or } \text{not } b \text{ and } c \Leftrightarrow (a \text{ or } ((\text{not } b) \text{ and } c))$

Totuși parantezele rămân recomandate

$(a \geq 15 \text{ and } a - b \geq 2) \text{ or } (b \geq 15 \text{ and } b - a \geq 2)$

$(a \geq 15 \text{ or } b \geq 15) \text{ and } \text{abs}(a - b) \geq 2$

Datele sunt evaluate boolean ca în C (true este $\neq 0$)

Operator	Traducere în operații logice
$x \text{ and } y$	If x is false, return x . Otherwise, return y .
$x \text{ or } y$	If x is true, return x . Otherwise, return y .
$\text{not } x$	If x is false, return True. Otherwise, return False.

Lista operatori: +, -, *, /, //, **, %, abs,

Cu observația că: $10//3 = 3$ și $10\%3 = 1$ adică $a = (a/b)(b) + (a\%b)$

Operatori de atribuire

Operator	Descriere	Exemplu
= atribuire simplă		$c = a + b$
$+ =$ operandului din stânga i se adună operandul din dr.		$c += a \rightarrow c = c + a$)
$- =$ din operandul stâng se scade operandul drept		$c -= a \rightarrow (c = c - a)$)
$* =$ operandul din stânga se multiplică cu operandul din dr.		$c *= a \rightarrow (c = c * a)$)
$/ =$ împarte operandul din stânga cu cel din dreapta		$c /= a \rightarrow (c = c / a)$)
$\% =$ operandului din dreapta i se atribuie restul împărțirii		$c \% a \rightarrow (c = c \% a)$)
$** =$ operandului din stânga i se atribuie puterea		$c **= a \rightarrow (c = c ** a)$)
$// =$ împărțirea întreagă este atribuită operandului stâng		$c // a \rightarrow (c = c // a)$)

Operatori pe biți

a = 0011 1100

b = 0000 1101

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

Operatori Membership (apartenență)

Operator	Descriere	Exemplu
in	verifică dacă un obiect este într-o secvență	x in y
not in	verifică dacă un obiect nu este în secvență	x not in y

Operatori de identitate

Operator	Descriere	Exemplu
is	verifică dacă 2 variabile au aceeași referință (adresă) de memorie	a is b
not is	verifică dacă 2 variabile nu au aceeași referință (adresă) de memorie	a not is b

Tipuri de date: şiruri de caractere

- ```
>>> ord("A")→65 | >>> ord("a") →97 | >>> chr(97) →'a' | >>> chr(65) →'A'
>>> str1="Hello" | >>> str2='spawn' | >>> print(str1, str2)→Hello
spawn
>>> type(str1)→<class 'str'> | >>> type(str2)→<class 'str'>
 - Citirea unui şir

```
>>> firstName = input("Cum te cheama?: ")
```
 - Forma generală de indexare <string>[<expr>] (ca în C)
```



```
>>> greet[-1] →'b' | >>> greet[-3] →'B'
```

# Tipuri de date:șiruri de caractere

- *Extragere subșiruri* (slicing)

|   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|
| H | e | l | l | o |   | B | o | b |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

```
>>> greet[0:3]→'Hel' | >>> greet[5:9]→' Bob' | >>> greet[:5]→'Hello' |
>>> greet[5:]→' Bob' | >>> greet[:]→'Hello Bob'
```

- *Concatenare (+) Repetīție (\*) Lungimea (nr caractere) len.*

```
>>> "spam" + "eggs"→'spameggs' | >>> "Spam" + "And"+ "Eggs"→'SpamAndEggs' |
>>> 3 * "spam"→'spamspamspam' | >>> "spam" * 5→'spamspamspamspamspam' |
>>> (3 * "spam") + ("eggs" * 5) →'spamspamspameggseggseggseggseggs'
>>> len("spam") →4
>>> for ch in "Spam!":
 print (ch, end=" ")
```

# Tipuri de date:șiruri de caractere

Alte metode:

- s.capitalize() – creează o copie a lui s cu prima literă făcută mare
- s.title() – creează o copie a lui s cu toate literele făcute mari
- s.center(width) – centrează pe s într-o zonă de o lățime dată
- s.count(sub) – numără aparițiile lui sub în s
- s.find(sub) – caută prima apariție a lui sub în s
- s.join(list) – concatenează o listă de şiruri având s ca separator
- s.ljust(width) – ca și center dar în stânga
- s.lower() – creează o copie a lui s cu toate literele făcute mici

# Tipuri de date:șiruri de caractere

- s.lstrip() – creează o copie a lui s cu toate spațiile albe eliminate
- s.replace(oldsub, newsub) – înlocuiește aparițiile în s a lui oldsub cu newsub
- s.rfind(sub) – cauță prima apariție a lui sub în s și întoarce cea mai din dreapta apariție
- s.rjust(width) – are efect ca și center, dar s este identat la dreapta (right-justified)
- s.rstrip() – șterge spațiile de la început și de la sfârșit
- s.split() – extrage o listă de substringuri funcție de un separator implicit ‘ ’ sau explicit oarecare
- s.upper() – creează o copie a lui s cu toate literele făcute mari

# Afișare obiecte - ca și în limbajul C

---

| <b>Caracter</b>    | <b>Argument Așteptat</b>             |
|--------------------|--------------------------------------|
| <b>c</b>           | Șir de lungime 1                     |
| <b>s</b>           | Șir de orice lungime                 |
| <b>d</b>           | Întregi în baza 10                   |
| <b>u</b>           | Întregi fără semn în baza 10         |
| <b>o</b>           | Întregi în baza 8                    |
| <b>x or X</b>      | Întregi în baza 16 (uppercase for X) |
| <b>e, E, f, g,</b> | Reale în diverse stiluri             |
| <b>G</b>           |                                      |
| <b>%</b>           | Procent literal                      |

## **Formatarea afișării - cam ca la C**

```
capitole = {1: 5, 2: 46, 3: 52, 4: 87, 5: 90}
```

```
hexStr = "3f8"
```

```
for x in capitole:
```

```
 print("Capitole " + str(x) + \
 str(capitole[x]).rjust(15, '.'))
```

```
print("Capitol %d %15s" % (x, str(capitole[x])))
```

```
print("\nHex String: " + hexStr.upper().rjust(8, '0')) # Right justify
```

```
print("\nHex String: " + hexStr.upper().ljust(8, '0')) # Left justify
```

```
for x in capitole:
```

```
 print("Capitol %d %15s" % (x, str(capitole[x]))) # formatare
String
```

## Exemplu de formatare - consolă

```
print("Hello {0} {1}, you may have won ${2}" .format("Mr.", "Smith", 10000))
```

```
Hello Mr. Smith, you may have won $10000
```

```
print('This int, {0:5}', was placed in a field of width 5'.format(7)) #cu ieșire
```

```
This int, 7, was placed in a field of width 5
```

```
print('This int, {0:10}', was placed in a field of width 10'.format(10))#cu ieșire
```

```
'This int, 10, was placed in a field of width 10'
```

```
print('This float, {0:10.5}', has width 10 and precision 5.'.format(3.1415926))
```

```
This float, 3.1416, has width 10 and precision 5.
```

```
print('This float, {0:10.5f}', is fixed at 5 decimal places.'.format(3.1415926))
```

```
This float, 3.14159, has width 0 and precision 5.
```

# **Functii simple pentru String**

- capitalize, center, count, expandtabs, find, index, decode, encode, endswith, isalnum, isalpha, isspace, istitle, isupper, join, len, isdigit, islower, isnumeric, ljust

```
Str = 'Iar au inceput sa adoarma!!!'
```

```
suffix = '!!!'
```

```
print (Str.endswith(suffix))
```

```
print (Str.endswith(suffix,20))
```

```
suffix = 'exam'
```

```
print (Str.endswith(suffix))
```

```
print (Str.endswith(suffix, 0, 19))
```

# Executarea de cod din interiorul unui șir

```
exec(str [,globals [,locals]])
radius = 3
cards = ['Ace', 'King', 'Queen', 'Jack']
codeStr = 'for card in cards: print ("Card = " + card)'
exec(codeStr)
eval(str [,globals [,locals]])
areaStr = 'pi*(radius*radius)'
print("\nArea = " + str(eval(areaStr, {'pi': 3.14}, {'radius': 5})))
values = [5, 3, 'blue', 'red']
substitute(m, [, kwargs]).
s = string.Template("Variable v = $v")
for x in values:
 print(s.substitute(v=x))
```

# Liste simple

```
x = [1,2,3]
print("\n"+str(x)) → [1, 2, 3]
y = x
print("\n"+str(y)) [1, 2, 3]
x[1] = 15
print("\n"+str(x)) [1, 15, 3]
x.append(12)
print("\n"+str(x)) [1, 15, 3, 12]
print("\n"+str(y)) [1, 15, 3, 12]
x = [1, 'hello', (3 + 2j)]
print("\n"+str(x)) [1, 'hello', (3+2j)]
print("\n"+str(x[2])) (3+2j)
print("\n"+str(x[0:2])) [1, 'hello']
```

```
x = [1,2,3]
y=x
print("\n"+str(x)) [1, 2, 3]
x = x + [9,10]
print("\n"+str(x)) [1, 2, 3, 9, 10]
print("\n"+str(y)) [1, 2, 3]
```

# Operații pe listă

| Operator                                                                         | Înțelesul acestuia    |
|----------------------------------------------------------------------------------|-----------------------|
| $\langle \text{seq} \rangle + \langle \text{seq} \rangle$                        | Concatenare           |
| $\langle \text{seq} \rangle * \langle \text{int-expr} \rangle$                   | Repetiție             |
| $\langle \text{seq} \rangle []$                                                  | Indexare              |
| $\text{len}(\langle \text{seq} \rangle)$                                         | Lungime               |
| $\langle \text{seq} \rangle [:]$                                                 | Slicing               |
| $\text{for } \langle \text{var} \rangle \text{ in } \langle \text{seq} \rangle:$ | Iterație              |
| $\langle \text{expr} \rangle \text{ in } \langle \text{seq} \rangle$             | Apartenență (Boolean) |

# Operații pe listă

| Metoda                                 | Înțelesul acesteia                                                  |
|----------------------------------------|---------------------------------------------------------------------|
| <code>&lt;list&gt;.append(x)</code>    | Adaugă x la sfârșitul listei                                        |
| <code>&lt;list&gt;.sort()</code>       | Sortează lista (o funcție de sortare poate fi trimisă ca parametru) |
| <code>&lt;list&gt;.reverse()</code>    | Inversează lista                                                    |
| <code>&lt;list&gt;.index(x)</code>     | Indexul pentru prima apariție a lui x în listă                      |
| <code>&lt;list&gt;.insert(i, x)</code> | Inserează x pe poziția i                                            |
| <code>&lt;list&gt;.count(x)</code>     | Numărul de apariții al lui x listă                                  |
| <code>&lt;list&gt;.remove(x)</code>    | Șterge prima apariție a lui x în listă                              |
| <code>&lt;list&gt;.pop(i)</code>       | Șterge elementul i și îi întoarce valoarea                          |

# Exemple liste

```
lst = [3, 1, 4, 1, 5, 9]
lst.append(2)
print("\n"+str(lst)) → [3, 1, 4, 1, 5, 9, 2]
lst.sort()
print("\n"+str(lst)) → [1, 1, 2, 3, 4, 5, 9]
lst.reverse()
print("\n"+str(lst)) → [9, 5, 4, 3, 2, 1, 1]
print("\n"+str(lst.index(4))) → 2
lst.insert(4, "Hello")
print("\n"+str(lst)) → [9, 5, 4, 3, 'Hello', 2, 1, 1]
print("\n"+str(lst.count(1))) → 2
lst.remove(1)
print("\n"+str(lst)) → [9, 5, 4, 3, 'Hello', 2, 1]
lst.pop(3)
print("\n"+str(lst)) → [9, 5, 4, 'Hello', 2, 1]
```

# Tuplele

```
hexStringChars = ('A', 'B', 'C', 'D', 'E', 'F')
hexStringNums = ('1', '2', '3', '4', '5', '6', '7', '8', '9', '0')
hexStrings = ["1FC", "1FG", "222", "Ten"]
for hexString in hexStrings:
 for x in hexString:
 if ((not x in hexStringChars) and
 (not x in hexStringNums)):
 print(hexString+" nu este in hexa")
 break
tupleList = list(hexStringChars)
print("\n"+str(tupleList))
listTuple = tuple(hexStrings)
print("\n"+str(listTuple))
```

și ieșirea programului:

1FG nu este in hexa  
Ten nu este in hexa

[A', 'B', 'C', 'D', 'E', 'F']  
('1FC', '1FG', '222', 'Ten')

# Dicționar

- #dicționar simplu unu la unu

```
numberDict = {1:'one', 2:'two', 3:'three', 4:'four'}
```

```
print("\n", str(numberDict))
```

#dicționar unu la mai mulți

```
letterDict = {'vowel':['a','e','i','o','u'],\
 'consonant':['b','c','d','f']}
```

```
print("\n", str(letterDict))
```

#dicționar mai mulți la mai mulți

```
numbers = (1,2,3,4,5,6,7,8,9,0)
```

```
letters = ('a','b','c','d','e','f')
```

```
punct = ('.', '!', '?')
```

```
charSetDict = {numbers:[], letters:[], punct:[]}
```

```
print("\n", str(charSetDict))
```

și rezultatul execuției

```
{1: 'one', 2: 'two', 3: 'three', 4: 'four'}
```

```
{'vowel': ['a', 'e', 'i', 'o', 'u'], 'consonant': ['b', 'c', 'd', 'f']}
```

```
[(1, 2, 3, 4, 5, 6, 7, 8, 9, 0): [], ('a', 'b', 'c', 'd', 'e', 'f'): [], ('.', '!', '?'): []]
```

# Adăugare de valori

```
numbers = ('1','2','3','4','5','6','7','8','9','0')
letters = ('a','b','c','d','e','f')
punct = ('.', '!', '?')
charSetDict = {numbers:[], letters:[], punct:[]}
cSet = input("Introduceti caractere:")
for c in cSet:
 for x in charSetDict.keys():
 if c in x:
 charSetDict[x].append(c)
 break;
charSetDict["Special"] = ['%', '$', '#']
charSetDict["Special"] = '><'
print("\n"+str(charSetDict))
```

și rezultatul execuției:

Introduceti caractere:12hjh78

{('1', '2', '3', '4', '5', '6', '7', '8', '9', '0'): ['1', '2', '7', '8'], ('a', 'b', 'c', 'd', 'e', 'f'): [], ('.', '!', '?'): [], 'Special': '><}'}

# Adăugare valori la dicționar

```
numbers = ('1','2','3','4','5','6','7','8','9','0')
letters = ('a','b','c','d','e','f')
punct = ('!','!','?')
charSetDict = {numbers:[], letters:[], punct:[]}
def display_cset(cset):
 #print
 for x in cset.items():
 if x[0] == numbers:
 print("Numere:")
 elif x[0] == letters:
 print("Cifre:")
 elif x[0] == punct:
 print("Punctuație:")
 else:
 print("Necunoscut:")
 print(x[1])
cSet = input("Introduceti caractere: ") #se adauga noi valori la chei
for c in cSet:
 for x in charSetDict.keys():
 if c in x:
 charSetDict[x].append(c)
 break;
display_cset(charSetDict)
charSetDict["Special"] = ['%', '$', '#'] #se adaugă o cheie și valoare nouă
display_cset(charSetDict)
charSetDict["Special"] = '><' #se schimbă valoarea unei chei existente
display_cset(charSetDict)
```

Introduceti caractere: 1,d,4d%,<3,4>

Numere:

['1', '4', '3', '4']

Cifre:

['d', 'd']

Punctuație:

[]

Numere:

['1', '4', '3', '4']

Cifre:

['d', 'd']

Punctuație:

[]

Necunoscut:

['%', '\$', '#']

Numere:

['1', '4', '3', '4']

Cifre:

['d', 'd']

Punctuație:

[]

Necunoscut:

><

# Modificare de elemente din dicționar

```
validkeys = (1,2,3)
keyGenDict={'keys': [1,2,3], 1: 'blue',
 2: 'fast', 3: 'test','key': 2}
print("Cheile:")
print(keyGenDict.keys())
print("\nValorile")
print(keyGenDict.values())
print("\nElemente:")
print(keyGenDict.items())
val = keyGenDict['key']
keyGenDict['key'] = 1
print("\nValorile")
print(keyGenDict.values())
keyGenDict.clear()
print("\nElemente:")
print(keyGenDict.items())
```

Cheile:  
dict\_keys(['keys', 1, 2, 3, 'key'])

Valorile  
dict\_values([[1, 2, 3], 'blue', 'fast', 'test', 2])

Elemente:  
dict\_items([('keys', [1, 2, 3]), (1, 'blue'), (2, 'fast'), (3, 'test'), ('key', 2)])

Valorile  
dict\_values([[1, 2, 3], 'blue', 'fast', 'test', 1])

Elemente:  
dict\_items([])

# Extragerea unei valori din dicționar

```
validkeys = (1,2,3)
keyGenDict={'keys':[1,2,3],1:'blue',
2:'fast', 3:'test','key':2}

def show_key (key):
 if(key in validkeys):
 keyVal = (keyGenDict["keys"])[key-1]
 print("Key = " + keyGenDict[keyVal])
 else:
 print("Invalid key")

#lista cheilor din dicționar
print(keyGenDict.keys()) -----> dict_keys(['keys', 1, 2, 3, 'key'])

#valorile dicționar
print(keyGenDict.items()) dict_items([('keys', [1, 2, 3]), (1, 'blue'), (2, 'fast'), (3, 'test'), ('key', 2)])

#valoare din cheie
val = keyGenDict["key"]
show_key(val) Key = fast
 Key = blue
```

# Gestiunea fișierelor în Python

- `open(path [,mode [,buffersize]])`
- `Infile = open("numbers.dat", "r")`
- `<file>.read()`
- `<file>.readline()`
- `<file>.readlines()`

# **Atributele unui obiect fișier**

- Atributele unui obiect fisier:**

După ce s-a deschis un fișier se obține o referință (obiect fișier) care are mai multe atribute.

| <b>• Atribut</b> | <b>Descriere</b>                                                                       |
|------------------|----------------------------------------------------------------------------------------|
| file.closed      | returnează true dacă fișierul este închis, false altfel.                               |
| file.mode        | returnează modul de acces în care a fost deschis fișierul.                             |
| file.name        | returnează numele fișierului.                                                          |
| file.softspace   | returnează false dacă este necesar spațiu explicit pentru afișare (print), true altfel |

# Tratare simplă fișiere

```
fobj_in = open("/home/bugs/PycharmProjects/fisiere1/otopeni.txt")
fobj_out =
open("/home/bugs/PycharmProjects/fisiere1/otopeni1.txt","w")
i = 1
for line in fobj_in:
 print(line.rstrip())
 fobj_out.write(str(i) + ": " + line)
 i = i + 1
fobj_in.close()
fobj_out.close()
```

## Pozitionare în fișier

```
fo= open("/home/bugs/PycharmProjects/fisiere1/otopeni.txt","r+")
str = fo.read(10);
print("Sirul citit este: "+str)
pos = fo.tell() # verifică poziția curentă
print("Pozitia curenta:",pos)
pos = fo.seek(0, 0); # repozitionarea cursorului la inceputul fișierului
str = fo.read(25);
print("din nou citire:"+str)
fo.close()
```

# Citește o anume linie

```
import linecache
filePath = "input.txt"
print(linecache.getline(filePath, 2))
print(linecache.getline(filePath, 4))
linecache.clearcache()
```

# Citește câte un cuvânt

```
filePath = "input.txt"
wordList = []
wordCount = 0
#citeste linii intr-o lista
file = open(filePath, 'r')
for line in file:
 for word in line.split():
 wordList.append(word)
 wordCount += 1
print(wordList)
print("Total words = %d" % wordCount)
```

```
L=[]
infile = open(filePath, "r")
for line in infile:
 L=L+[str(s) for s in line[:-1].split(' ')]
it=iter(L)
for cuvant in it:
 print("\n"+cuvant)
infile.close()
```

# Fișiere - comenzi sistem

- de unde : import os, sys
- Metoda rename():
  - are 2 argumente: numele curent și noul nume al fișierului;

Sintaxa:

```
os.rename(current_file_name, new_file_name)
```

Exemplu:

```
import os
os.rename("test1.txt", "test2.txt")
```

- Metoda delete()
  - metoda se găsește în modulul os și are un argument
- Sintaxa:

```
os.delete(file_name)
```

Exemplu:

```
Ștergem fișierul f2.txt
import os
os.delete("f2.txt")
```

# Fișiere - comenzi sistem

Metoda mkdir()

Are un argument și creează un director cu numele argumentului

Sintaxa:

```
os.mkdir("newdir")
```

Exemplu:

```
import os
os.mkdir("test") # Create a directory "test"
```

Metoda chdir()

Are un argument care conține numele noului director curent

Sintaxa:

```
os.chdir("newdir")
```

Exemplu:

În exemplul care urmează ne poziționam pe directorul "/home/newdir":

```
import os
os.chdir("/home/newdir")
```

# Fișiere - comenzi sistem

Metoda `getcwd()`

Afișează numele directorului curent

Sintaxa:

```
os.getcwd()
```

Exemplu:

```
import os # This would give location of the current directory
os.getcwd()
```

Metoda `rmdir()`

Are un argument care conține numele directorului fișierului ce urmează a fi șters.

Sintaxa:

```
os.rmdir('dirname')
```

Exemplu:

Vom șterge directorul test; calea completă "/tmp/test".

```
import os
os.rmdir("/tmp/test")
```

# **Metode asociate fișierelor**

file.close()  
file.flush()  
file.fileno()  
file.isatty()  
file.next()  
file.read([size])  
file.readline([size])  
file.readlines([sizehint])  
file.seek(offset[, whence])  
file.tell()  
file.truncate([size])  
file.write(str)  
file.writelines(sequence)

# Tratare corectă operații I/O fișiere

```
inPath = input("Fisierul de intrare: ")
outPath = input("Fisierul de iesire: ")
try:
 file = open(inPath, 'r')#deschide pentru citire
 # incepe citirea
 file.close()
except FileNotFoundError:
 print("Eroare deschidere fisier")
try:
 file = open(outPath, 'wb') #deschide pentru scriere
 # scrie în fisier
 file.close()
except FileNotFoundError:
 print("Eroare scriere in fisier")
```

# Print to File

```
import sys

wordList = ["Red", "Blue", "Green"]
filePath = "output.txt"

def printToFile(filePath, mode, text):
 original = sys.stdout
 sys.stdout = open(filePath, mode)
 print(text)
 sys.stdout = original

for word in wordList:
 printToFile(filePath, 'a', "\n"+str(word)+" Color Adjust")
```

# Funcții

- Definiția:

```
def <name>(<formal-parameters>):
 <body>
```

```
def dictToBinary(the_dict):
 str = json.dumps(the_dict)
 binary = ''.join(format(ord(letter), 'b') for letter in str)
 return binary
```

- Apelul funcției

```
<name>(<actual-parameters>)
```

```
ceva=dictToBinary(dictonar)
```

# Funcții care întorc mai multe valori

- Câte odată o funcție trebuie să întoarcă mai mult de o valoare.

```
def sumDiff(x, y):
 sum = x + y
 diff = x - y
 return sum, diff
```

- Python suportă return multiplu precum și inițializare multiplă  
`num1, num2 = eval(input("Enter two numbers(num1, num2)"))`

```
s, d = sumDiff(num1, num2)
```

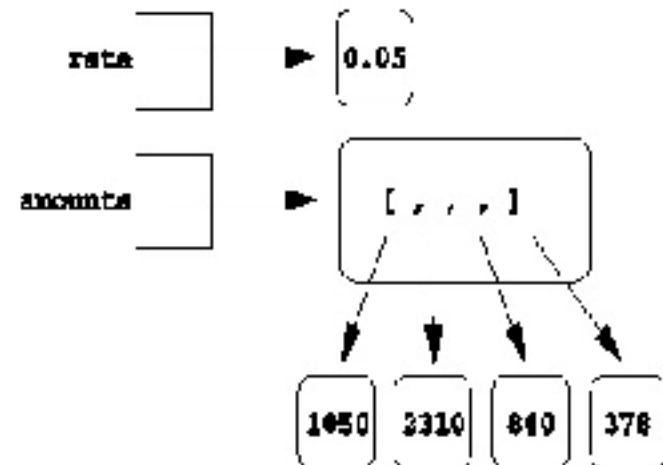
```
print("The sum is", s, "and the difference is", d)
```

# Transfer parametri prin liste/tablouri

```
def addInterest(balances, rate):
 for i in range(len(balances)):
 balances[i] = balances[i] * (1+rate)

def test():
 amounts = [1000, 2200, 800, 360]
 rate = 0.05
 addInterest(amounts, 0.05)
 print(amounts)

test()
```



# Cuvinte cheie ca argumente

- Argumentele de apel sunt de forma "*keyword = value*".

- De exemplu pentru funcția

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
 print "-- This parrot wouldn't", action,
 print "if you put", voltage, "Volts through it."
 print "-- Lovely plumage, the", type
 print "-- It's", state, "!"
```

- Putem avea un apel de forma

```
parrot(1000)
parrot(action = 'VOOOOOM', voltage = 1000000)
parrot('a thousand', state = 'pushing up the daisies')
parrot('a million', 'bereft of life', 'jump')
```

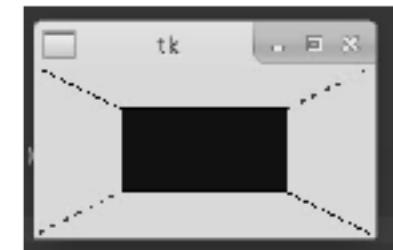
- Dar nu putem avea un apel de forma

```
parrot() # required argument missing
parrot(voltage=5.0, 'dead') # non-keyword argument following keyword
parrot(110, voltage=220) # duplicate value for argument
parrot(actor='John Cleese') # unknown keyword
```

# Grafică simplă

```
from tkinter import *
master = Tk()
w = Canvas(master, width=200, height=100)
w.pack()
w.create_line(0, 0, 200, 100)
l=w.create_line(0, 100, 200, 0, fill="red", dash=(4, 4))
r=w.create_rectangle(50, 25, 150, 75, fill="blue")
mainloop()

dacă doresc modificări directe am
w.itemconfig(r, fill="red")
w.delete(l)
sau
w.delete(ALL)
```



# Exemplu Python vs CPP

```
import time
class now:
 def __init__(self):
 self.t=time.time()
 self.year, \
 self.month, \
 self.day, \
 self.hour, \
 self.minute, \
 self.second, \
 self.dow, \
 self.doy, \
 self.dst =
 time.localtime(self.t)
 n=now()
 print("The year is" + str(n.year))
```

```
#include <stdio.h>
2 #include <time.h>
3 class now
4 {
5 public:
6 time_t t;
7 int year;
8 int month;
9 int day;
11 int hour;
12 int minute;
13 int second;
14 int dow;
15 int doy;
17
31 };
33
```

```
now()
18 { time(&t);
20 struct tm * ttime;
21 ttime = localtime(&t);
22 year = 1900 + ttime->tm_year;
23 month = ttime->tm_mon;
24 day = ttime->tm_mday;
25 hour = ttime->tm_hour;
26 minute = ttime->tm_min;
27 second = ttime->tm_sec;
28 dow = ttime->tm_wday;
29 doy = ttime->tm_yday; }

main (int argc, char ** argv)
34 {
35 now n ;
36 fprintf (stdout, "The year is %d\n"
n", n.year) ;
37 }
```

# CPP vs Python

- În C++, avem terminator de instrucțiune “;” nu și în Python.
- În C++, marcajul de bloc este cu {}, în Python, prin identare.
- În C++, clasa are un membru ascuns numit "this", care este vizibil pentru orice metodă din clasă dacă este nevoie. În Python, echivalentul este "self", iar folosirea lui este obligatorie .
- În C++ (și în C), trebuie adăugat 1900 la anul întors de `localtime()` ; În Python este rezolvată problema.
- În C++, metoda apelată atunci când se creează o instanță a clasei now poartă numele clasei: `now()`. Când o clasă este instantiată în Python, acesta caută o metodă numită `__init__()` și o apelează
- Pentru copierea unui obiect se folosește metoda `clone()`

# OOP în Python - structurarea datelor

```
class Person:
 def __init__(self, name, age, pay=0, job=None):
 self.name = name
 self.age = age
 self.pay = pay
 self.job = job
 if __name__ == '__main__':

#zona pentru testare independenta a modului
bob = Person('Bob Smith', 42, 30000, 'sweng')
sue = Person('Sue Jones', 45, 40000, 'music')
print(bob.name, sue.pay)
print(bob.name.split()[-1])
sue.pay *= 1.10
print(sue.pay)
```

# OOP în Python – adăugare comportament

```
class Person:
 def __init__(self, name, age, pay=0, job=None):
 self.name = name
 self.age = age
 self.pay = pay
 self.job = job
 def lastName(self):
 return self.name.split()[-1]
 def giveRaise(self, percent):
 self.pay *= (1.0 + percent)
if __name__ == '__main__':
 bob = Person('Bob Smith', 42, 30000, 'sweng')
 sue = Person('Sue Jones', 45, 40000, 'music')
 print(bob.name, sue.pay)
 print(bob.lastName())
 sue.giveRaise(.10)
 print(sue.pay)
```

# OOP în Python – adăugare mostenire

- Abordarea este cea cunoscută deja:

```
from person import Person
```

```
class Manager(Person):
 def giveRaise(self, percent, bonus=0.1):
 self.pay *= (1.0 + percent + bonus)
```

```
if __name__ == '__main__':
 tom = Manager(name='Tom Doe', age=50, pay=50000)
 print(tom.lastName())
 tom.giveRaise(.20)
 print(tom.pay)
```

# OOP în Python – adăugare mostenire

```
from person import Person
from manager import Manager
bob = Person(name='Bob Smith', age=42, pay=10000)
sue = Person(name='Sue Jones', age=45, pay=20000)
tom = Manager(name='Tom Doe', age=55, pay=30000)
db = [bob, sue, tom]
for obj in db:
 obj.giveRaise(.10)
for obj in db:
 print(obj.lastName(), '=>', obj.pay)
```

Smith => 11000.0

Jones => 22000.0

Doe => 36000.0

# OOP în Python – adăugare persistență

```
import shelve
from person import Person
from manager import Manager

bob = Person('Bob Smith', 42, 30000,
'sweng')

sue = Person('Sue Jones', 45, 40000,
'music')

tom = Manager('Tom Doe', 50, 50000)

db = shelve.open('class-shelve')
db['bob'] = bob
db['sue'] = sue
db['tom'] = tom

db.close()
```

```
import shelve
db = shelve.open('class-shelve')
for key in db:
 print key, '=>\n ', db[key].name, db[key].pay

bob = db['bob']
print(bob.lastName())
print(db['tom'].lastName())

#O modificare a valorilor existente în zona de persis
import shelve
db = shelve.open('class-shelve')
sue = db['sue']
sue.giveRaise(.25)
db['sue'] = sue
tom = db['tom']
tom.giveRaise(.20)
db['tom'] = tom
db.close()
```

# Clase Exemplu 2

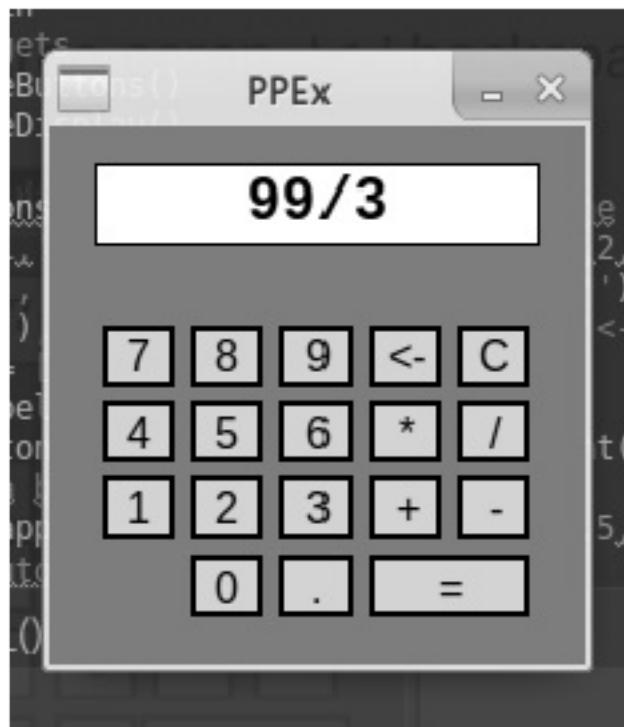
```
class Student:
 def __init__(self, name, hours, qpoints):
 self.name = name
 self.hours = float(hours)
 self.qpoints = float(qpoints)
 def getName(self):
 return self.name
 def getHours(self):
 return self.hours
 def getQPoints(self):
 return self.qpoints
 def gpa(self):
 return self.qpoints/self.hours
def makeStudent(infoStr):
 name, hours, qpoints = infoStr.split("\t")
 return Student(name, hours, qpoints)
def main():
 filename = input("Fisierul cu notele: ")
```

|                   |      |        |
|-------------------|------|--------|
| Adams, Henry      | 127  | 228    |
| Comptewell, Susan | 100  | 400    |
| DibbleBit, Denny  | 18   | 41.5   |
| Jones, Jim        | 48.5 | 155    |
| Smith, Frank      | 37   | 125.33 |

```
try:#protectia anti prost
 infile = open(filename, 'r')
 best = makeStudent(infile.readline())
 for line in infile:
 s = makeStudent(line)
 if s.gpa() > best.gpa():
 best = s
 infile.close()
except FileNotFoundError:
 print("Eroare deschidere fisier")
 print("TCel mai bun student este:",
best.getName())
 print ("ore:", best.getHours())
 print("GPA:", best.gpa())
```

## Clase exemplu 3

- Cifrele (0-9) ; punct zecimal (.) ; 4 operatii (+,-,\*,/)
- Câteva chei speciale: 'C' șterge ecran, '<- ' backspace '=' calculează



# Clasa buton

```
from graphics import *
class Button:
 def __init__(self, win, center, width,
 height, label):
 w,h = width/2.0, height/2.0
 x,y = center.getX(), center.getY()
 self.xmax, self.xmin = x+w, x-w
 self.ymax, self.ymin = y+h, y-h
 p1 = Point(self.xmin, self.ymin)
 p2 = Point(self.xmax, self.ymax)
 self.rect = Rectangle(p1,p2)
 self.rect.setFill('lightgray')
 self.rect.draw(win)
 self.label = Text(center, label)
 self.label.draw(win)
 self.deactivate()
```

```
def clicked(self, p):
 return (self.active and
 self.xmin <= p.getX() <= self.xmax and
 self.ymin <= p.getY() <= self.ymax)

def getLabel(self):
 #intoarce eticheta buton.
 return self.label.getText()

def activate(self):
 #Seteaza buton ca 'activ'
 self.label.setFill('black')
 self.rect.setWidth(2)
 self.active = True

def deactivate(self):
 #seteaza buton ca 'inactiv'
 self.label.setFill('darkgrey')
 self.rect.setWidth(1)
 self.active = False
```

# Clasa calculator

```
from graphics import *
from button import Button
class Calculator:
 def __init__(self):
 win = GraphWin("PPEx")
 win.setCoords(0,0,6,7)
 win.setBackground("slategray")
 self.win = win
 self.__createButtons()
 self.__createDisplay()
 def __createButtons(self): # creare lista butoane
 bSpecs = [(2,1,'0'), (3,1,'.'),(1,2,'1'), (2,2,'2'), (3,2,'3'),
 (4,2,'+'), (5,2,'-'),(1,3,'4'), (2,3,'5'), (3,3,'6'), (4,3,'*'),
 (5,3,'/'),(1,4,'7'), (2,4,'8'), (3,4,'9'), (4,4,'<-'),(5,4,'C')]
 self.buttons = []
 for (cx,cy,label) in bSpecs:
 self.buttons.append(Button(self.win,Point(cx,cy),.75,.75,label))
 self.buttons.append(Button(self.win, Point(4.5,1), 1.75, .75, "="))
 for b in self.buttons:
 b.activate()
```

# Clasa calculator

```
def __createDisplay(self):
 bg = Rectangle(Point(.5,.5,.5), Point(5.5,6.5))
 bg.setFill('white')
 bg.draw(self.win)
 text = Text(Point(3,6), "")
 text.draw(self.win)
 text.setFace("courier")
 text.setStyle("bold")
 text.setSize(16)
 self.display = text
def getButton(self):
 while True:
 p = self.win.getMouse()
 for b in self.buttons:
 if b.clicked(p):
 return b.getLabel()
def run(self):
 while True:
 key = self.getButton()
 self.processButton(key)

def processButton(self, key):
 text = self.display.getText()
 if key == 'C':
 self.display.setText("")
 elif key == '<-':
 # Backspace, sterge ultimul caracter
 self.display.setText(text[:-1])
 elif key == '=':
 try:
 result = eval(text)
 except:
 result = 'ERROR'
 self.display.setText(str(result))
 else:
 self.display.setText(text + key)

if __name__ == '__main__':
 # creez obj calculator
 theCalc = Calculator()
 # se lanseaza in executie.
 theCalc.run()
```

**Să sperăm că am obținut rezultatul dorit**



shutterstock.com • 256641034

# Paradigma Programarii Generice

Cursul nr. 7  
Mihai Zaharia

# Clarificări cu privire la proba practică și te(R)oRetică

1. Cum se utilizează materialele de curs

- se ia fiecare slide și se înțelege (cel mai probabil necesită să citiți în plus deoarece fiecare are un set de cunoștiințe diferite).
- se citeste documentatia UML de uml.org (iar pentru fiecare program se face si diagrama de obiecte!)
- se testează fiecare exemplu din curs - pentru fiecare se face un proiect separat
- pentru acest an nu se va cere la laborator graal vm si sdl

2. Materialele de laborator se tratează într-o manieră similară

3. La proba practică problemele de pe bilet vor fi una de kotlin și una de python

4. Cum vor fi create aceste bilete?

- direct probleme din curs (în cazul unor aspecte mai complicate)
- combinații din exemplele parțiale prezentate la curs (e.g. tkinter)

variații la problemele de laborator și temele pe acasă

5. Cum va fi testul teoretic ?

- întrebări închise: (ca la poli-țieni) se va alege dintr-un număr de variante de răspuns și acesta va fi unic
- întrebări deschise: va trebui ca studentul să răspundă liber dar să nu bată câmpii (ca la interviu firmă)

6. Cum vor fi întrebările?

Aceste vor urmări următoarele aspecte:

- teoretic (e.g. definiți principiul O sau cand se foloește fațada și când se folosește burlacul?)
- de limbaj - se dă o bucată de cod și fie se cere rezultatul execuției fie are variante de răspuns
- fie sunt nuanțe de utilizare concreta a unor instrucțiuni în anumite contexte concrete (iar stil firmă)

# Functii parametrize

```
fun random(one: Any, two: Any, three: Any): Any //ex1
```

```
fun <T> random(one: T, two: T, three: T): T // ex2
```

```
val randomGreeting: String = random("hei", "hy", "comment ca
va")//ex3
```

```
fun <K, V>put(key: K, value: V): Unit //ex4
```

# Tipuri parametrizate

```
class Sequence<T> //ex1
```

- și un exemplu de utilizare

```
val seq = Sequence<Boolean>()
```

- class Dictionary<K, V>/ex2

- și un exemplu de utilizare

```
val dict = Dictionary<String, String>()
```

# Polimorfism limitat superior

```
fun <T : Comparable<T>>min(first: T, second: T): T
{
 val k = first.compareTo(second)
 return if (k <= 0) first else second
}
```

- Deoarece Comparable este o bibliotecă standard care definește operația de compareTo rezultă că valorile lui T pot fi extinse din următoarele tipuri:

```
val a: Int = min(4, 5)
```

```
val b: String = min("e", "c")
```

- deci limitarea este la nivel de submulțimea {Int, String}

## Limitări multiple

```
fun <T>minSerializable(first: T, second: T): T
where T : Comparable<T>, T : Serializable
{
 val k = first.compareTo(second) return if (k <= 0)
 first
 else
 second
}
```

# Limitări multiple

```
class Year(val value: Year) : Comparable<Year>
{ override fun compareTo(other: Year): Int = this.value.compareTo(other.value) }
• linia următoare va da eroare la compilare
val a = minSerializable(Year(1969), Year(2001))
• Dar dacă extindem tipul pentru ca să suporte și serializare atunci va merge
class SerializableYear(val value: Int) : Comparable<SerializableYear>, Serializable
{ override fun compareTo(other: SerializableYear): Int =
 this.value.compareTo(other.value) }
val b = minSerializable(SerializableYear(1969), SerializableYear(1802))
• Și clasele pot defini limitări superioare multiple de tip
class MultipleBoundedClass<T> where T : Comparable<T>, T : Serializable
```

# Tipuri generice de date & modificatori

```
// ex1 - Java
interface Source<T>
{
 T nextT();

}
// ex2 - Java

void demo(Source<String> strs)
{
 Source<Object> objects = strs; // !!! NEPERIMIS in Java
 ...
}
```

# Tipuri generice de date

```
interface Source<out T>
{
 fun nextT(): T
}

fun demo(strs: Source<String>)
{
 val objects: Source<Any> = strs
 // acum acest lucru este permis deoarece T este clar un parametru produs/ de
 // iesire adica out
 // ...
}
```

# Tipuri generice de date

```
interface Comparable<in T>
{
 operator fun compareTo(other: T): Int
}

fun demo(x: Comparable<Number>)
{
 x.compareTo(1.0) // 1.0 are tipul Double, care este un subtip al lui
 Number

 // ECI se poate asigna x unei variabile de tipul Comparable<Double>

 val y: Comparable<Double> = x // OK!
}
```

# Tipuri generice de date

```
class Array<T>(val size: Int)
{
 fun get(index: Int): T { ... }
 fun set(index: Int, value: T) { ... }
}
```

SE observă că nu poate fi nici co- nici contra-varianță în T și asta impune o serie de limitări. De exemplu fie funcția:

```
fun copy(from: Array<Any>, to: Array<Any>)
{
 assert(from.size == to.size)
 for (i in from.indices)
 to[i] = from[i]
}
```

# Tipuri generice de date

Aceasta ar trebui să copie elemente dintr-un tablou într-altul. Să vedem cum ar arata utilizarea ei:

```
val ints: Array<Int> = arrayOf(1, 2, 3)
```

```
val any = Array<Any>(3) { "" }
```

```
copy(ints, any)
```

// ^ tipul furnizat pentru unul din parametri este Array<Int>  
dar tipul Array<Any> era cel asteptat - vezi definiția

```
fun copy(from: Array<out Any>, to: Array<Any>) { ... }
```

# Tipuri generice de date

Pentru **Foo<out T : TUpper>**, unde T este un parametru covariant mărginit superior de TUpper atunci Foo<\*> este echivalentul lui **Foo<out TUpper>**. Și înseamnă că atunci când T este necunoscut se pot citi în siguranță valori ale lui TUpper din Foo<\*>.

Pentru **Foo<in T>**, unde T este un parametru covariant de tip **Foo<\*>** este echivalent cu **Foo<in Nothing>**. Și înseamnă că nu este nimic care poate fi scris într-o manieră sigură în/către Foo<\*> atunci când T este necunoscut.

Pentru **Foo<T : TUpper>**, unde T este un parametru invariant de tip cu limitare superioară la TUpper atunci Foo<\*> este echivalent cu **Foo<out TUpper>** pentru cazul citirii unor valori și cu **<in Nothing>** pentru cazul scrierii unor valori.

Function<\*, String> echivalentă cu Function<in Nothing, String>;

Function<Int, \*> echivalentă cu Function<Int, out Any?>;

Function<\*, \*> echivalentă cu Function<in Nothing, out Any?>.

# Funcții generice

```
fun <T> singletonList(item: T): List<T> //ex1
{
 // ...
}
sau
fun <T> T.basicToString(): String // o funcție extensie
{
 // ...
}
val l = singletonList<Int>(1) //ex2
val l = singletonList(1) //ex3
```

## Constrângeri Generice

```
fun <T : Comparable<T>> sort(list: List<T>) { ... } // ex1
```

```
sort(listOf(1, 2, 3)) // OK. Int este un subtip al lui Comparable<Int>
```

```
sort(listOf(HashMap<Int, String>()))
```

```
// Error: HashMap<Int, String> NU este un subtip al lui
Comparable<HashMap<Int, String>>
```

# Constrângeri Generice

```
fun <T> copyWhenGreater(list: List<T>, threshold: T): List<String>
```

where T : CharSequence,

```
 T : Comparable<T> {
```

```
 return list.filter { it > threshold }.map { it.toString() }
```

```
}
```

## **Ștergerea tipului (Type erasure)**

- pentru intantelel lui Foo<Bar> si Foo<Baz?> în urma stergerii tipului rezultă aceași descriere și anume Foo<\*> //ex1

# Tipuri de date algebrice

sealed class List<out T>

- Apoi vom defini două implementări: una va conține un nod cu o valoare iar a doua un nod gol

class Node<T>(val value: T, val next: List<T>) : List<T>() object  
Empty : List<Nothing>()

- O listă vidă va conține numai un obiect gol (FĂRĂ NULLLLLLL)
- În acest caz vom fixa tipul acestui nod utilizând tipul Nothing

sealed class List<out T>

{ fun isEmpty() = when (this) { is Node -> false is Empty -> true} }

## Exemplu listă algebrică

```
fun size():Int= when (this)
{
 is Node -> 1 + this.next.size()
 is Empty -> 0
}
```

- Multe funcții sunt similare cu ceea ce știți de la ADT de exemplu funcția cap arată ca mai jos:

```
fun head(): T = when (this)
{
 is Node<T> -> this.value
 is Empty -> throw RuntimeException("Empty list")
}
```

## Exemplu listă algebrică

- O soluție ar fi să permitem ca T să fie parametru de intrare, chiar dacă anterior i-am spus compilatorului să nu-l admită, prin suprascrierea verificării de variație pentru această funcție:

```
fun append(t: @UnsafeVariance T): List<T> = when (this)
{
 is Node<T> -> Node(this.value, this.next.append(t))
 is Empty -> Node(t, Empty)
}
```

Cealaltă posibilitate este să declarăm append ca o funcție extensie unde parametrul tip este invariant:

```
fun <T> List<T>.append(t: T): List<T> = when (this)
{
 is Node<T> -> Node(this.value, this.next.append(t))
 is Empty -> Node(t, Empty)
}
```

# Exemplu listă algebraică

```
sealed class List<out T>
{fun isEmpty() = when (this) { is Empty -> true is Node -> false}
fun size():Int= when (this) {is Empty -> 0 is Node -> 1 + this.next.size() }
fun tail(): List<T> = when (this) { is Node -> this.next is Empty -> this}
fun head(): T = when (this)
 { is Node<T> ->this.value
 isEmpty -> throw RuntimeException("Empty list")}
operator fun get(pos: Int): T { require(pos>= 0, { "Position must be >=0" })
return when (this) {is Node<T> -> if (pos == 0) head() else
this.next.get(pos - 1) is Empty -> throw IndexOutOfBoundsException() } }
fun append(t: @UnsafeVariance T): List<T> = when (this) { is Node<T> ->
Node(this.value, this.next.append(t)) is Empty -> Node(t, Empty) }
```

# Exemplu listă algebrică

companion object

```
{ operator fun <T>invoke(vararg values: T): List<T>
 { var temp: List<T> = Empty for (value in values)
 {temp = temp.append(value)}
 return temp
}
```

```
}
```

```
private class Node<out T>(val value: T, val next: List<T>) : List<T>()
private object Empty : List<Nothing>()
```

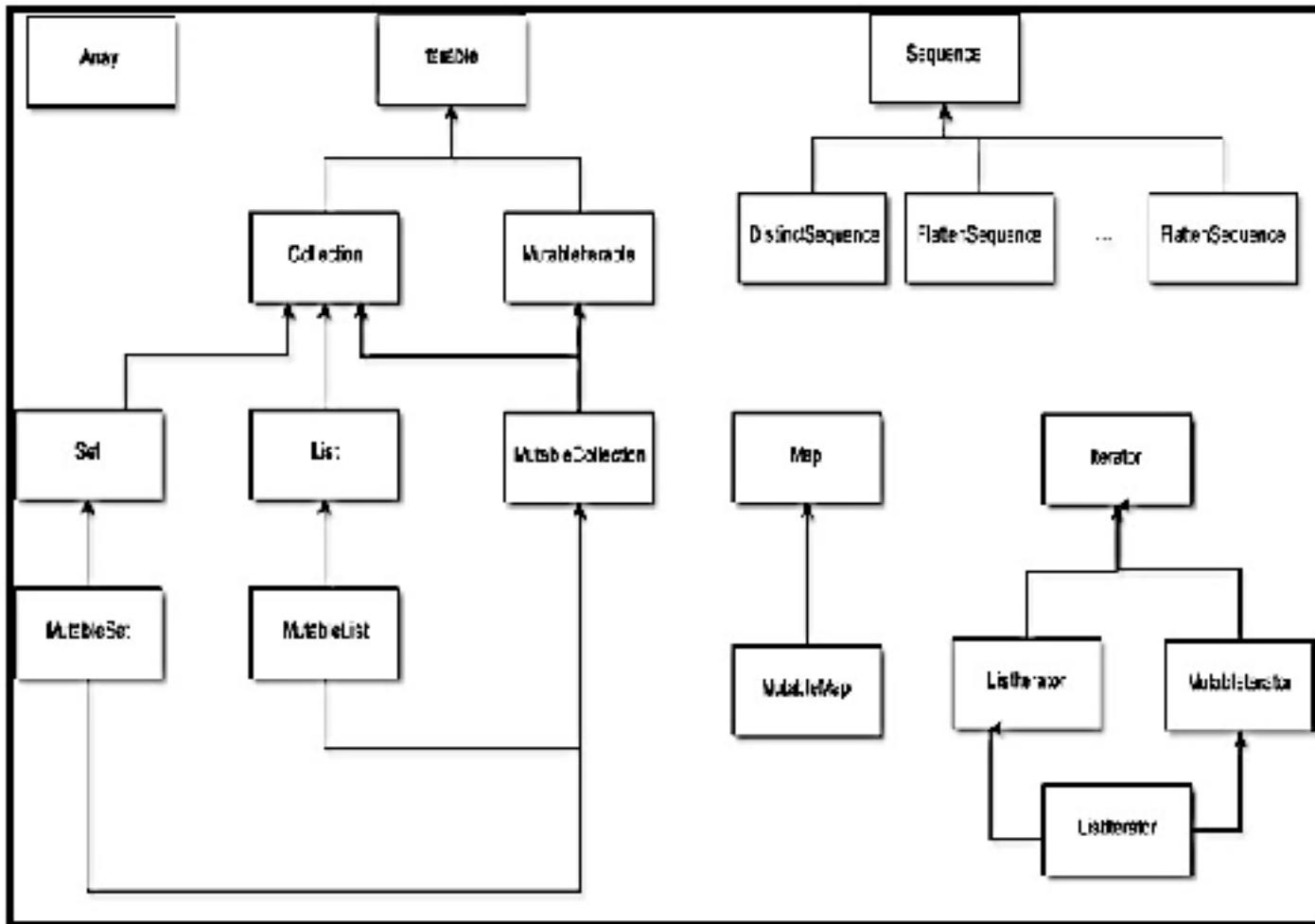
- și utilizarea :

```
val list = List("this").append("is").append("my").append("list")
println(list.size()) // prints 4
println(list.head()) // prints "this"
println(list[1]) // prints "is"
println(list.drop(2).head()) // prints "my"
```

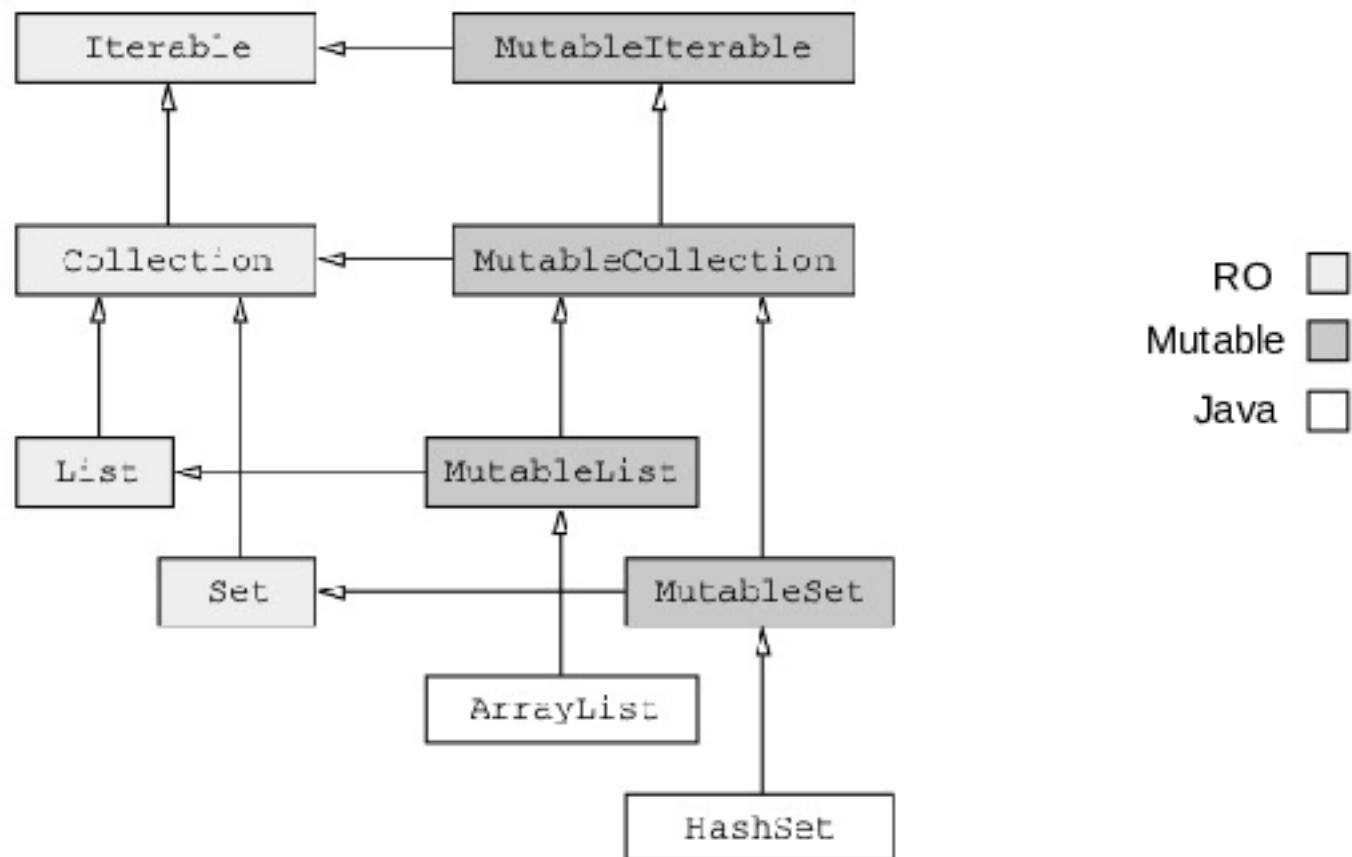
# **Colecții**

kotlin.collections namespaces

# Colecții



# Colecții



# Colecții

```
public interface Iterable<out T> //ex1
{ public abstract operator fun iterator(): Iterator<T>}

public interface Collection<out E> : Iterable<E>/> //ex2
{
 public val size: Int
 public fun isEmpty(): Boolean
 public operator fun contains(element: @UnsafeVariance E): Boolean
 override fun iterator(): Iterator<E>
 public fun containsAll(elements: Collection<@UnsafeVariance E>):
 Boolean
}
```

# Colectii Kotlin



SetKt

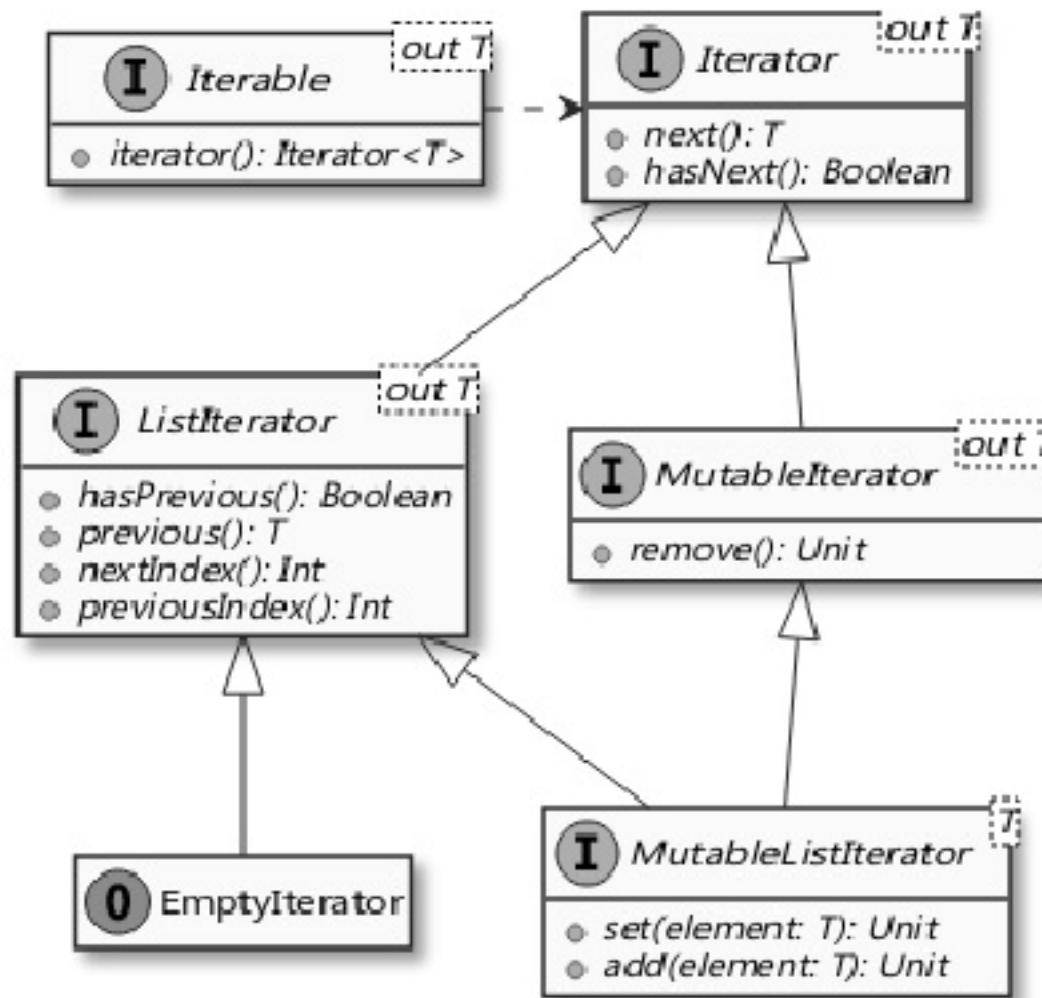
[emptySet\(\): Set<T>](#)  
[setOf\(\): Set<T>](#)  
[setOf\(T\): Set<T>](#)  
[setOf\(vararg T\): Set<T>](#)  
[mutableSetOf\(\): MutableSet<T>](#)  
[mutableSetOf\(vararg T\): MutableSet<T>](#)  
[hashSet\(\): HashSet<T>](#)  
[hashSet\(vararg T\): HashSet<T>](#)



CollectionsKt

[emptyList\(\): List<T>](#)  
[listOf\(vararg T\): List<T>](#)  
[listOf\(\): List<T>](#)  
[listOf\(T\): List<T>](#)  
[mutableListOf\(\): MutableList<T>](#)  
[mutableListOf\(vararg T\): MutableList<T>](#)  
[arrayListOf\(\): ArrayList<T>](#)  
[arrayListOf\(vararg T\): ArrayList<T>](#)

# List



# Colecții - List

```
public interface MutableIterable<out T> : Iterable<T> //ex1
{ override fun iterator(): MutableIterator<T> }
public interface List<out E> : Collection<E>
{
 //Operatii inspecție date
 override val size: Int
 override fun isEmpty(): Boolean
 override fun contains(element: Any): Boolean
 override fun containsAll(elements: Collection<@UnsafeVariance E>): Boolean
 public operator fun get(index: Int): E
 public fun indexOf(element: @UnsafeVariance E): Int
 public fun lastIndexOf(element: @UnsafeVariance E): Int
 //Iteratori Lista
 public fun listIterator(): ListIterator<E>
 public fun listIterator(index: Int): ListIterator<E>
 //Vizualizare date
 public fun subList(fromIndex: Int, toIndex: Int): List<E>
}
```

# Liste

```
val intList: List<Int> = listOfPrintln("Lista Intregi[
 ${intList.javaClass.canonicalName}]:
 ${intList.joinToString(", ")} ")
val emptyList: List<String> = emptyList<String>()
println("Lista Vida[${emptyList.javaClass.canonicalName}]:${emptyList.joinToString(", ")}")
val nonNulls: List<String> = listOfNotNull<String>(null, "a", "b","c")
println("Lista siruri nevida[${nonNulls.javaClass.canonicalName}]:${nonNulls.joinToString (",")})"
val doubleList: ArrayList<Double> = arrayListOf(84.88, 100.25, 999.99)
println("Lista Dubla:${doubleList.joinToString(",")}")
val cartoonsList: MutableList<String> = mutableListOf("Tom&Jerry",
"Dexter's Laboratory", "Johnny Bravo", "Cow&Chicken")
println("Lista DA[${cartoonsList.javaClass.canonicalName}]:
 ${cartoonsList.joinToString(",")})")
cartoonsList.addAll(arrayOf("Ed, Edd n Eddy","Ren&Stimpy"))
println("Lista DA[${cartoonsList.javaClass.canonicalName}]:
 ${cartoonsList.joinToString(",")})")
```

# Colecții - MutableList

```
public interface MutableList<E> : List<E>, MutableCollection<E>
{ //Operații de modificare
 override fun add(element: E): Boolean
 override fun remove(element: E): Boolean
 //Operații de modificare în masă
 override fun addAll(elements: Collection<E>): Boolean
 public fun addAll(index: Int, elements: Collection<E>): Boolean
 override fun removeAll(elements: Collection<E>): Boolean
 override fun retainAll(elements: Collection<E>): Boolean
 override fun clear(): Unit
 //Operatori pentru acces pozitional
 public operator fun set(index: Int, element: E): E
 public fun add(index: Int, element: E): Unit
 public fun removeAt(index: Int): E //List Iterators
 override fun listIterator(): MutableListIterator<E>
 override fun listIterator(index: Int): MutableListIterator<E>
 //Vizualizare
 override fun subList(fromIndex: Int, toIndex: Int): MutableList<E>
}
```

# Colecții - Set

```
public interface Set<out E> : Collection<E> //ex2
{ //Operatii inspecție date
override val size: Int
override fun isEmpty(): Boolean
override fun contains(element: @UnsafeVariance E): Boolean override fun iterator(): Iterator<E>
 //Operatii de masă
override fun containsAll(elements: Collection<@UnsafeVariance E>): Boolean }
public interface MutableCollection<E> : Collection<E>, MutableIterable<E> //ex2
{ //Operatii inspecție date
override fun iterator(): MutableIterator<E>
 //Operatii de modificare
public fun add(element: E): Boolean
public fun remove(element: E): Boolean
 //Operatii de masă
public fun addAll(elements: Collection<E>): Boolean
public fun removeAll(elements: Collection<E>): Boolean
public fun retainAll(elements: Collection<E>): Boolean
public fun clear(): Unit }
```

# Colecții - MutableSet

```
public interface MutableSet<E> : Set<E>, MutableCollection<E>
{
 //Operatii inspecție date
 override fun iterator(): MutableIterator<E>
 //Operații de modificare
 override fun add(element: E): Boolean
 override fun remove(element: E): Boolean
 //Operații de modificare în masă
 override fun addAll(elements: Collection<E>): Boolean
 override fun removeAll(elements: Collection<E>): Boolean
 override fun retainAll(elements: Collection<E>): Boolean
 override fun clear(): Unit
}
```

# Set

```
fun main(args: Array<String>) {
 val nums = setOf(11, 5, 3, 8, 1, 9, 6, 2)
 val sortAsc = nums.sorted()
 println(sortAsc)
 val sortDesc = nums.sortedDescending()
 println(sortDesc)
 val revNums = nums.reversed()
 println(revNums)
 val cars = setOf(Car("Mazda", 6300), Car("Toyota", 12400),
 Car("Skoda", 5670), Car("Mercedes", 18600))
 val res = cars.sortedBy { car -> car.name }
 res.forEach { e -> println(e) }
 println("*****")
 val res2 = cars.sortedByDescending { car -> car.name }
 res2.forEach { e -> println(e) }
}
```

# LinkedHashSet

```
val set = HashSet<String>()
 set.add("a")
 set.add("b")
 set.add("c")
```

```
val array = arrayOfNulls<String>(set.size)
set.toArray(array)
println("Array: ${Arrays.toString(array)}")
```

```
Set<Angajati> angajatiSet = new HashSet<>();
List<Angajati> angajatiList = new ArrayList<>();

long iterations = 1000;
Angajati angajati = new Angajati(100L, "Harry");

for (long i = 0; i < iterations; i++)
{
 angajatiSet.add(new Angajati(i, "John"));
 angajatiList.add(new Angajati(i, "John"));
}
angajatiList.add(angajati);
angajatiSet.add(angajati);
```

# HashSet

```
var hashSetOf1: HashSet<Int> = hashSetOf<Int>(2,6,13,4,29,15)
val mySet = setOf(6,4,29)
```

```
println(".....parcuregere hashSetOf1.....")
 for (element in hashSetOf1)
 { println(element)}
println("....dimensiunea lui hashSetOf1....")
 println(hashSetOf1.size)
println("....hashSetOf1 contine valoarea 13?....")
 println(hashSetOf1.contains(13))
println("....hashSetOf1 contine toate valorile din mySet?...")
 println(hashSetOf1.containsAll(mySet))
}
```

# TreeSet

```
SortedSet<String> fruits = new TreeSet<>(Comparator.reverseOrder());
```

```
fruits.add("Banana");
fruits.add("Apple");
fruits.add("Pineapple");
fruits.add("Orange");
```

```
System.out.println("Fruits Set : " + fruits);
```

# Colecții - Map

```
public interface Map<K, out V>
//Operatii inspecție date
public val size:Int
public fun isEmpty(): Boolean
public fun containsKey(key: K): Boolean
public fun containsValue(value: @UnsafeVariance V): Boolean
public operator fun get(key: K): V?
public fun getOrDefault(key: K, defaultValue: @UnsafeVariance V): V
 { //în implementarea implicită din JDK
 return null as V }
//Vizualizare
public val keys: Set<K>
public val values: Collection<V>
public val entries: Set<Map.Entry<K, V>>
public interface Entry<out K, out V>
 { public val key: K public val value: V}
}
```

# Parcurgere hartă

## cu forloop

```
val items = HashMap<String, Int>()
```

```
 items["A"] = 10
```

```
 items["B"] = 20
```

```
for ((k, v) in items)
 { println("$k = $v") }
```

## cu forEach

```
items.forEach
```

```
 { k, v -> println("$k = $v") }
```

# Colecții - MutableMap

```
public interface MutableMap<K, V> : Map<K, V>
{
 //Operații de modificare
 public fun put(key: K, value: V): V?
 public fun remove(key: K): V?
 //Operații de modificare în masă
 public fun putAll(from: Map<out K, V>): Unit
 public fun clear(): Unit
 //Vizualizare
 override val keys: MutableSet<K>
 override val values: MutableCollection<V>
 override val entries: MutableSet<MutableMap.MutableEntry<K, V>>
 public interface MutableEntry<K,V>: Map.Entry<K, V>
 { public fun setValue(newValue: V): V}
}
```

# Map - Hărți

```
data class Customer(val firstName: String, val lastName: String, val id: Int)
val carsMap: Map<String, String> = mapOf("a" to "aston martin", "b" to "bmw", "m" to "mercedes",
 "f" to "ferrari")
println("cars[${carsMap.javaClass.canonicalName}]:${carsMap}")
println("car maker starting with 'f':${carsMap.get("f")}" //Ferrari
println("car maker starting with 'X':${carsMap.get("X")}" //null
val states: MutableMap<String, String>= mutableMapOf("AL" to "Alabama", "AK" to "Alaska",
 "AZ" to "Arizona")
states += ("CA" to "California")
println("States ${states.javaClass.canonicalName}:$states")
println("States keys:${states.keys}")//AL, AK, AZ, CA
println("States values:${states.values}")//Alabama, Alaska, Arizona, California

val customers: java.util.HashMap<Int, Customer> = hashMapOf(1 to Customer("Dina", "Kreps", 1),
 2 to Customer("Andy", "Smith", 2))
val linkedHashMap: java.util.LinkedHashMap<String, String> =
linkedMapOf("red" to "#FF0000", "azure" to "#F0FFFF", "white" to "#FFFFFF")

val sortedMap: java.util.SortedMap<Int, String> = sortedMapOf(4 to "d", 1 to "a", 3 to "c", 2 to "b")
println("Sorted map[${sortedMap.javaClass.canonicalName}]:${sortedMap}")
```

# Hash Map - immutable

```
val builder = StringBuilder()
val colors = mapOf("GOLD" to "#FFD700", "YELLOW" to "#FFFF00",
 "ALICEBLUE" to "#F0F8FF", "BISQUE" to "#FFE4C4")
//initializare
builder.append("Parcurgere\n")
colors.forEach{key,value -> builder.append("\n$key:$value")}

val keys>List<String> = colors.keys.toList() //cheile trimise in lista

val values>List<String> = colors.values.toList() // valorile trimise in lista

builder.append("\n\nHashMap keys list\n") //parcurgere chei
keys.forEach{builder.append("$it,") }

builder.append("\n\nHashMap values list\n") //parcurgere valori
values.forEach{ builder.append("$it,") }
```

# HasMap mutable

```
val builder = StringBuilder()
val colors = mutableMapOf<String, String>() // initializare
mutable

colors.put("INDIANRED", "#CD5C5C") // adaug elem
colors.put("CRIMSON", "#DC143C")
 colors.put("SALMON", "#FA8072")
 colors.put("LIGHTCORAL", "#F08080")

builder.append("Parcurgere")
colors.forEach{key,value -> builder.append("\n$key,$value")}

colors.remove("CRIMSON") // stergere
 builder.append("\n\n Dupa stergerea unui element")
for ((key,value) in colors) // reafisez
 {builder.append("\n$key:$value")}

colors.put("SALMON", "NEW VALUE")
// adaugare/modificare
```

```
builder.append("\n\nEste Hash Map goala? :
${colors.isEmpty()}")

val value = colors.get("LIGHTCORAL") // citire
cheie
builder.append("\n\nLIGHTCORAL value $value")

val reds = mutableMapOf("RED" to "#FF0000",
 "FIREBRICK" to "#B22222", "CRIMSON" to
 "#DC143C")

builder.append("\n\nParcurgere harta")
reds.forEach{key,value->
 builder.append("\n$key : $value")}

textView.text = builder.toString()
```

# LinkedHashMap

```
customers.mapKeys
{
 it.toString() } // "1" =
Customer("Dina", "Kreps", 1),
 customers.map
 { it.key * 10 to it.value.id } // 10= 1, 20 =2
 customers.mapValues
 { it.value.lastName } // 1="Kreps", 2="Smith"
 customers.flatMap
 { (it.value.firstName + it.value.lastName).toSet()}.toSet()
 //D, i, n, a, K, r, e, p, s, A, d, y, S, m, t, h]
linkedHashMap.filterKeys
 { it.contains("r") } //red=#FF0000,
states.filterNot
 { it.value.startsWith("C") } //AL=Alabama, AK=Alaska,AZ=Arizona
}
```

# LinkedHashMap

```
interface Cache {
 val size: Int
 operator fun set(key: Any, value: Any)
 operator fun get(key: Any): Any?
 fun remove(key: Any): Any?
 fun clear() }
class LRUCache(private val delegate: Cache, private val minimalSize: Int =
DEFAULT_SIZE) : Cache {
 private val keyMap = object : LinkedHashMap<Any, Any> (minimalSize, .75f, true)
 { override fun removeEldestEntry(eldest: MutableMap.MutableEntry<Any, Any>):
 Boolean { val tooManyCachedItems = size > minimalSize
 if (tooManyCachedItems) eldestKeyToRemove = eldest.key
 return tooManyCachedItems }
 }
}
```

# TreeMap

```
SortedMap<String, String> fileExtensions = new TreeMap<>() //ex1 cu un comparator explicit
new Comparator<String>()
{ override public int compare(String s1, String s2)
 { return s2.compareTo(s1); } });
fileExtensions.put("python", ".py");
fileExtensions.put('kotlin', ".kt");

SortedMap<String, String> fileExtensions = new TreeMap<>(String.CASE_INSENSITIVE_ORDER); //ex2
fileExtensions.put("PYTHON", ".py");
fileExtensions.put("KOTLIN", ".kt");

TreeMap<Integer, String> employees = new TreeMap<>(); //ex3
employees.put(1003, "Rajeev");
employees.put(1001, "James");
employees.put(1002, "Sachin");
employees.put(1004, "Chris");
print(employees.size());
Integer id = 1004; // caut aangajat
if(employees.containsKey(id)) {
 // Get the value associated with a given key in a TreeMap
 String name = employees.get(id); // ii af lu numele
 print(name);
} else
 print("eroare");
```

# Colecții - Array

```
public class Array<T> : Cloneable
{
 public inline constructor(size:Int, init: (Int) ->T)
 public operator fun get(index: Int): T
 public operator fun set(index: Int, value: T): Unit
 public val size: Int
 public operator fun iterator(): Iterator<T>
 public override fun clone(): Array<T>
}
```

# Colecții - Iterator

```
public interface Iterator<out T>
{ public operator fun next(): T public operator fun hasNext(): Boolean }
public interface MutableIterator<out T> : Iterator<T>
{ public fun remove(): Unit }
public interface ListIterator<out T> : Iterator<T>
{//Operatii inspecție date
override fun next(): T
override fun hasNext(): Boolean
public fun hasPrevious(): Boolean
public fun previous(): T
public fun nextIndex(): Int
public fun previousIndex(): Int }
public interface MutableListIterator<T> : ListIterator<T>, MutableIterator<T>
{//Operatii inspecție date
override fun next(): T
override fun hasNext(): Boolean
//Operații de modificare
override fun remove(): Unit
public fun set(element: T): Unit
public fun add(element: T): Unit
}
```

# **Colecțiile Kotlin sunt deasupra celor Java**

- Kotlin-->

<-Mașina Virtuală

## **Tipuri platformă (cochilie)**

- String! sau ArrayList<Int!>! //ex1

pentru String getName() //ex2

    val name=getName (IDE va afisa String! ca tip)

    val name:String?= getName()

    val name:String = getName().

void addFlag(String flag) //ex3

    override fun addFlag(flag:String):Unit

    override fun addFlag(flag:String?).

## Cochilii - exemplu

```
fun <T> itWorks(list: List<T>): Unit
{
 println("Java Class Type:${list.javaClass.canonicalName}")
}

val jlist = ArrayList<String>()
jlist.add("sample")
itWorks(jlist)
itWorks(Collections.singletonList(1))
```

# Exemplu Array

```
val intArray = arrayOf(1, 2, 3, 4) //1
println("Int array:${intArray.joinToString(",")}")
println("Element at index 1 is:${intArray[1]}")
```

```
val stringArray = kotlin.arrayOfNulls<String>(3) //2
stringArray[0] = "a"
stringArray[1] = "b"
stringArray[2] = "c"
//stringArrays[3]="d" //genereaza eroare de pasire dimensiune
println("String array:${stringArray.joinToString(",")}")
```

```
val studentArray = Array<Student>(2) { index -> //3
 when (index)
 {
 0 -> Student(1, "Alexandra", "Brook")
 1 -> Student(2, "James", "Smith")
 else ->throw IllegalArgumentException("Prea multi")
 }
}
```

```
println("Student array:${studentArray.joinToString(",")}")
println("Student at index 0:${studentArray[0]}")
```

```
val longArray = emptyArray<Long>() //4
println("Long array:${longArray.joinToString(",")}")
```

# Funcții ajutătoare (helper) - ArraysKt

```
println("Primul element din IntArray:${ints.first()}")
println("Ultimul element din IntArray:${ints.last()}")
println("În primele trei elemente IntArray:${ints.take(3).joinToString(",")}")
println("În ultimele trei elemente din
IntArray:${ints.takeLast(3).joinToString(",")}")
println("În elementelele mai mici ca 5 din IntArray:
${ ints.takeWhile { it < 5 }.joinToString(",") }")
println("În fiecare al treilea element din IntArray:
${ints.filterIndexed { index, element -> index % 3 == 0}.joinToString(",")}")
```

## Alt exemplu tratare tablouri

```
public fun IntArray.take(n: Int): List<Int>
{
 require(n >= 0) { "Numarul de elemente $n este negativ" }
 if (n == 0) return emptyList()
 if (n >= size) return toList()
 if (n == 1) return listOf(this[0])
 var count = 0
 val list = ArrayList<Int>(n)
 for (item in this)
 {
 if (count++ == n)
 break;
 list.add(item)
 }
 return list
}
```

# Exemple conversie tablouri

```
val strings = ints.map { element ->"Item " + element.toString() }
println("Converteste elementele IntArray intr-un
string:${strings.joinToString(",")}")
public inline fun <R> IntArray.map(transform: (Int) ->R): List<R>
{
 return mapTo(ArrayList<R>(size), transform)
}
public inline fun <R, C : MutableCollection<in R>>
IntArray.mapTo(destination: C, transform: (Int) ->R): C
{
 for (item in this)
 destination.add(transform(item))
 return destination
}
```

# Exemplu conversie tablouri

```
val charArray = charArrayOf('a', 'b', 'c') //ex1
val tripleCharArray = charArray.flatMap { c ->charArrayOf(c, c,c).asIterable() }
println("Tripleaza fiecare elemnt din charArray: ${tripleCharArray.joinToString(",")}")
//ex2
public inline fun <R> CharArray.flatMap(transform: (Char) ->Iterable<R>): List<R>
 { return flatMapTo(ArrayList<R>(), transform)}
public inline fun <R, C : MutableCollection<in R>>
 CharArray.flatMapTo(destination: C, transform: (Char) -> Iterable<R>): C
 {for (element in this)
 {
 val list = transform(element)
 destination.addAll(list)
 }
 return destination }
```

# Este imutabilul garantat?

```
(intList as AbstractList<Int>).set(0, 999999) ex1
println("Lista Intregi[${intList.javaClass.canonicalName}]:${intList.joinToString(", ")}")

(nonNulls as java.util.ArrayList).addAll(arrayOf("x", "y")) //ex2
println("lista tari [${nonNulls.javaClass.canonicalName}]:${nonNulls.joinToString(", ")}")

val hacked: List<Int>= listOfNotNull(0,1) //ex3
CollectionsJ.dangerousCall(hacked)
println("Lista modificata[${hacked.javaClass.canonicalName}]:${hacked.joinToString(", ")}")

//cod Java
public class CollectionsJ
{
 public static void dangerousCall(Collection<Integer> l)
 { l.add(1000);}
}
```

# Exemplu de utilizare a celorlalte funcții

```
data class Planet(val name: String, val distance: Long)
val planets = listOf(Planet("Mercury", 57910000), Planet("Venus", 108200000), Planet("Earth", 149600000), Planet("Mars", 227940000), Planet("Jupiter", 778330000), Planet("Saturn", 1424600000), Planet("Uranus", 2873550000), Planet("Neptune", 4501000000), Planet("Pluto", 5945900000))
println(planets.last()) //Pluto
println(planets.first()) //Mercury
println(planets.get(4)) //Jupiter
println(planets.isEmpty()) //false
println(planets.isNotEmpty()) //true
println(planets.asReversed()) //"Pluto", "Neptune"
println(planets.elementAtOrNull(10)) //Null
```

# *Paradigma Modelelor de Proiectare*

Cursul nr. 8  
Mihai Zaharia

# **Design Pattern - Definiții**

- **Conform dicționarului Merriam-Webster** termenul de pattern înseamnă:
  - 1. o formă sau model propus pentru imitare
  - 2. ceva proiectat sau folosit ca model pentru a face lucruri (calapodul croitorului)
  - 3. o formă sau un proiect
  - 4. o configurație de evenimente
  - 5. ruta prestabilită a unui avion
  - 6. model comportamental
- Are ca sinonim indicat termenul de **model**

# **Cum a apărut termenul?**

Au preluat anagajatorii din IT modelul de interviu prezentat mai jos

# **Istoria evoluției conceptului în IT**

- 1987 Cunningham și Beck - limbaj
- 1990 "Gașca celor patru" – G4 - catalog
- 1995 GoF - carte

## Apoi...

- Riehle și Zullighoven menționează trei tipuri de modele software

Model conceptual

Model de proiectare

Model de  
programare

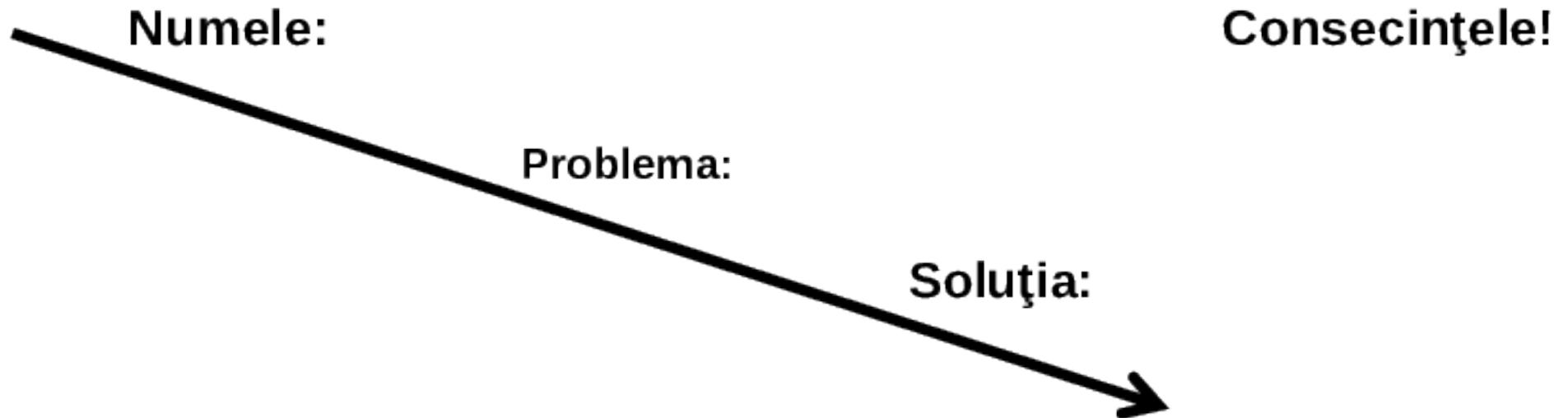
# **Unde sunt utile?**

**OOP+ADT**

**modelul aplicației**

**GoF**

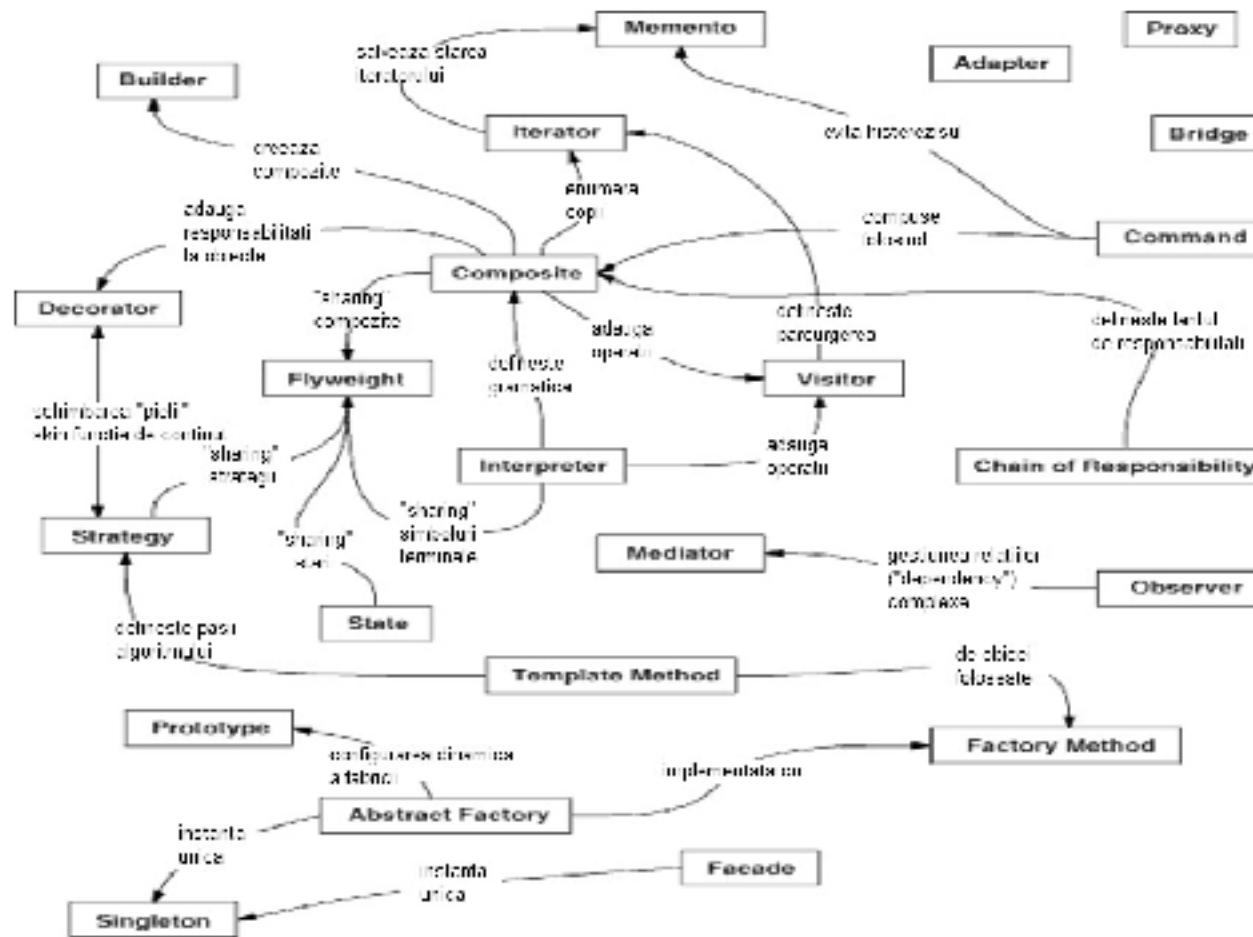
# **Elementele unui model**



# G4

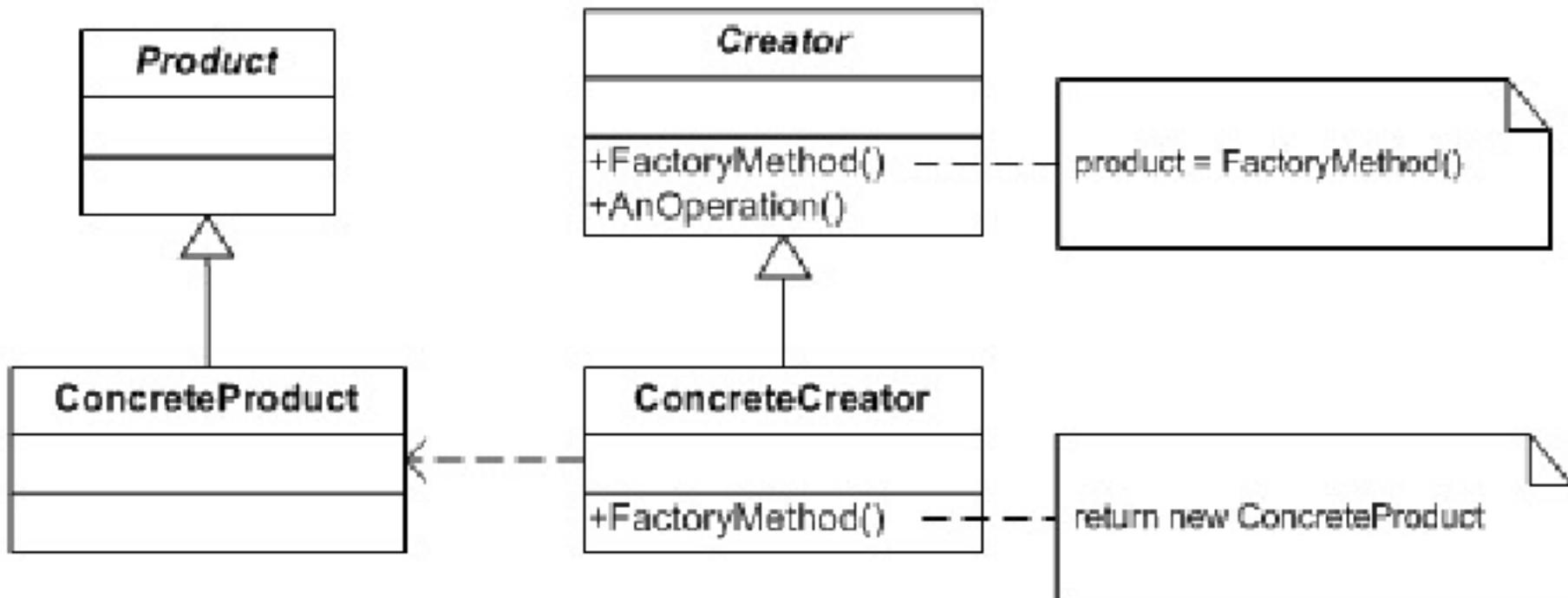
|         |        | Scop                                                  |                                                                                      |                                                                                                                   |
|---------|--------|-------------------------------------------------------|--------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
|         |        | Creational                                            | Structural                                                                           | Comportamental                                                                                                    |
| Domeniu | Clasă  | Fabric Method                                         | Adapter (clasă)                                                                      | Interpreter<br>Template Method                                                                                    |
|         | Obiect | Abstract Factory<br>Builder<br>Prototype<br>Singleton | Adapter (obiect)<br>Bridge<br>Composite<br>Decorator<br>Facade<br>Flyweight<br>Proxy | Chain of Responsibility<br>Command<br>Iterator<br>Mediator<br>Memento<br>Observer<br>State<br>Strategy<br>Visitor |

# Relații între modelele GoF

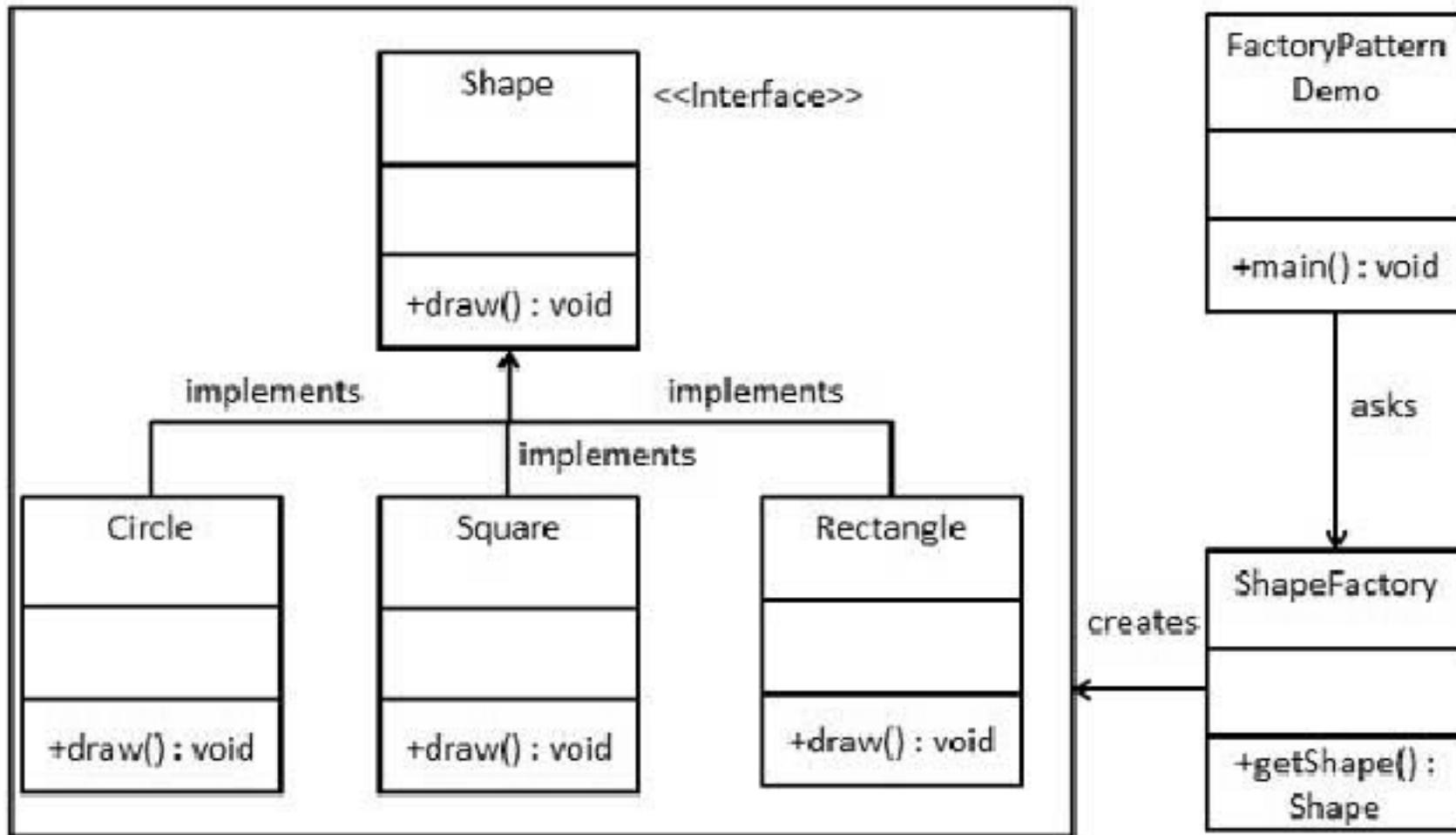


# **Modele créa&gt;**

# Modelul Fabrică de obiecte



# Modelul Fabrică de obiecte - caz de utilizare



## Modelul Fabrică de obiecte - caz de utilizare - implementare

```
interface Shape
{ fun draw() }

class ShapeFactory {
 fun getShape(shapeType: String?): Shape?
 if (shapeType.equals("CIRCLE", true))
 return Circle()
 if (shapeType.equals("RECTANGLE", true))
 return Rectangle()
 if (shapeType.equals("SQUARE", true))
 return Square()
 return null
}

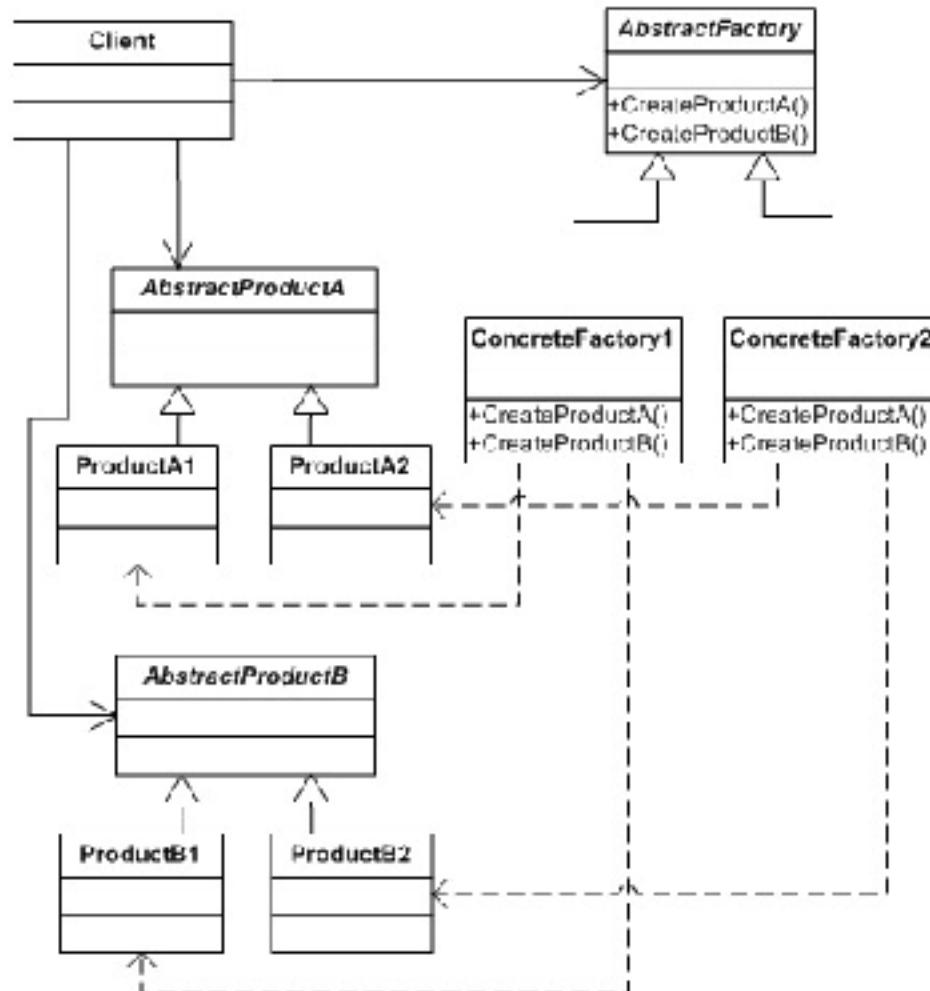
fun main(args: Array<String>)
{
 val shapeFactory = ShapeFactory()
 shapeFactory.getShape("CIRCLE")?.draw()
 shapeFactory.getShape("RECTANGLE")?.draw()
 shapeFactory.getShape("SQUARE")?.draw()
}
```

```
class Circle : Shape
{
 override fun draw()
 { println("Inside Circle::draw() method.") }
}

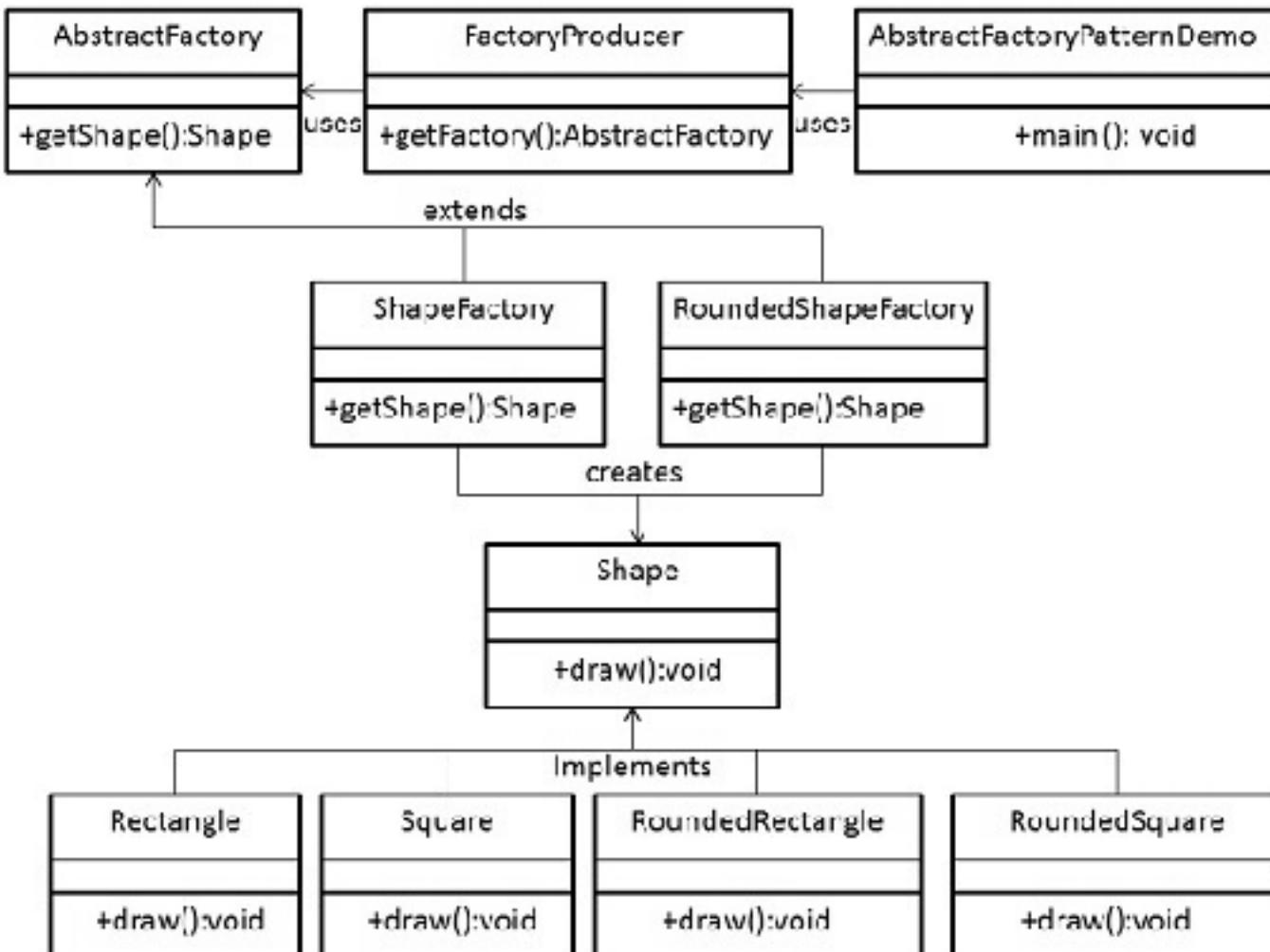
class Rectangle : Shape
{
 override fun draw()
 { println("Inside Rectangle::draw() method.") }
}

class Square : Shape
{
 override fun draw()
 { println("Inside Square::draw() method.") }
}
```

# Model Fabrica abstractă



# Modelul Fabrica abstractă - caz de utilizare



# Modelul Fabrica abstractă - implementare

```
interface Shape
{ fun draw() }
interface Color
{ fun fill() }
abstract class AbstractFactory {
 abstract fun getColor(color: String): Color?
 abstract fun getShape(shape: String): Shape?
}
class ShapeFactory : AbstractFactory() {
 override fun getShape(shape: String): Shape?
 { if (shape.equals("CIRCLE", true)) return Circle()
 if (shape.equals("RECTANGLE", true)) return Rectangle()
 if (shape.equals("SQUARE", true)) return Square()
 return null }
 override fun getColor(color: String): Color? = null }
class ColorFactory : AbstractFactory() {
 override fun getShape(shape: String): Shape? = null
 override fun getColor(color: String): Color?
 { if (color.equals("RED", true)) return Red()
 if (color.equals("GREEN", true)) return Green()
 if (color.equals("BLUE", true)) return Blue()
 return null } }
object FactoryProducer {
 fun getFactory(choice: String): AbstractFactory?
 { if (choice.equals("SHAPE", true)) return ShapeFactory()
 if (choice.equals("COLOR", true)) return ColorFactory()
 return null } }
```

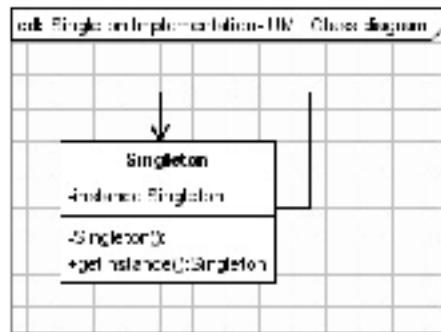
```
class Circle : Shape {
 override fun draw()
 { println("Inside Circle::draw() method.") } }
class Square : Shape {
 override fun draw()
 { println("Inside Square::draw() method.") } }
class Rectangle : Shape {
 override fun draw()
 { println("Inside Rectangle::draw() method.") } }
class Red : Color {
 override fun fill()
 { println("Inside Red::fill() method.") } }
class Green : Color {
 override fun fill()
 { println("Inside Green::fill() method.") } }
class Blue : Color {
 override fun fill()
 { println("Inside Blue::fill() method.") } }
fun main(args: Array<String>)
{ val shapeFactory = FactoryProducer.getFactory("SHAPE")
 shapeFactory?.getShape("CIRCLE")?.draw()
 shapeFactory?.getShape("RECTANGLE")?.draw()
 shapeFactory?.getShape("SQUARE")?.draw()

 val colorFactory = FactoryProducer.getFactory("COLOR")
 colorFactory?.getColor("RED")?.fill()
 colorFactory?.getColor("GREEN")?.fill()
 colorFactory?.getColor("BLUE")?.fill() }
```

# Modelul burlacului

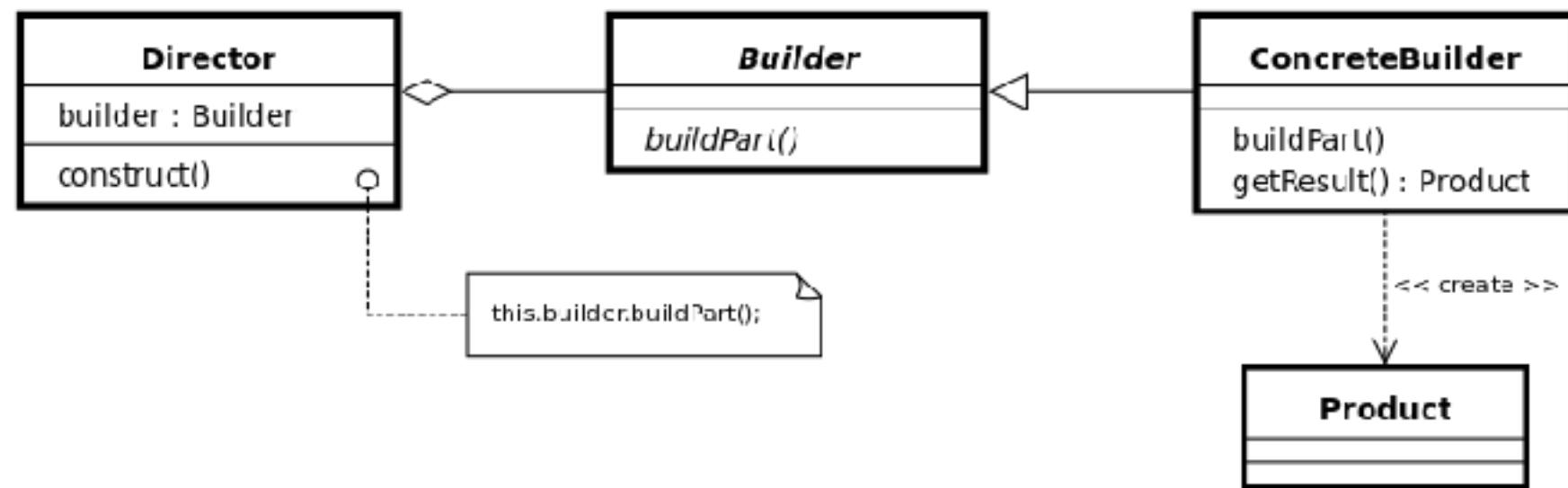
## object burlac

```
object Payroll
{
 val allEmployees = arrayListOf<Person>()
 fun calculateSalary()
 {
 for (person in allEmployees)
 {
 ...
 }
 }
}
```



# **Modelul constructor**

# Modelul constructor



# Model constructor - implementare concretă

```
data class Mail(val to: String,
 val title: String = "",
 val message: String = "",
 val cc: List<String> = listOf(),
 val bcc: List<String> = listOf(),
 val attachments: List<java.io.File> = listOf())
```

```
class MailBuilder(val to: String)
{
 private var mail: Mail = Mail(to)
 fun title(title: String): MailBuilder
 {
 mail.title = title
 return this
 }
 // acesta se repeta pentru alte variatii
 fun build(): Mail
 { return mail }
}
```

- și utilizare imediată:

```
val mail = Mail("one@recipient.org",
 "Hi", "How are you")
```

- sau utilizare de obiect construit particularizat:

```
val email = MailBuilder("hello@hello.com").title
 ("What's up?").build()
```

# Modelul prototip

# Model prototip - implementare de caz

```
open class Bike : Cloneable
{
 private var gears: Int = 0
 private var bikeType: String? = null
 var model: String? = null
 private set

 init
 {
 bikeType = "Standard"
 model = "Carpati"
 gears = 4
 }

 public override fun clone(): Bike {
 return Bike()
 }
}
```

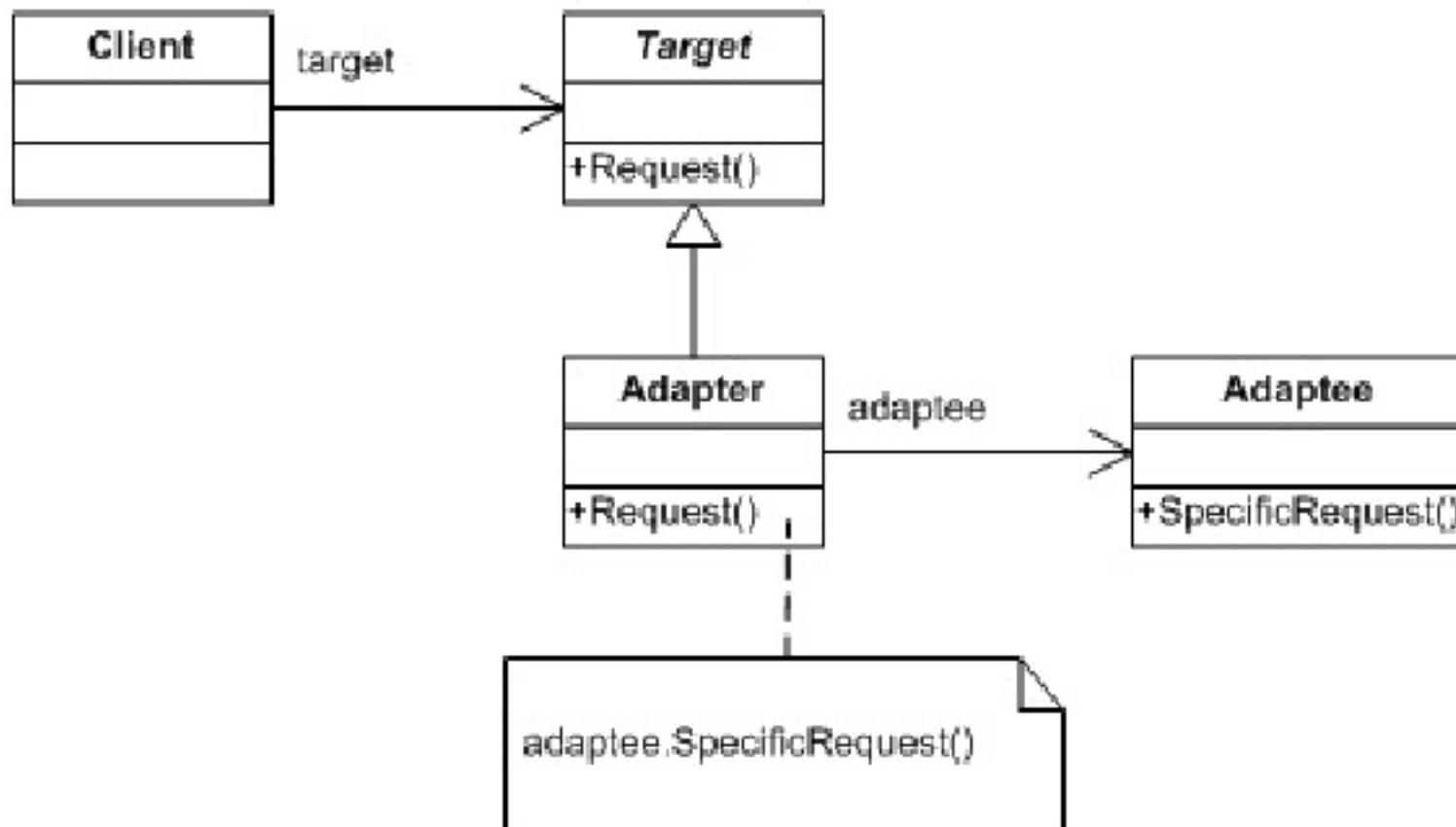
```
 fun makeAdvanced()
 {
 bikeType = "Advanced"
 model = "Jaguar"
 gears = 6
 }
 }

 fun makeJaguar(basicBike: Bike): Bike
 {
 basicBike.makeAdvanced()
 return basicBike
 }

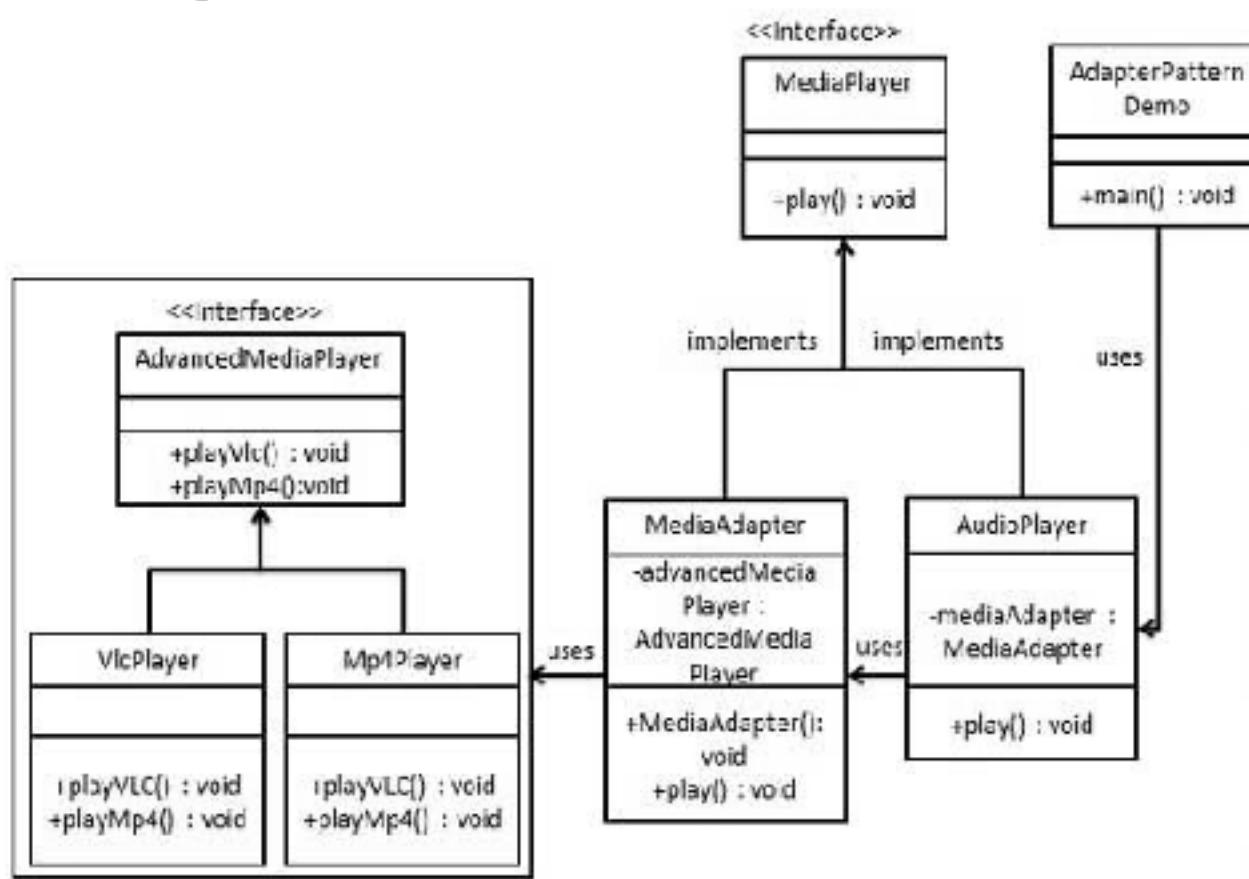
 fun main(args: Array<String>)
 {
 val bike = Bike()
 val basicBike = bike.clone()
 val advancedBike = makeJaguar(basicBike)
 println("Bicicleta mai buna: " + advancedBike.model!!)
 }
}
```

# **Modele structurale**

# Modelul Adaptor



# Model Adaptor - caz de utilizare

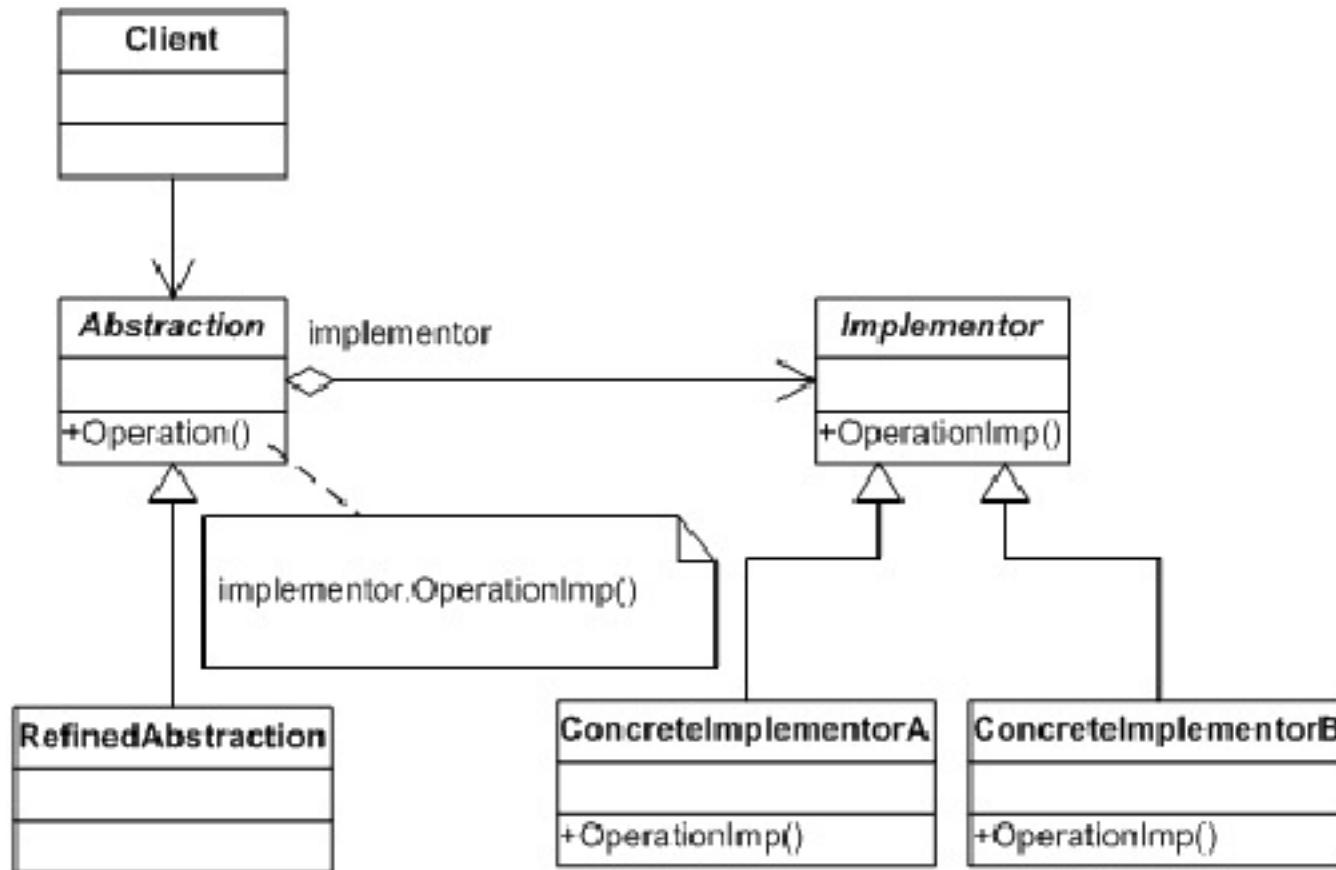


# Model Adaptor - implementare

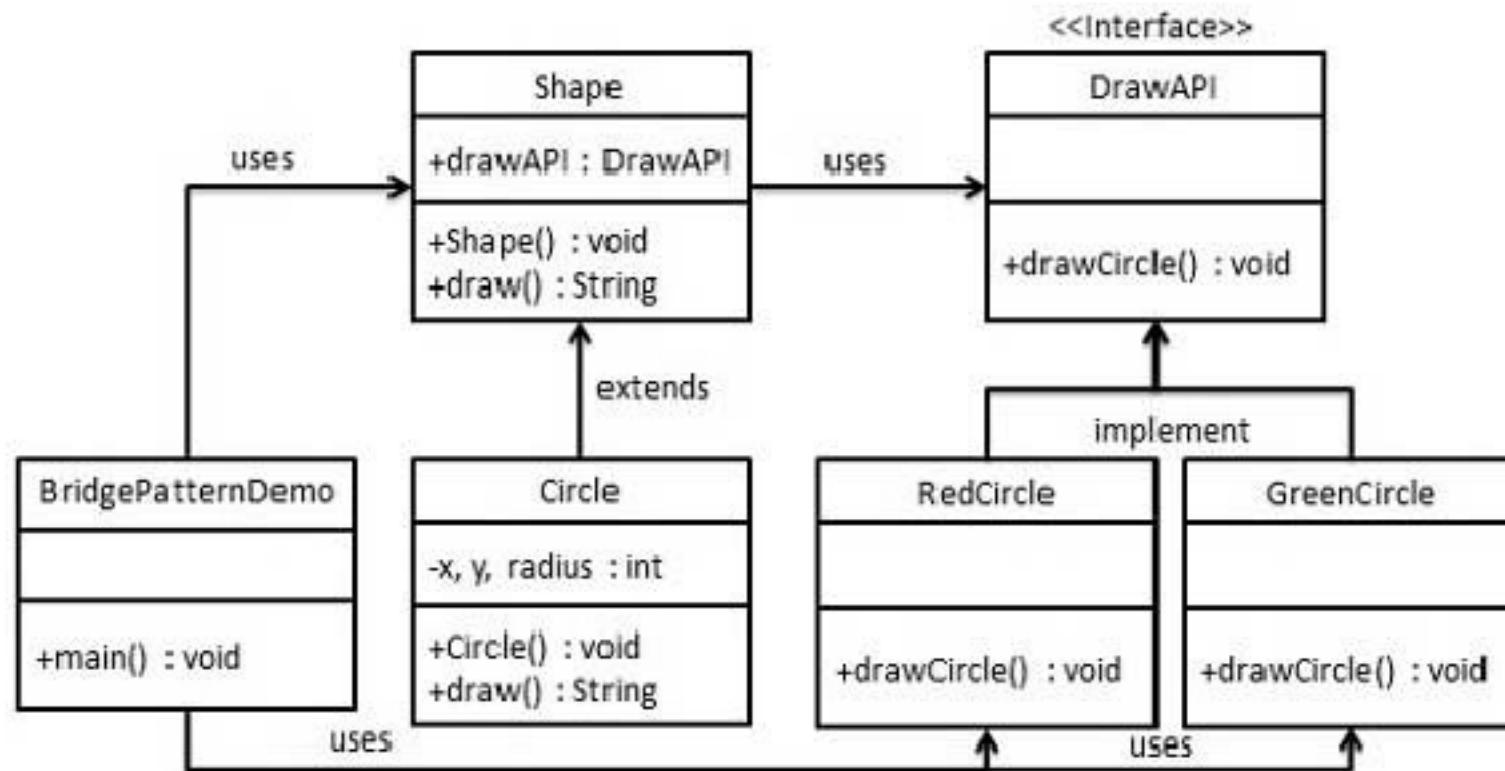
```
interface AdvanceMediaPlayer
 {fun playVlc(fileName: String)
 fun playMp4(fileName: String) }
interface MediaPlayer
 { fun play(audioType: String, fileName: String) }
open class MediaAdapter : MediaPlayer
{ private var advancedMusicPlayer: AdvanceMediaPlayer? = null
 override fun play(audioType: String, fileName: String)
 { if (audioType.equals("vlc", true))
 {if (advancedMusicPlayer == null)
 { advancedMusicPlayer = VlcPlayer()
 advancedMusicPlayer?.playVlc(fileName) }
 else if (audioType.equals("mp4", true))
 {if (advancedMusicPlayer == null)
 { advancedMusicPlayer = Mp4Player()
 advancedMusicPlayer?.playMp4(fileName) } } }
 }
class AudioPlayer : MediaAdapter()
{ override fun play(audioType: String, fileName: String)
 { if (audioType.equals("mp3", true))
 { println("Playing mp3 file. Name: $fileName ") }
 else if (audioType.equals("vlc", true) || audioType.equals("mp4", true))
 { MediaAdapter().play(audioType, fileName) }
 else { println("Invalid media. $audioType format not supported") } } }
```

```
class Mp4Player : AdvanceMediaPlayer {
 override fun playMp4(fileName: String) {
 println("Playing mp4 file. Name: $fileName")
 }
 override fun playVlc(fileName: String) {
 println("Only support mp4 type")
 }
}
class VlcPlayer : AdvanceMediaPlayer {
 override fun playMp4(fileName: String) {
 println("Only support vlc type")
 }
 override fun playVlc(fileName: String) {
 println("Playing vlc file. Name: $fileName")
 }
}
fun main(args: Array<String>) {
 val audioPlayer = AudioPlayer()
 audioPlayer.play("mp3", "beyond the horizon.mp3")
 audioPlayer.play("mp4", "alone.mp4")
 audioPlayer.play("vlc", "far far away.vlc")
 audioPlayer.play("avi", "mind me.avi") }
```

# Modelul Pod



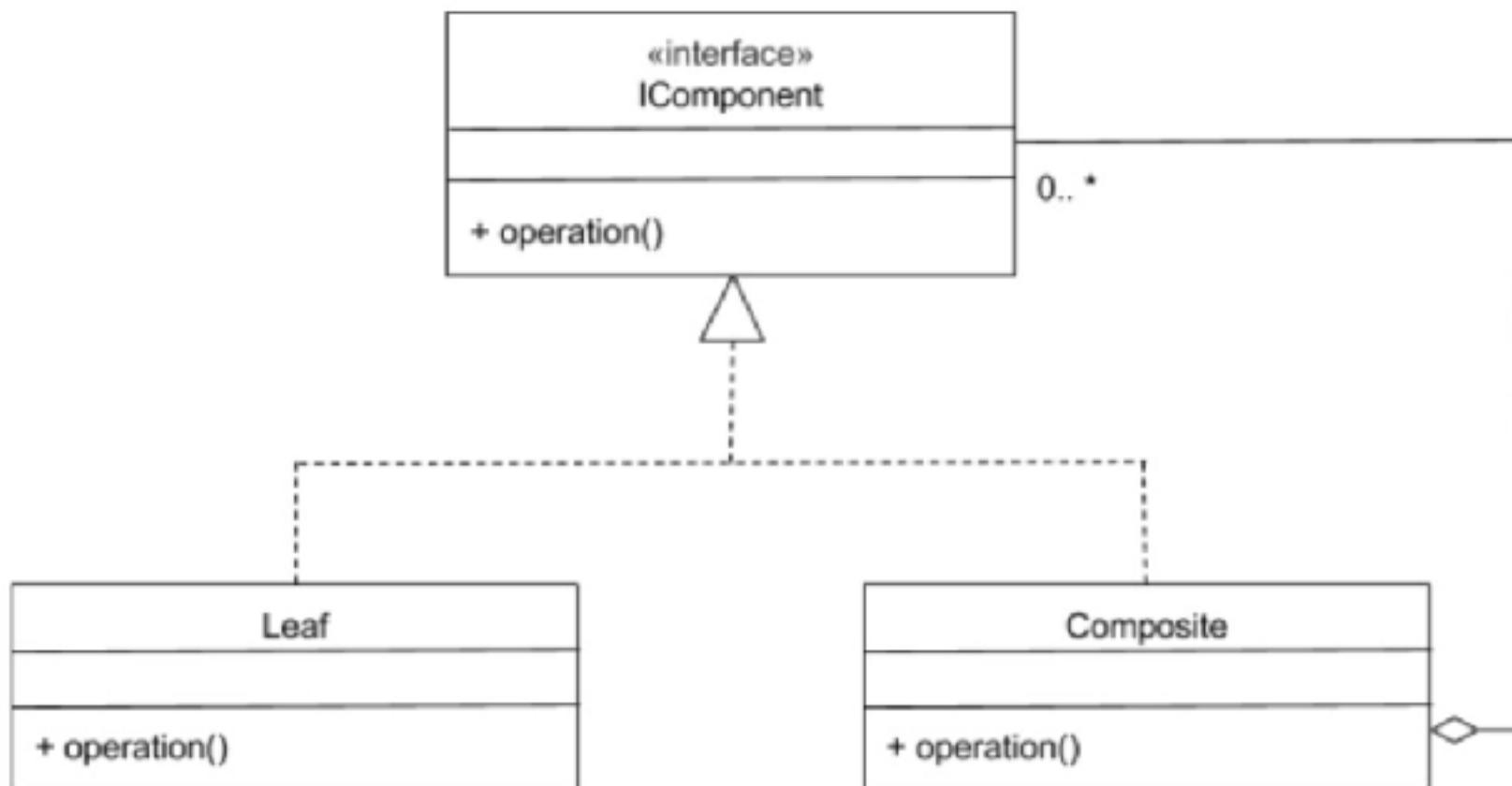
# Modelul Pod - caz de utilizare



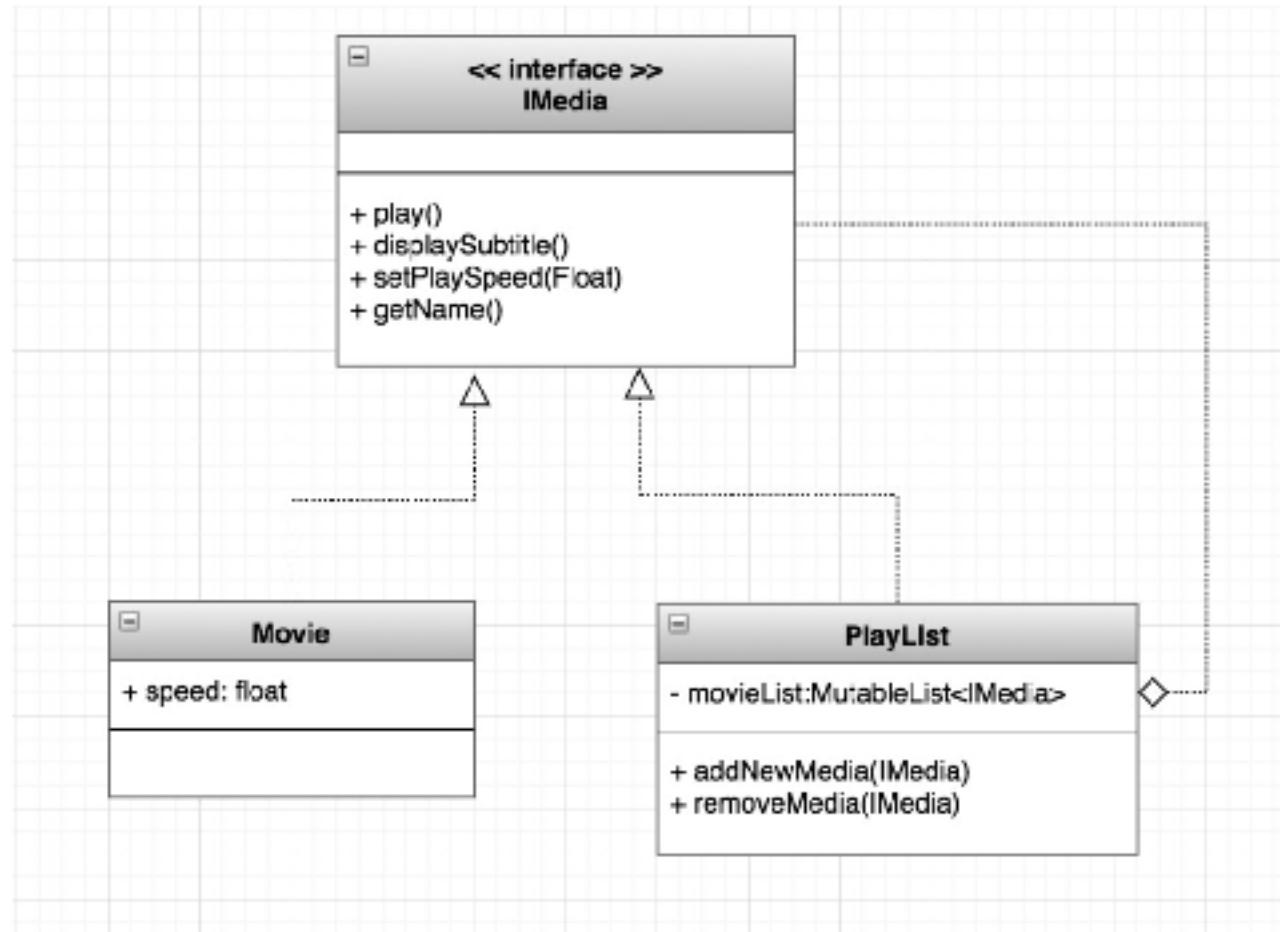
# Modelul Pod - implementare

```
interface DrawAPI
{ fun drawCircle(radius: Int, x: Int, y: Int); }
abstract class Shape(protected val drawAPI: DrawAPI)
{ abstract fun draw() }
class Circle(val x: Int, val y: Int, val radius: Int, drawAPI: DrawAPI) : Shape(drawAPI)
{ override fun draw()
 { drawAPI.drawCircle(radius, x, y) }
}
class GreenCircle : DrawAPI
{ override fun drawCircle(radius: Int, x: Int, y: Int)
 { println("Drawing Circle[color: green, radius: $radius, x: $x, y: $y]") }
}
class RedCircle : DrawAPI
{ override fun drawCircle(radius: Int, x: Int, y: Int)
 { println("Drawing Circle[color: red, radius: $radius, x: $x, y: $y]") }
}
fun main(args: Array<String>)
{ val redCircle = Circle(100, 100, 10, RedCircle())
 val greenCircle = Circle(100, 100, 10, GreenCircle())
 redCircle.draw()
 greenCircle.draw()
}
```

# Model Compus - forma generală



# Model Compus - caz de utilizare



# Compus - caz de utilizare - implementare

```
interface IMedia
{
 fun play()
 fun displaySubtitle()
 fun setPlaySpeed(speed:Float)
 fun getName():String
}

class Movie(val title:String):IMedia
{
 private var speed = 1f
 override fun play()
 {
 println("Now playing: ${title}...")
 }
 override fun displaySubtitle()
 {
 println("display subtitle")
 }
 override fun setPlaySpeed(speed:Float)
 {
 this.speed = speed
 println("current play speed set to: $speed")
 }
 override fun getName():String
 {
 return title
 }
}

class PlayList(val title:String):IMedia
{
 var movieList:MutableList<IMedia> = mutableListOf()
 fun addNewMedia(media:IMedia) = movieList.add(media)
 fun removeMedia(media:IMedia)
 {
 movieList = movieList.filter{ it.getName() != media.getName() }.toMutableList()
 }
 override fun play()
 {
 movieList.forEach { it.play() }
 }
 override fun displaySubtitle()
 {
 println("display certain subtitle")
 }
 override fun setPlaySpeed(speed:Float)
 {
 movieList.forEach { it.setPlaySpeed(speed) }
 }
 override fun getName():String
 {
 return title
 }
}
```

```
fun main(args:Array<String>)
{
 val actionMoviePlayList:PlayList = PlayList("Action Movies")
 val movieB:IMedia = Movie("The Dark Knight")
 val movieC:IMedia = Movie("Inception")
 val movieD:IMedia = Movie("The Matrix")

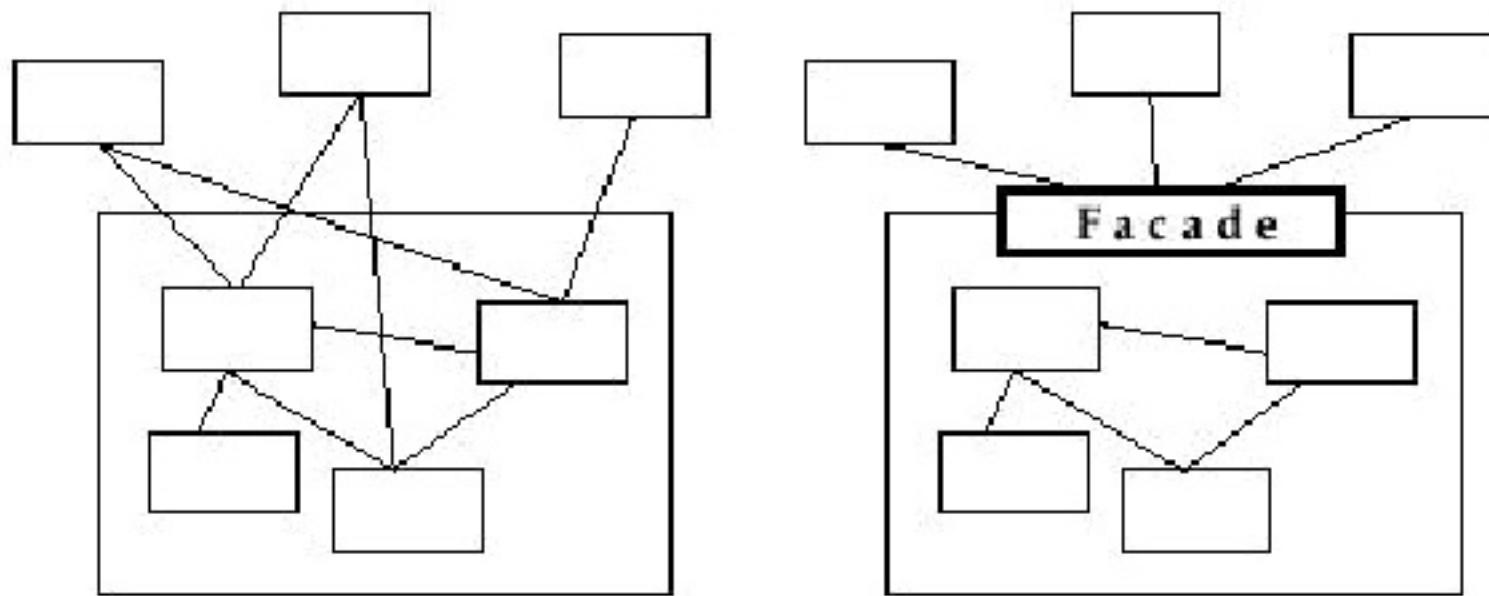
 actionMoviePlayList.apply
 {
 addNewMedia(movieB)
 addNewMedia(movieC)
 addNewMedia(movieD)
 }

 val dramaPlayList:PlayList = PlayList("Drama Play List")
 val movie1:IMedia = Movie("The Godfather")
 val movie2:IMedia = Movie("The Shawshank Redemption")

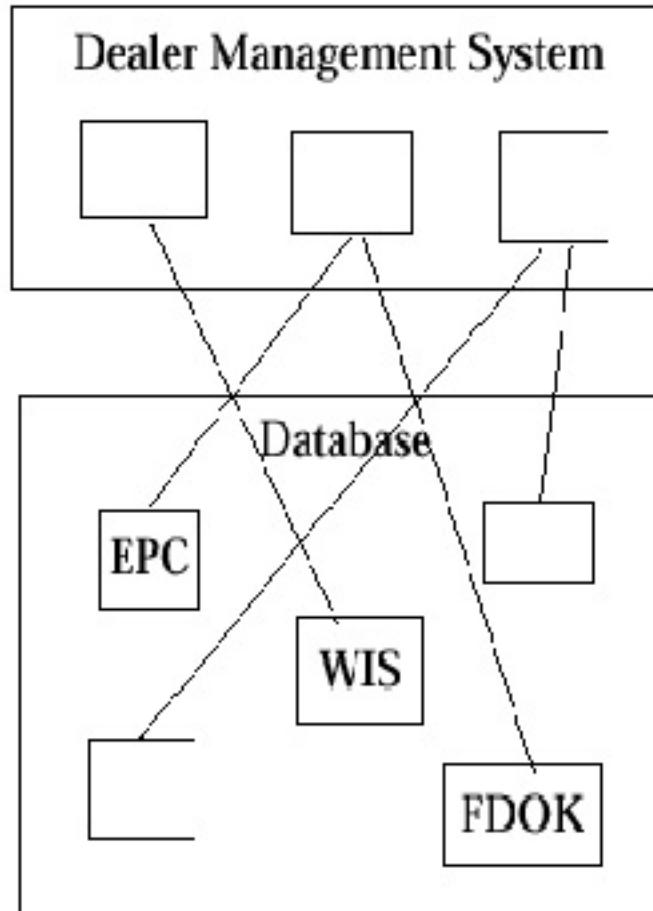
 dramaPlayList.apply
 {
 addNewMedia(movie1);
 addNewMedia(movie2)
 }

 val myPlayList:PlayList = PlayList("My Play List")
 myPlayList.apply
 {
 addNewMedia(actionMoviePlayList)
 addNewMedia(dramaPlayList)
 }
 myPlayList.play()
}
```

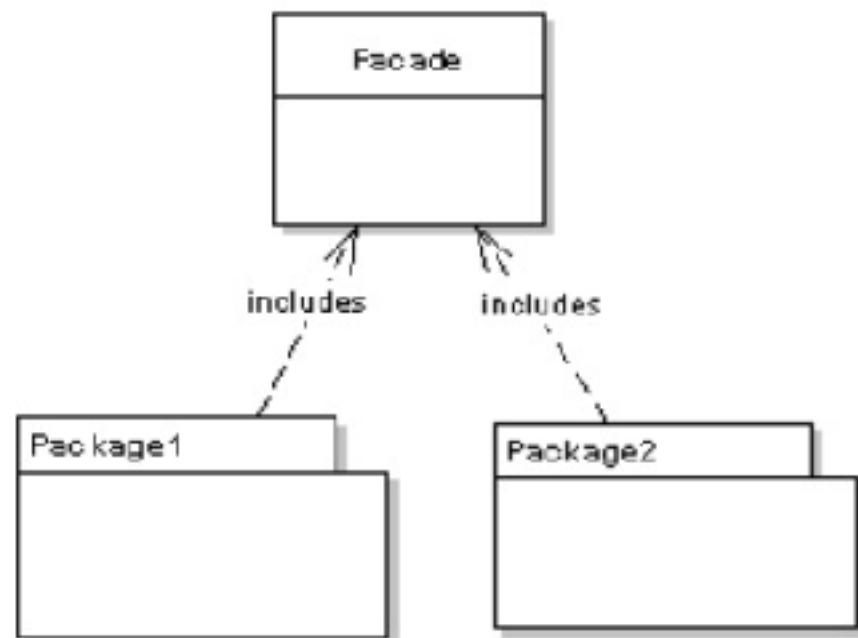
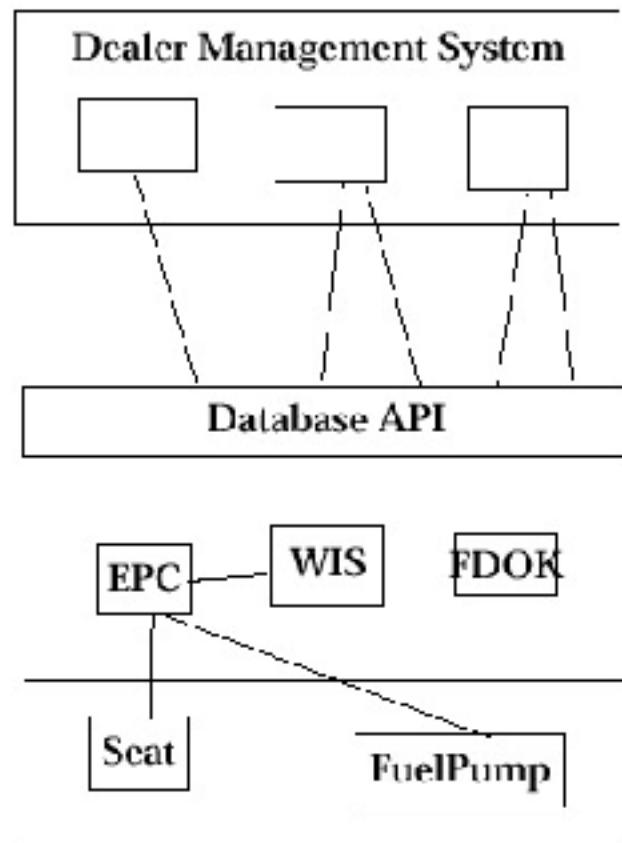
# Model Fațadă



# Arhitectură deschisă



# Arhitectură închisă



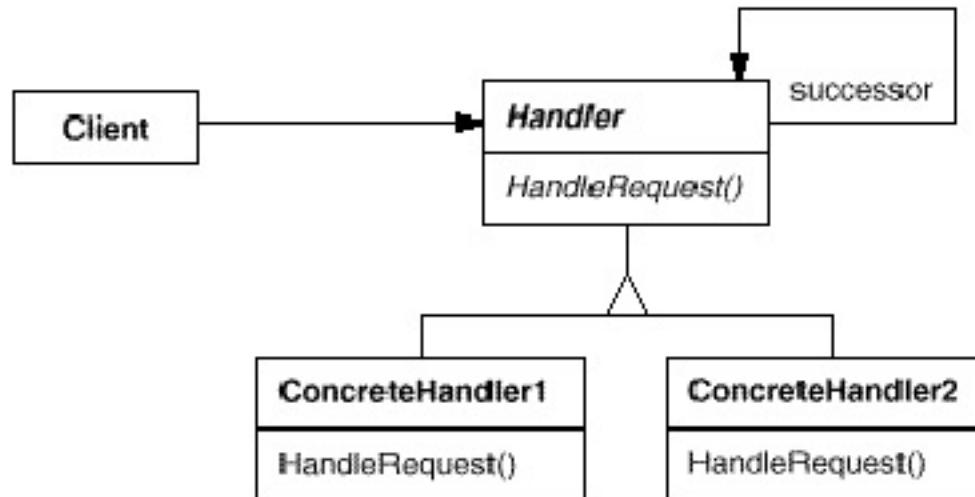
# Model Fațadă - implementare

```
class CPU
{ fun freeze() = println("Freezing.")
 fun jump(position: Long) = println("Jump to $position.")
 fun execute() = println("Executing.") }
class HardDrive
{ fun read(lba: Long, size: Int): ByteArray = byteArrayOf() }
class Memory
{ fun load(position: Long, data: ByteArray) = println("Loading from memory position: $position") }

/* Fata de */
class Computer(val processor: CPU = CPU(), val ram: Memory = Memory(), val hd: HardDrive = HardDrive())
{ companion object
 { val BOOT_ADDRESS = 0L
 val BOOT_SECTOR = 0L
 val SECTOR_SIZE = 0 }
 fun start()
 { processor.freeze()
 ram.load(BOOT_ADDRESS, hd.read(BOOT_SECTOR, SECTOR_SIZE))
 processor.jump(BOOT_ADDRESS)
 processor.execute() }
}
fun main(args: Array<String>)
{ val computer = Computer()
 computer.start() }
```

# **Modele comportamentale**

# Modelul lanț de responsabilități



Unde o structură tipică de înlățuire de obiecte ar fi



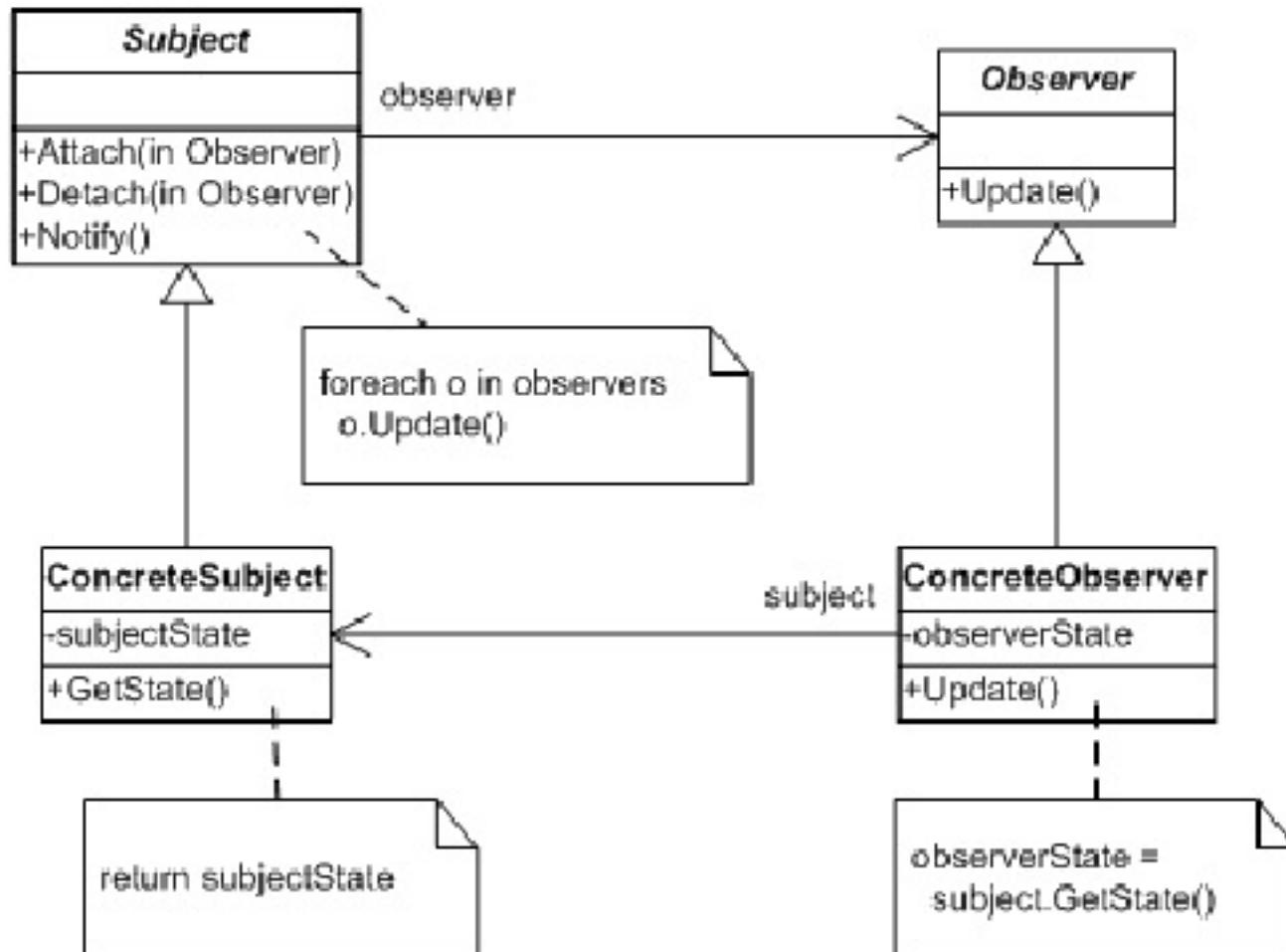
# Modelul lanț de responsabilități - implementare

```
import org.assertj.core.api.Assertions.assertThat
import org.junit.jupiter.api.Test
interface HeadersChain
{ fun addHeader(inputHeader: String): String }
class AuthenticationHeader(val token: String?, var next: HeadersChain? = null) : HeadersChain
{ override fun addHeader(inputHeader: String): String
 { token ?: throw IllegalStateException("Token should be not null")
 return inputHeader + "Authorization: Bearer $token\n"
 .let{ next?.addHeader(it) ?: it } } }
class ContentTypeHeader(val contentType: String, var next: HeadersChain? = null) : HeadersChain
{ override fun addHeader(inputHeader: String): String =
 inputHeader + "ContentType: $contentType\n"
 .let{ next?.addHeader(it) ?: it } }
class BodyPayload(val body: String, var next: HeadersChain? = null) : HeadersChain
{ override fun addHeader(inputHeader: String): String =
 inputHeader + "$body"
 .let{ next?.addHeader(it) ?: it } }
class ChainOfResponsibilityTest
{
 @Test
 fun `Chain Of Responsibility`()
 { //crearea elementelor lanțului
 val authenticationHeader = AuthenticationHeader("123456")
 val contentTypeHeader = ContentTypeHeader("json")
 val messageBody =
 BodyPayload("Body:\n\n\"username\":\"dbacinski\"\n") }

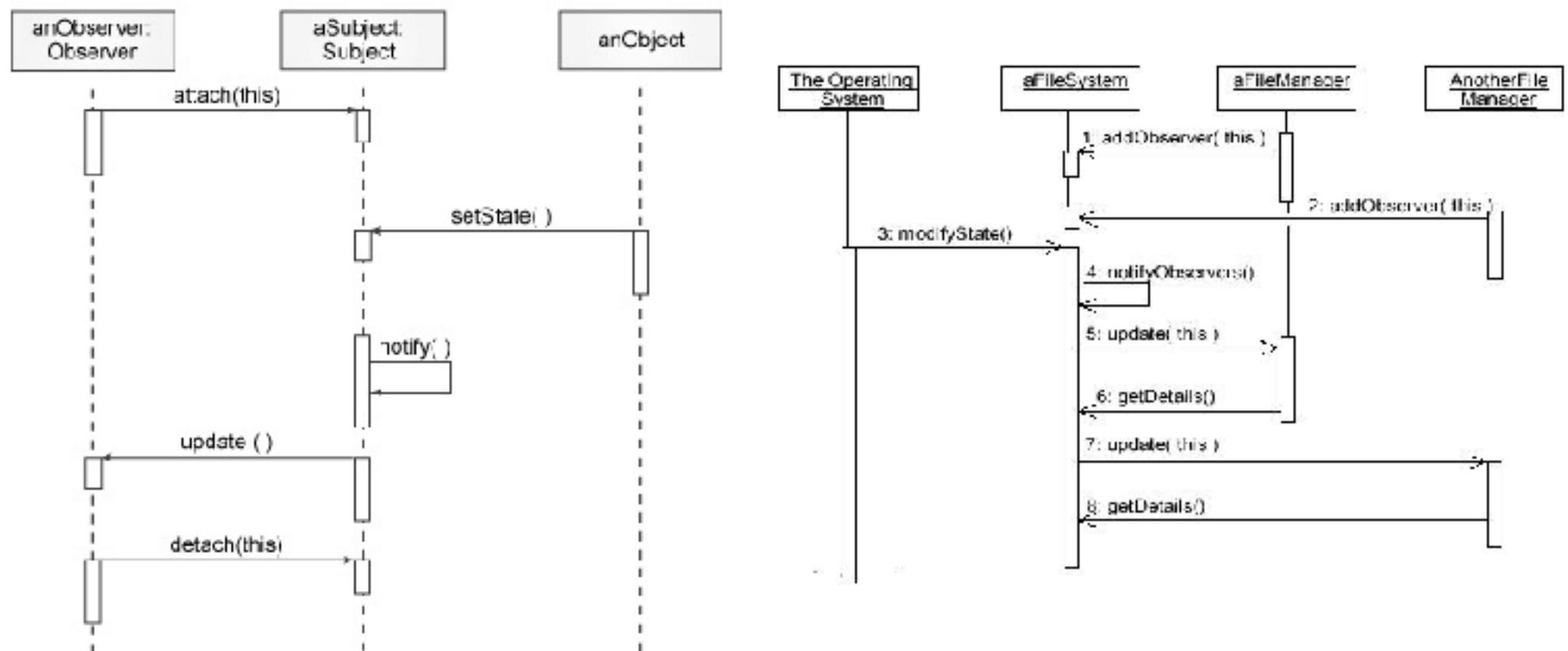
 //se construieste lanțul
 authenticationHeader.next = contentTypeHeader
 contentTypeHeader.next = messageBody
 //se executa lanțul
 val messageWithAuthentication =
 authenticationHeader.addHeader("Headers with Authentication:\n")
 println(messageWithAuthentication)
 val messageWithoutAuth =
 contentTypeHeader.addHeader("Headers:\n")
 println(messageWithoutAuth)
 assertThat(messageWithAuthentication).isEqualTo(
 """
 Headers with Authentication:
 Authorization: Bearer 123456
 ContentType: json
 Body:
 { "username":"bonjovi2987" }
 """.trimIndent())
 assertThat(messageWithoutAuth).isEqualTo(
 """
 Headers:
 ContentType: json
 Body:
 { "username":"dbacinski" }
 """.trimIndent())
 }
}
```

# **Modelul Observator**

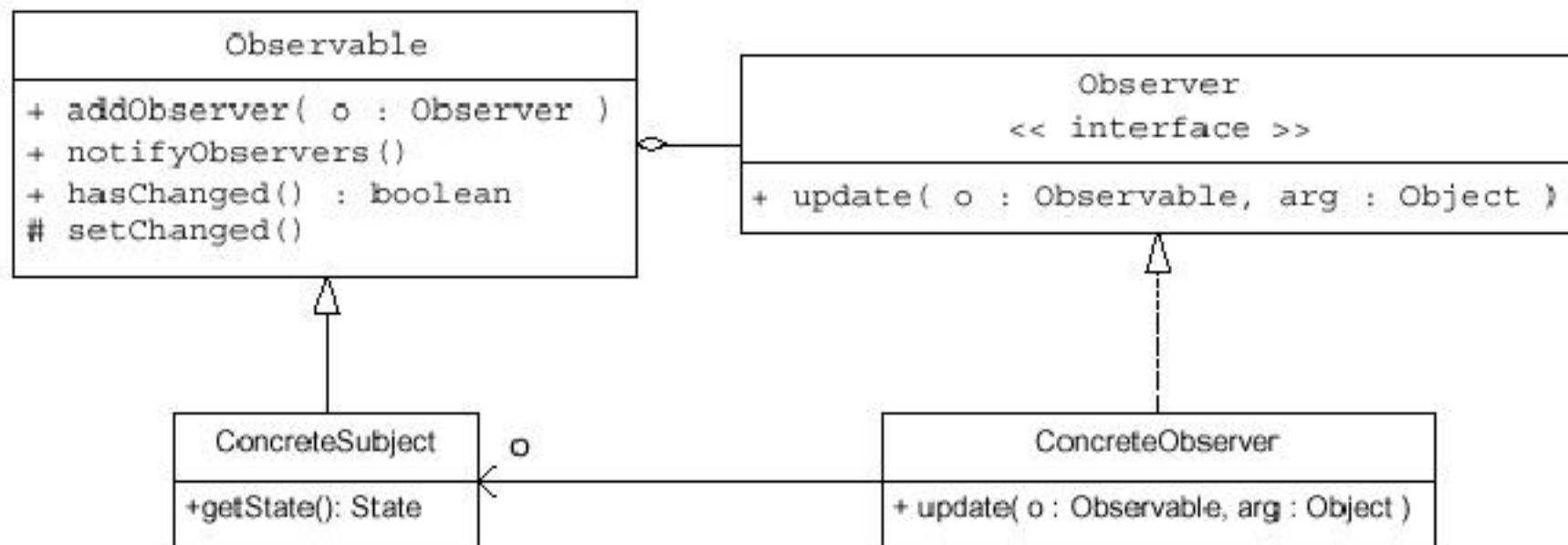
# Modelul Observator



# Model Observator - Secvențiere temporală



# Suportul apelat din spate de la Java



```
interface ValueChangeListener {
 fun onValueChanged(newValue:String)
}

class PrintingTextChangedListener : ValueChangeListener {
 override fun onValueChanged(newValue: String) =
 println("Text is changed to: $newValue")
}

class ObservableObject(listener:ValueChangeListener) {
 var text: String by Delegates.observable(
 initialValue = "",
 onChange = {
 prop, old, new ->
 listener.onValueChanged(new)
 }
)
}

fun main(args: Array<String>) {
 val observableObject = ObservableObject(PrintingTextChangedListener())
 observableObject.text = "Hello"
 observableObject.text = "There"
}
```

# Model Observator - Caz de utilizare

```
import kotlin.properties.ObservableProperty
//wrapper peste suportul - Java Observer
import kotlin.properties.ReadWriteProperty
import kotlin.reflect.KProperty

inline fun <T> observable(initialValue: T,
crossinline beforeChange:
 (property: KProperty<*>, oldValue: T, newValue: T) -> Boolean,
crossinline afterChange:
 (property: KProperty<*>, oldValue: T, newValue: T) ->Unit):
 ReadWriteProperty<Any?, T> = object:
 ObservableProperty<T>(initialValue)
 { override fun afterChange(property: KProperty<*>, oldValue: T,
 newValue: T) = afterChange(property, oldValue, newValue)
 override fun beforeChange(property: KProperty<*>, oldValue: T,
 newValue: T) = beforeChange(property, oldValue, newValue)
 }
interface PropertyObserver
{ fun willChange(propertyName: String, newPropertyValue: Any?)
 fun didChange(propertyName: String, oldPropertyValue: Any?) }
```

```
class Observer : PropertyObserver
{ override fun willChange(propertyName: String, newPropertyValue: Any?)
 { if (newPropertyValue is String && newPropertyValue == "test")
 { println("Okay. Look") }
 }
 override fun didChange(propertyName: String, oldPropertyValue: Any?)
 { if (oldPropertyValue is String && oldPropertyValue == "<no name>")
 { println("Sorry about the mess..") }
 }
}
class User(val propertyObserver: PropertyObserver?) {
 var name: String by observable("<no name>",
 { prop, old, new ->
 println("Before change: $old -> $new")
 propertyObserver?.willChange(name, new)
 return@observable true},
 { prop, old, new ->
 propertyObserver?.didChange(name, old)
 println("After change: $old -> $new")
 })
}
fun main(args: Array<String>)
{ val observer = Observer()
 val user = User(observer)
 user.name = "test" }
```

# **Modelul automatului finit State ???**

# Modelul automatului finit - caz de utilizare

```
class CoffeeMachine
{ var state: CoffeeMachineState
 val MAX_BEANS_QUANTITY = 100
 val MAX_WATER_QUANTITY = 100
 var beansQuantity = 0
 var waterQuantity = 0
 val offState = Off(this)
 val noIngredients = NoIngredients(this)
 val ready = Ready(this)
 init { state = offState }
 fun turnOn() = state.turnOn()
 fun fillInBeans(quantity: Int) = state.fillInBeans(quantity)
 fun fillInWater(quantity: Int) = state.fillInWater(quantity)
 fun makeCoffee() = state.makeCoffee()
 fun turnOff() = state.turnOff()
 override fun toString(): String
 { return """COFFEE MACHINE → ${state.simpleName}
 | water quantity : $waterQuantity
 | beans quantity : $beansQuantity
 """.trimMargin() }
abstract class CoffeeMachineState(val coffeeMachine: CoffeeMachine)
{ open fun makeCoffee(): Unit = throw UnsupportedOperationException("Op. not supported")
 open fun fillInBeans(quantity: Int): Unit = throw UnsupportedOperationException("Operation not supported")
 open fun fillInWater(quantity: Int): Unit = throw UnsupportedOperationException("Operation not supported")
 open fun turnOn(): Unit = throw UnsupportedOperationException("Operation not supported")
 fun turnOff(): Unit
 { coffeeMachine.state = coffeeMachine.offState } }
class Off(coffeeMachine: CoffeeMachine) : CoffeeMachineState(coffeeMachine)
{ override fun turnOn()
 { coffeeMachine.state = coffeeMachine.noIngredients
 println("Coffee machine turned on") } }
```

```
class NoIngredients(coffeeMachine: CoffeeMachine) : CoffeeMachineState(coffeeMachine)
{ override fun fillInBeans(quantity: Int)
 { if ((coffeeMachine.beansQuantity + quantity) <=
coffeeMachine.MAX_BEANS_QUANTITY)
 { coffeeMachine.beansQuantity += quantity
 println("Beans filled in")
 if (coffeeMachine.waterQuantity > 0)
 { coffeeMachine.state = coffeeMachine.ready } } }
 override fun fillInWater(quantity: Int)
 { if ((coffeeMachine.waterQuantity + quantity) <=
coffeeMachine.MAX_WATER_QUANTITY)
 { coffeeMachine.waterQuantity += quantity
 println("Water filled in")
 if (coffeeMachine.beansQuantity > 0)
 { coffeeMachine.state = coffeeMachine.ready } } }
class Ready(coffeeMachine: CoffeeMachine) : CoffeeMachineState(coffeeMachine)
{ override fun makeCoffee()
 { coffeeMachine.beansQuantity-
 coffeeMachine.waterQuantity-
 println("Making coffee... DONE")
 if (coffeeMachine.beansQuantity == 0 || coffeeMachine.waterQuantity == 0)
 { coffeeMachine.state = coffeeMachine.noIngredients } } }
fun main(args: Array<String>)
{ val coffeeMachine = CoffeeMachine()
 coffeeMachine.turnOn()
 println(coffeeMachine)
 coffeeMachine.fillInBeans(2)
 println(coffeeMachine)
 coffeeMachine.fillInWater(2)
 println(coffeeMachine)
 coffeeMachine.makeCoffee()
 println(coffeeMachine)
 coffeeMachine.makeCoffee()
 println(coffeeMachine)
 coffeeMachine.turnOff()
 println(coffeeMachine) }
```

**Modelul Vizitator? pa...pa...**

# Exemplu de tratare

```
abstract class ResultOrErrorVisitor<T>
{
 abstract T visit(Result<T> result);
 abstract T visit(Error<T> error);
}

interface ResultOrError<T>
{
 T accept(ResultOrErrorVisitor<T> visitor);
}

class Result<T> implements ResultOrError<T>
{
 public final T result;
 Result(T result)
 {
 this.result = result;
 }
 @Override
 public T accept(ResultOrErrorVisitor<T> visitor)
 {
 return visitor.visit(this);
 }
}

class Error<T> implements ResultOrError<T>
{
 public final Throwable error;
 Error(Throwable error)
 {
 this.error = error;
 }
 @Override
 public T accept(ResultOrErrorVisitor<T> visitor)
 {
 return visitor.visit(this);
 }
}
```

În Kotlin

```
sealed class ResultOrError<out T>
{
 data class Result<out T>(val result: T) : ResultOrError<T>()
 data class Error<out T>(val error: Throwable) : ResultOrError<T>()
```

# Modelul comandă

```
data class User(val name: String) {
}

interface Command
{
 fun execute(user: User)
}

class AddUser : Command
{
 override fun execute(user: User)
 { println("Adding a new user with name: "+
 user.name) }
}

class DeleteUser : Command
{
 override fun execute(user: User)
 { println("Deleting user with name: "+user.name) }
}
```

```
class EditUser : Command
{
 override fun execute(user: User)
 { println("Editing user with name: "+user.name) }
}

fun main(args: Array<String>)
{
 var user = User('Kotlin')

 var add = AddUser()
 add.execute(user)

 var edit = EditUser();
 edit.execute(user)

 var delete = DeleteUser()
 delete.execute(user)
}
```

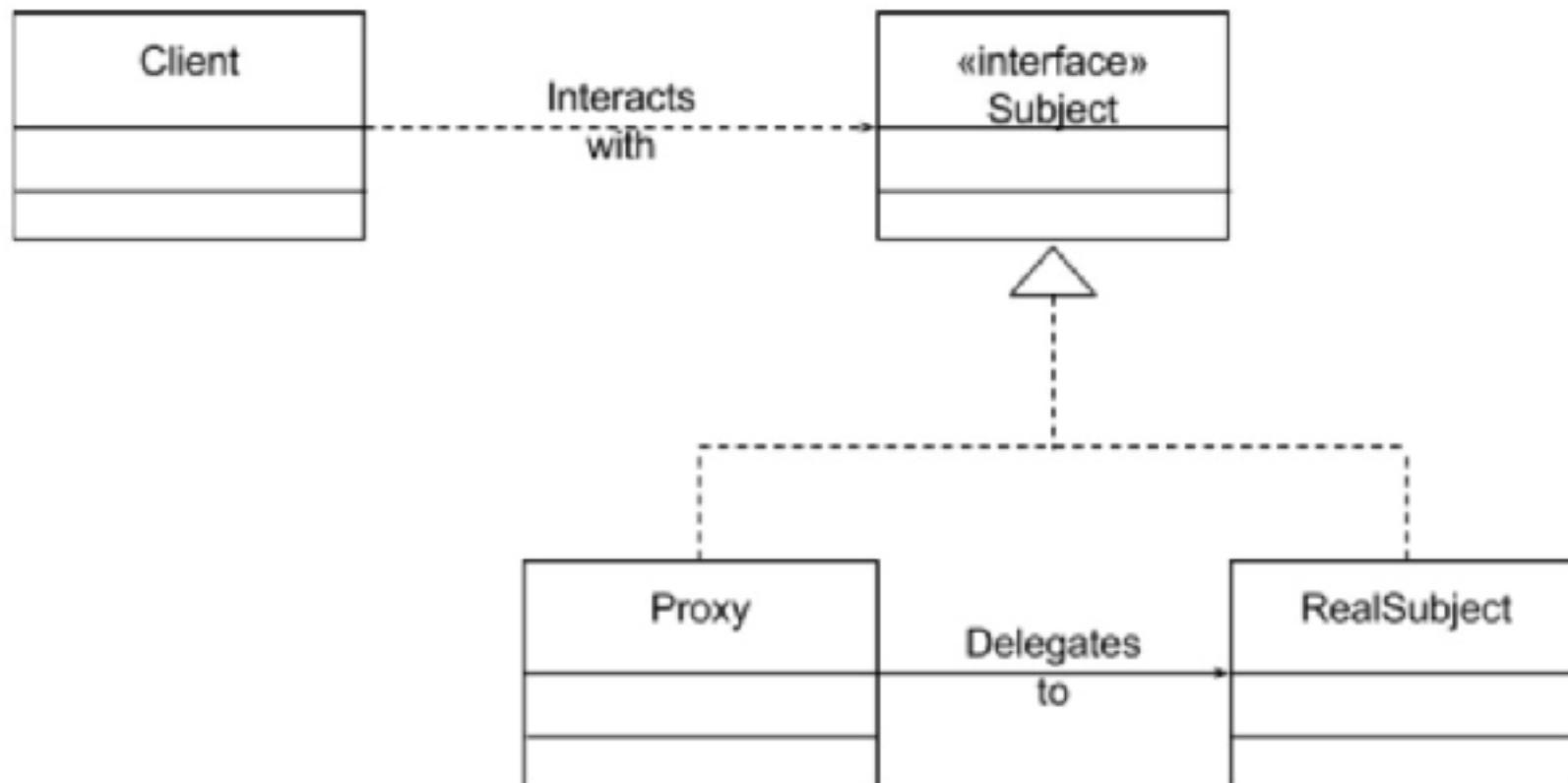
# modelul restaurare/reamintire (latină: memento)

```
import kotlin.collections.ArrayList
data class Memento(val state: String)
class Originator
{ //acest String este doar pentru exemplu in realitate
//acesta ar fi inlocuit de obiectul al carui stare
//trebuie salvata si apoi restaurata
 var state: String? = null
 fun createMemento(): Memento
 { return Memento(state!!) }
 fun setMemento(memento: Memento)
 { state = memento.state }
}

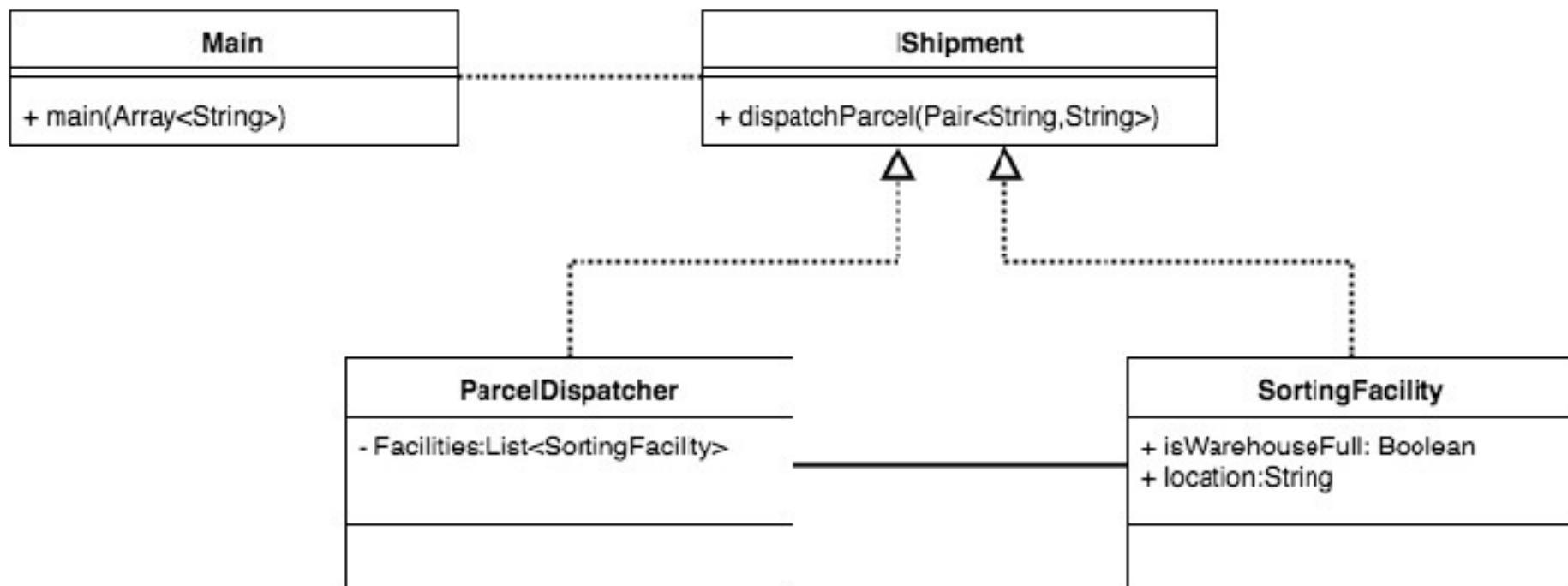
class Caretaker
{ private val statesList = ArrayList<Memento>()
 fun addMemento(m: Memento)
 { statesList.add(m) }
 fun getMemento(index: Int): Memento
 { return statesList.get(index) }
}
```

```
fun main(args: Array<String>)
{ val originator = Originator()
 originator.state = "Ironman"
 var memento = originator.createMemento()
 val caretaker = Caretaker()
 caretaker.addMemento(memento)
 originator.state = "Captain America"
 originator.state = "Hulk"
 memento = originator.createMemento()
 caretaker.addMemento(memento)
 originator.state = "Thor"
 println("Stare curenta Origine:" + originator.state!!)
 println("Restaurare Origine...")
 memento = caretaker.getMemento(1)
 originator.setMemento(memento)
 println("Stare curenta Origine: " + originator.state!!)
 println("Din nou Restaurare...")
 memento = caretaker.getMemento(0)
 originator.setMemento(memento)
 println("Stare curenta Origine:" + originator.state!!)
}
```

# Intermediar - forma generală



# Intermediar - caz de utilizare



# Intermediar - caz de utilizare - implementare

```
interface IShipment
{
 // pachetul este reprezentat de o pereche
 // unde primul String - continutul pachetului iar al doilea locația
 // (conținut to locație)
 fun dispatchParcel(parcel:Pair<String,String>)
}

class SortingFacility(val location:String,var
isWarehouseFull:Boolean) : IShipment
{
 override fun dispatchParcel(parcel: Pair<String, String>)
 { println("${location} facility doing dispatching business...") }
}

class ParcelDispatcher : IShipment
{
 // locația a fost aleasă ca string din motive academice!
 private var facility = listOf<SortingFacility>
 (
 SortingFacility("North",true),
 SortingFacility("North West",false),
 SortingFacility("South",false),
 SortingFacility('West',true),
 SortingFacility('East',false)
)
}

override fun dispatchParcel(parcel: Pair<String, String>)
{
 val facilityNearTpParcelLocation = facility.filter
 { it.location.contains(parcel.second,true)
 && !it.isWarehouseFull }.first()
 facilityNearTpParcelLocation.dispatchParcel(parcel)
}

fun main(args:Array<String>)
{
 var pachet = "SmartPhone" to "Nord"
 var parcelDispatcher = ParcelDispatcher()
 parcelDispatcher.dispatchParcel(pachet)
}
```

# Modelul iterator

```
class Schita(val name: String)
class Schite(val Schite: MutableList<Schita> = mutableListOf()) : Iterable<Schita>
{
 override fun iterator(): Iterator<Schita> = SchiteIterator(Schite)
}
class SchiteIterator(val Schite: MutableList<Schita> = mutableListOf(), var current: Int = 0) :
Iterator<Schita>
{
 override fun hasNext(): Boolean = Schite.size > current
 override fun next(): Schita
 {
 val Schita = Schite[current]
 current++
 return Schita
 }
}
fun main(args: Array<String>)
{
 val Schite = Schite(mutableListOf(Schita("Test1"), Schita("Test2")))
 Schite.forEach { println(it.name) }
}
```

# Modelul Strategie

```
interface BookingStrategy
{
 val fare: Double
}

class CarBookingStrategy : BookingStrategy
{
 override val fare: Double = 12.5

 override fun toString(): String
 { return "CarBookingStrategy" }
}

class TrainBookingStrategy : BookingStrategy
{
 override val fare: Double = 8.5

 override fun toString(): String
 { return "TrainBookingStrategy" }
}
```

```
class Customer(var bookingStrategy: BookingStrategy)
{
 fun calculateFare(numOfPassangeres: Int): Double
 {
 val fare = numOfPassangeres *
 bookingStrategy.fare
 println("Calculating fares using " +
 bookingStrategy)
 return fare
 }
}

fun main(args: Array<String>)
{
 //strategia de rezolvare a rezervarii pentru o masina
 val cust = Customer(CarBookingStrategy())
 var fare = cust.calculateFare(5)
 println(fare)

 //strategia de rezolvare a rezervarii pentru tren
 cust.bookingStrategy = TrainBookingStrategy()
 fare = cust.calculateFare(5)
 println(fare)
}
```

# **Modelul Mediator**

-

# Model Mediator - caz de utilizare

```
interface Command
 { fun land() }

class Flight(private val atcMediator: IATCMediator) : Command
{ override fun land()
 { if (atcMediator.isLandingOk)
 { println("Landing done....")
 atcMediator.setLandingStatus(true)
 } else
 println("Will wait to land....") }
 fun getReady()
 { println("Getting ready...") } }

class Runway(private val atcMediator: IATCMediator) :
Command
{ init { atcMediator.setLandingStatus(true) }
 override fun land()
 { println("Landing permission granted...")
 atcMediator.setLandingStatus(true) } }

interface IATCMediator
{ val isLandingOk: Boolean
 fun registerRunway(runway: Runway)
 fun registerFlight(flight: Flight)
 fun setLandingStatus(status: Boolean) }

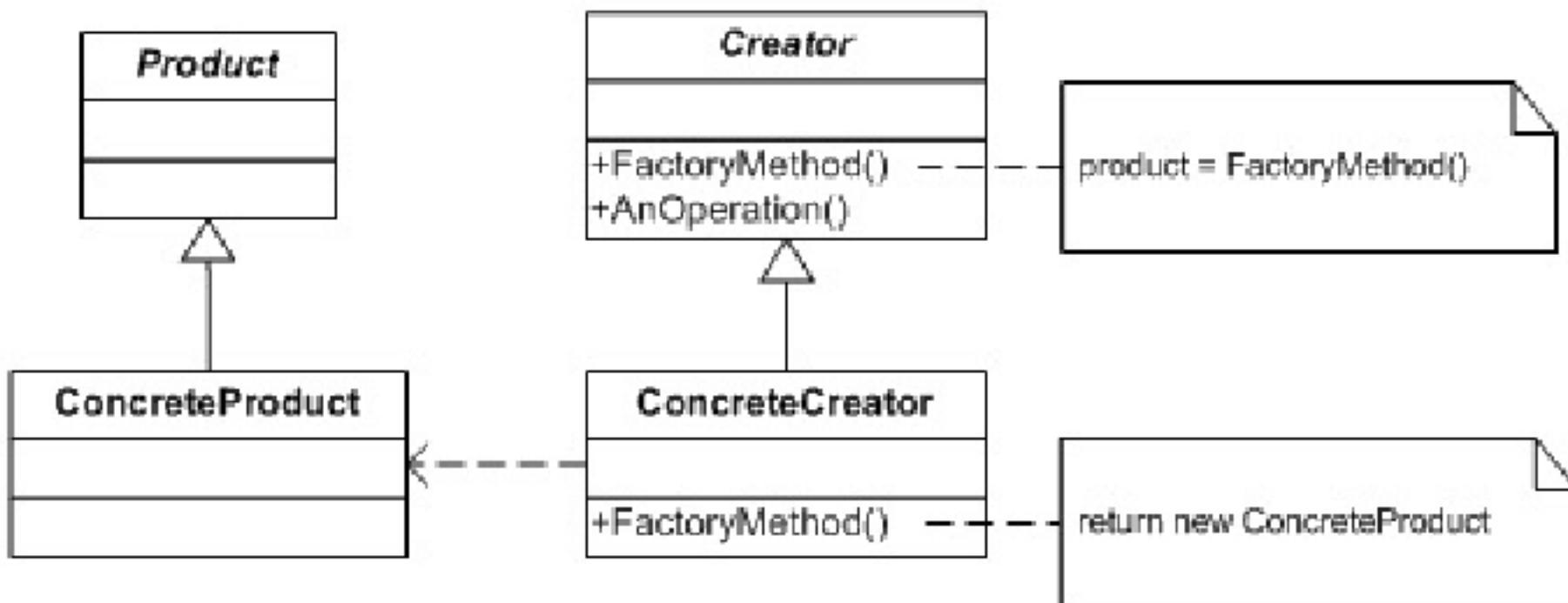
class ATCMediator : IATCMediator
{ private var flight: Flight? = null
 private var runway: Runway? = null
 override var isLandingOk: Boolean = false
 override fun registerRunway(runway: Runway)
 { this.runway = runway }
 override fun registerFlight(flight: Flight)
 { this.flight = flight }
 override fun setLandingStatus(status: Boolean)
 { isLandingOk = status } }

fun main(args: Array<String>)
{
 val atcMediator = ATCMediator()
 val sparrow101 = Flight(atcMediator)
 val mainRunway = Runway(atcMediator)
 atcMediator.registerFlight(sparrow101)
 atcMediator.registerRunway(mainRunway)
 sparrow101.getReady()
 mainRunway.land()
 sparrow101.land()
}
```

# Paradigma Modelelor de Proiectare

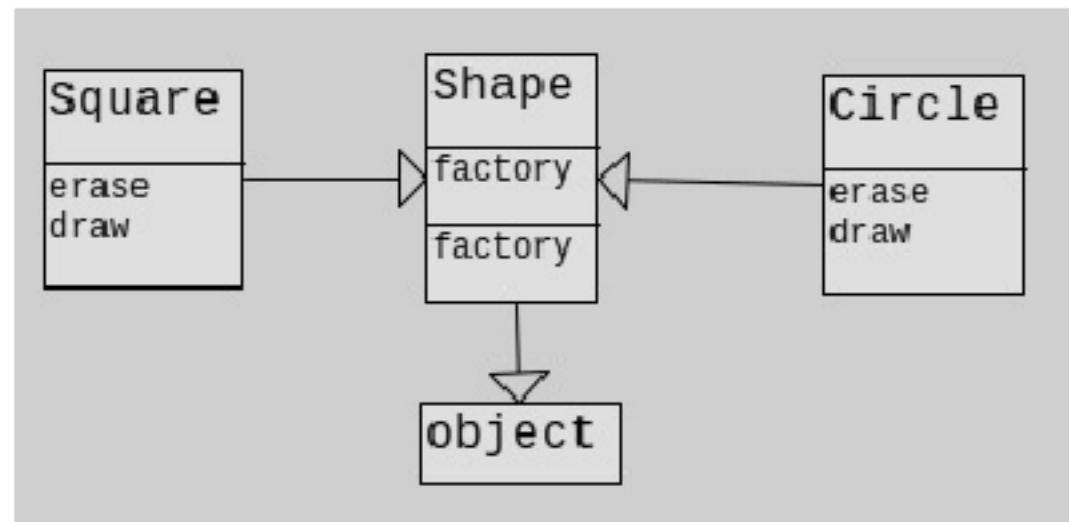
Cursul nr. 9  
Mihai Zaharia

# Modelul Fabrică de obiecte

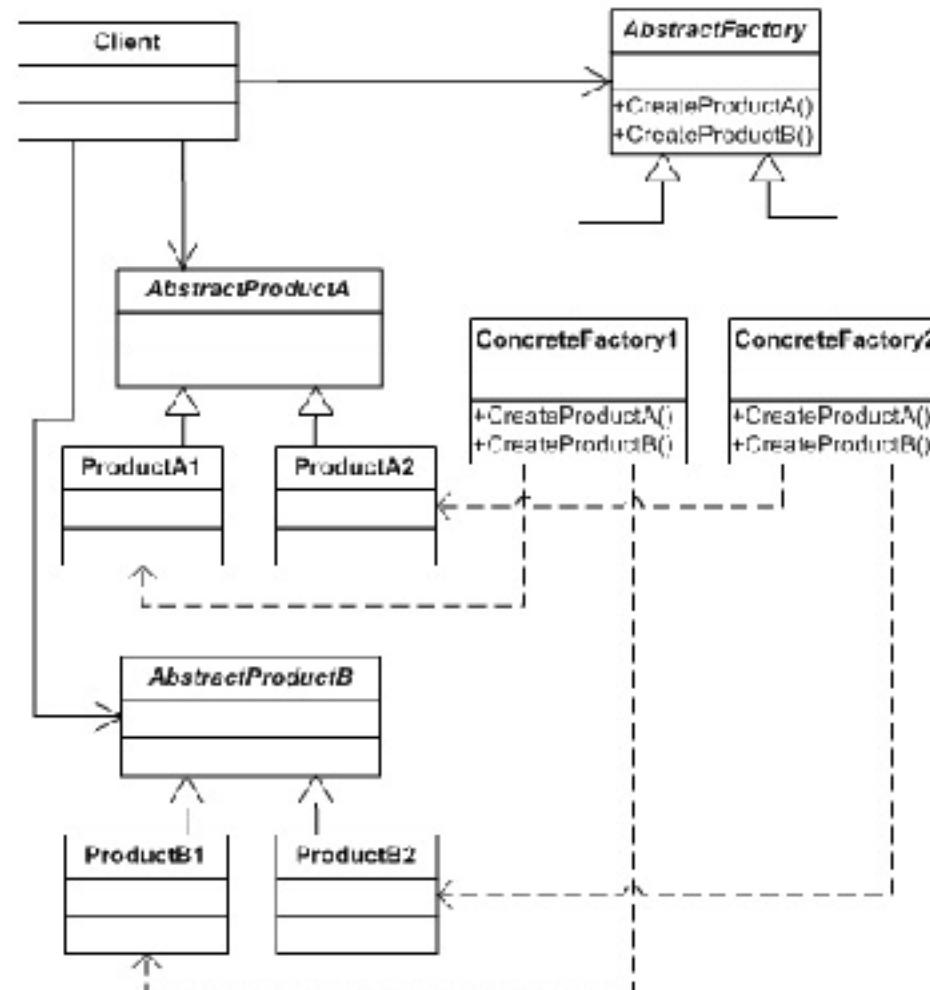


# Modelul Fabrică de obiecte - caz de utilizare

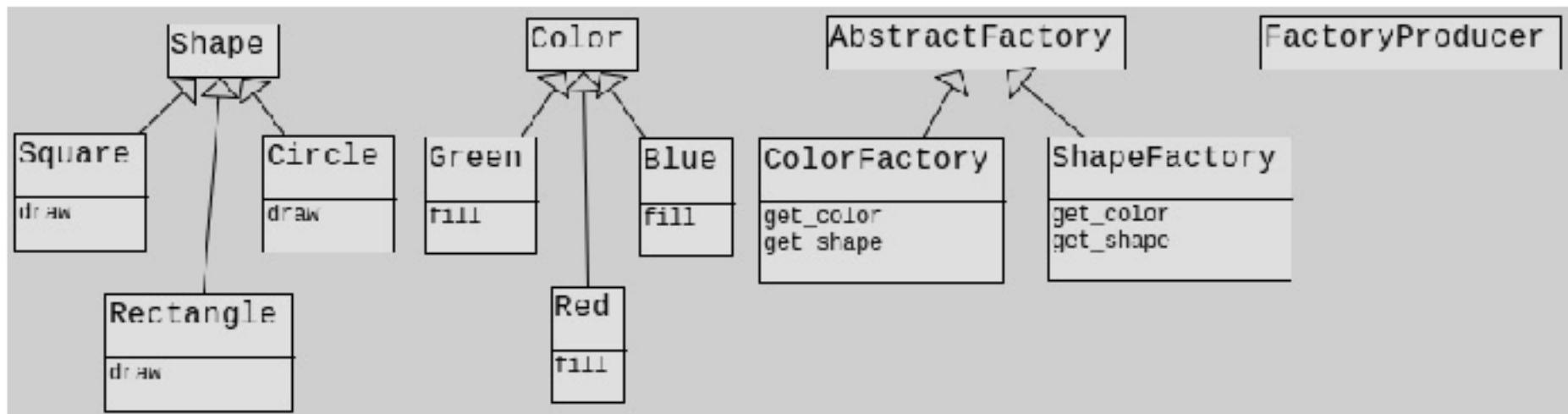
```
class Shape(object):
 def factory(type): #return eval(type + "()")
 if type == "Circle": return Circle()
 if type == "Square": return Square()
 assert 0, "Bad shape creation: " + type
 factory = staticmethod(factory)
class Circle(Shape):
 def draw(self): print("Circle.draw")
 def erase(self): print("Circle.erase")
class Square(Shape):
 def draw(self): print("Square.draw")
 def erase(self): print("Square.erase")
se genereaza numele formelor
def shapeNameGen(n):
 types = Shape.__subclasses__()
 for i in range(n):
 yield random.choice(types).__name__
shapes = [Shape.factory(i) for i in shapeNameGen(7)]
for shape in shapes:
 shape.draw()
 shape.erase()
```



# Model Fabrica abstractă



# Modelul Fabrică de obiecte - clasic

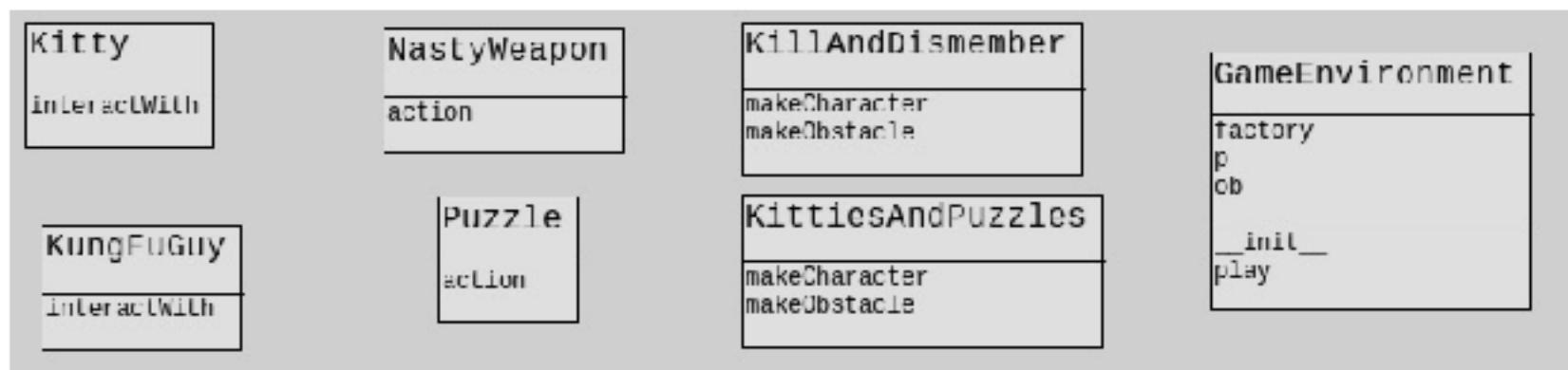
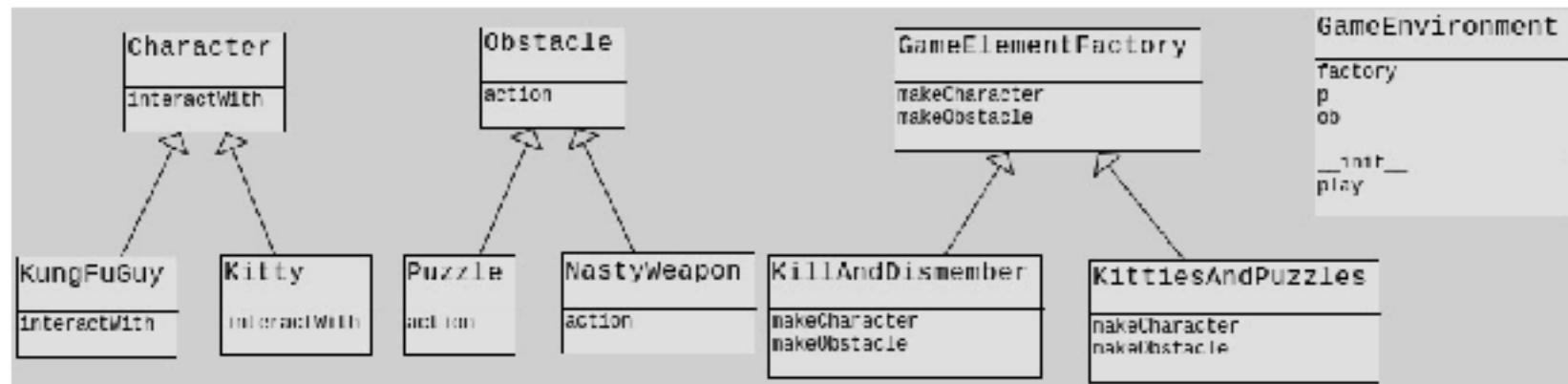


# Și implementarea

```
import abc
class Shape(metaclass=abc.ABCMeta): #interfata pentru forme
 @abc.abstractmethod
 def draw(self):
 pass
class Color(metaclass=abc.ABCMeta):#interfata pentru culori
 @abc.abstractmethod
 def fill(self):
 pass
class AbstractFactory(metaclass=abc.ABCMeta):#creare clasa abstracta pt obtinere obj
 @abc.abstractmethod
 def get_color(self):
 pass
 @abc.abstractmethod
 def get_shape(self):
 pass
class Rectangle(Shape):
 def draw(self):
 print("Inside Rectangle::draw() method.")
class Square(Shape):
 def draw(self):
 print("Inside Square::draw() method.")
class Circle(Shape):
 def draw(self):
 print("Inside Circle::draw() method.")
class Red(Color):
 def fill(self):
 print("Inside Red::fill() method.")
class Green(Color):
 def fill(self):
 print("Inside Green::fill() method.")
class Blue(Color):
 def fill(self):
 print("Inside Blue::fill() method.")
```

```
crearea generator fabrici
class FactoryProducer:
 @staticmethod
 def get_factory(choice):
 if choice == "SHAPE":
 return ShapeFactory()
 elif choice == "COLOR":
 return ColorFactory()
 return None
if __name__ == '__main__':
 shape_factory = FactoryProducer.get_factory("SHAPE")
 shape1 = shape_factory.get_shape("CIRCLE");
 shape1.draw()
 shape2 = shape_factory.get_shape("RECTANGLE");
 shape2.draw()
 shape3 = shape_factory.get_shape("SQUARE");
 shape3.draw()
 color_factory = FactoryProducer.get_factory("COLOR");
 color1 = color_factory.get_color("RED");
 color1.fill()
 color2 = color_factory.get_color("GREEN");
 color2.fill()
 color3 = color_factory.get_color("BLUE");
 color3.fill()
```

# Să reanalizăm un pic fabrica de fabrici



# Modelul Fabrica abstractă - OOP vs Structurat

```
class Obstacle:# versiunea generală cf oop
 def action(self): pass
class Character:
 def interactWith(self, obstacle): pass
class Kitty(Character):
 def interactWith(self, obstacle):
 print("Kitty has encountered a",
 obstacle.action())
class KungFuGuy(Character):
 def interactWith(self, obstacle):
 print("KungFuGuy now battles a",
 obstacle.action())
class Puzzle(Obstacle):
 def action(self):
 print("Puzzle")
class NastyWeapon(Obstacle):
 def action(self):
 print("NastyWeapon")
class GameElementFactory:# fabrica abstractă
 def makeCharacter(self): pass
 def makeObstacle(self): pass
class KittiesAndPuzzles(GameElementFactory):# fabrică concrete
 def makeCharacter(self): return Kitty()
 def makeObstacle(self): return Puzzle()
class KillAndDismember(GameElementFactory):
 def makeCharacter(self): return KungFuGuy()
 def makeObstacle(self): return NastyWeapon()
class GameEnvironment:
 def __init__(self, factory):
 self.factory = factory
 self.p = factory.makeCharacter()
 self.ob = factory.makeObstacle()
 def play(self):
 self.p.interactWith(self.ob)
g1 = GameEnvironment(KittiesAndPuzzles())
g2 = GameEnvironment(KillAndDismember())
g1.play()
g2.play()

class Kitty:# versiunea front end-ului
 def interactWith(self, obstacle):
 print("Kitty has encountered a",
 obstacle.action())
class KungFuGuy:
 def interactWith(self, obstacle):
 print("KungFuGuy now battles a",
 obstacle.action())
class Puzzle:
 def action(self): print("Puzzle")
class NastyWeapon:
 def action(self): print("NastyWeapon")
class KittiesAndPuzzles:# fabrică concrete
 def makeCharacter(self): return Kitty()
 def makeObstacle(self): return Puzzle()
class KillAndDismember:
 def makeCharacter(self): return KungFuGuy()
 def makeObstacle(self): return NastyWeapon()
class GameEnvironment:
 def __init__(self, factory):
 self.factory = factory
 self.p = factory.makeCharacter()
 self.ob = factory.makeObstacle()
 def play(self):
 self.p.interactWith(self.ob)
g1 = GameEnvironment(KittiesAndPuzzles())
g2 = GameEnvironment(KillAndDismember())
g1.play()
g2.play()
```

# Modelul burlacului

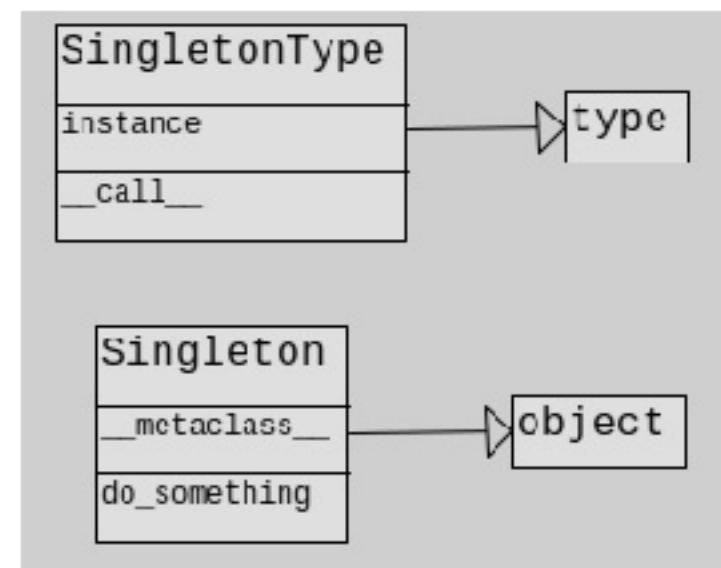
```
class SingletonType(type):
 instance = None

 def __call__(cls, *args, **kw):
 if not cls.instance:
 cls.instance = super(SingletonType, cls).__call__(*args, **kw)
 return cls.instance

class Singleton(object):
 __metaclass__ = SingletonType

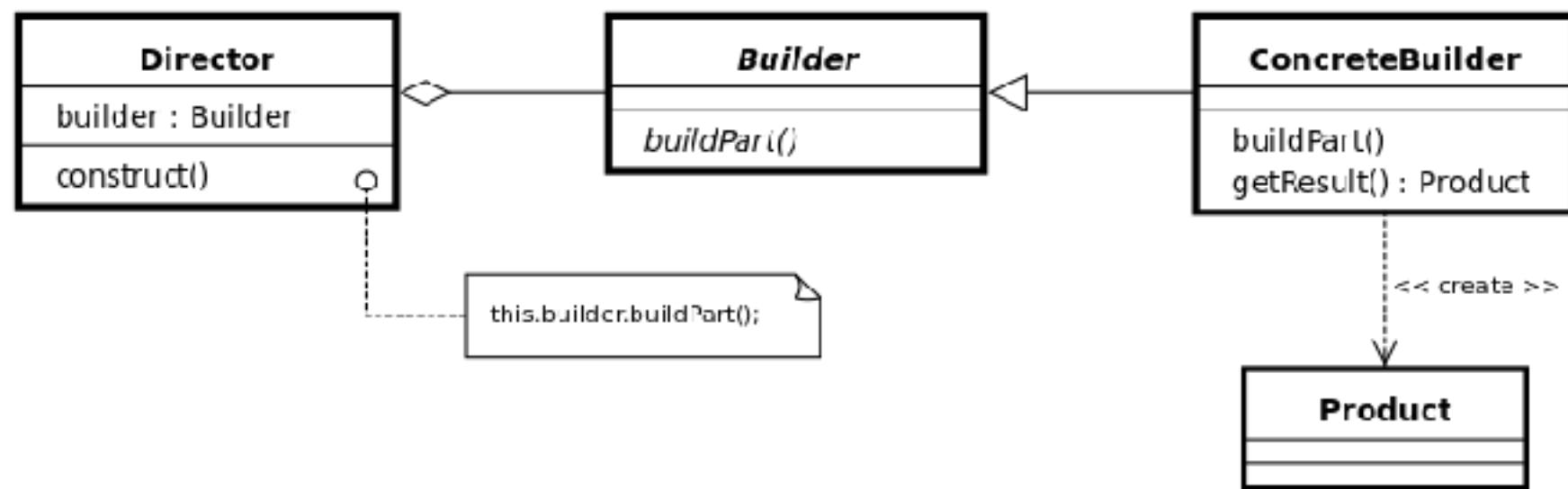
 def do_something(self):
 print('Singleton')

s = Singleton()
s.do_something()
```

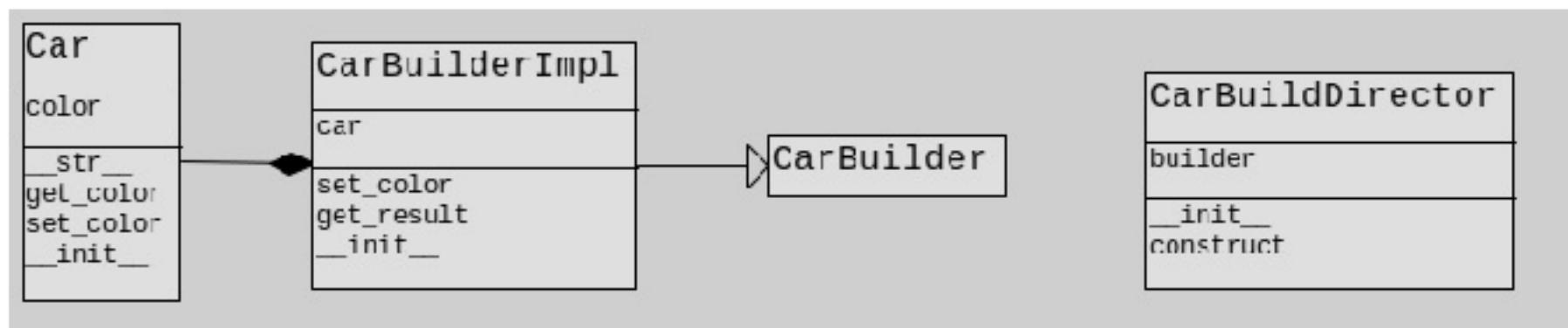


# Modelul constructor

# Modelul constructor



# Modelul constructor - caz de utilizare



# Model constructor - implementare concretă

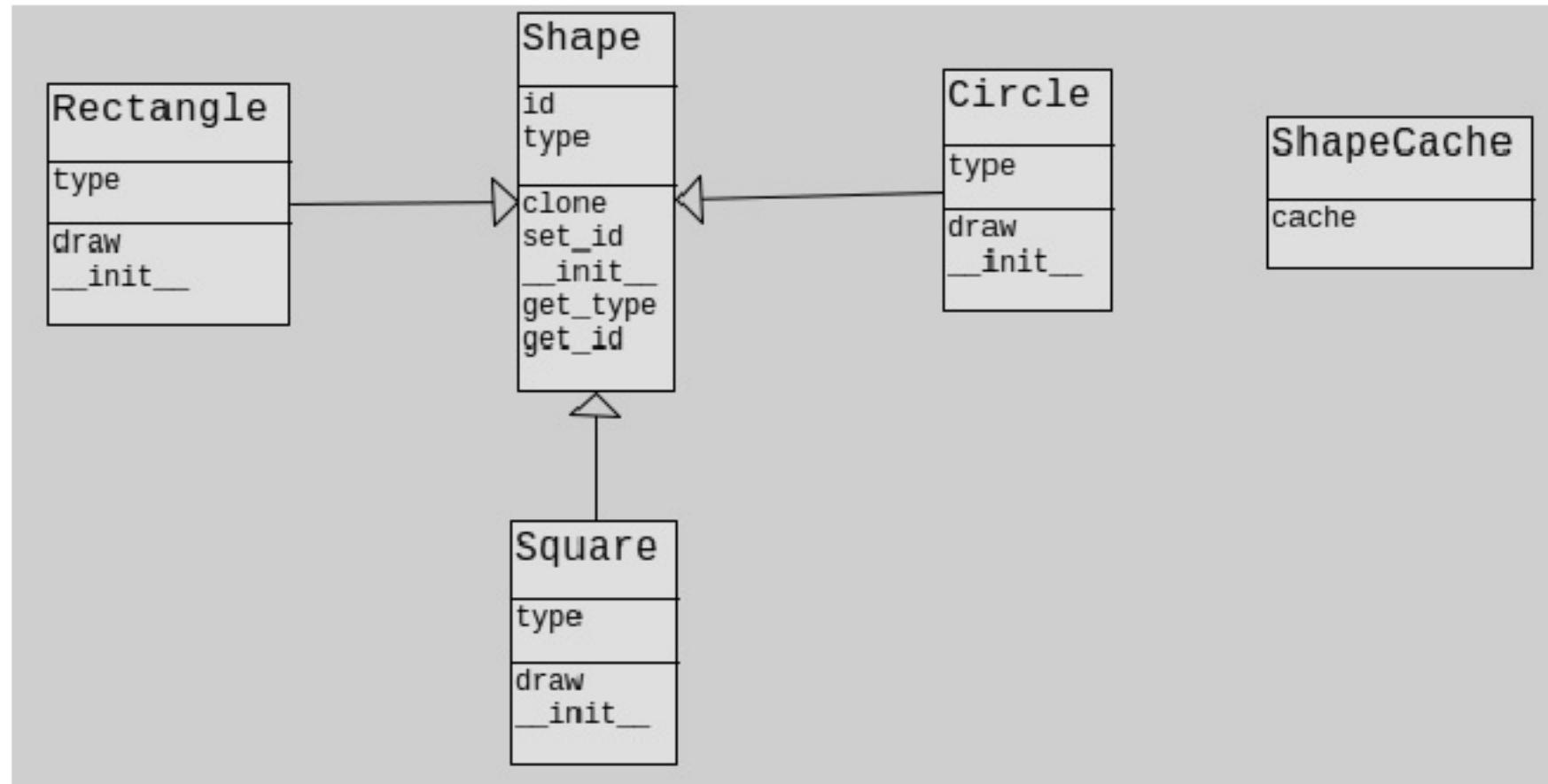
```
import abc

class Car:#produsul creat de Creator
 def __init__(self):
 self.color = None
 def get_color(self):
 return self.color
 def set_color(self, color):
 self.color = color
 def __str__(self):
 return "Car [color={0}]".format(self.color)
class CarBuilder(metaclass=abc.ABCMeta): #abstractia Creator
 @abc.abstractmethod
 def set_color(self, color):
 pass
 @abc.abstractmethod
 def get_result(self):
 pass
class CarBuilderImpl(CarBuilder):
 def __init__(self):
 self.car = Car()
 def set_color(self, color):
 self.car.set_color(color)
 def get_result(self):
 return self.car

class CarBuildDirector:
 def __init__(self, builder):
 self.builder = builder
 def construct(self):
 self.builder.set_color("Red")
 return self.builder.get_result()

if __name__ == '__main__':
 builder = CarBuilderImpl()
 carBuildDirector = CarBuildDirector(builder)
 print(carBuildDirector.construct())
```

# Modelul prototip



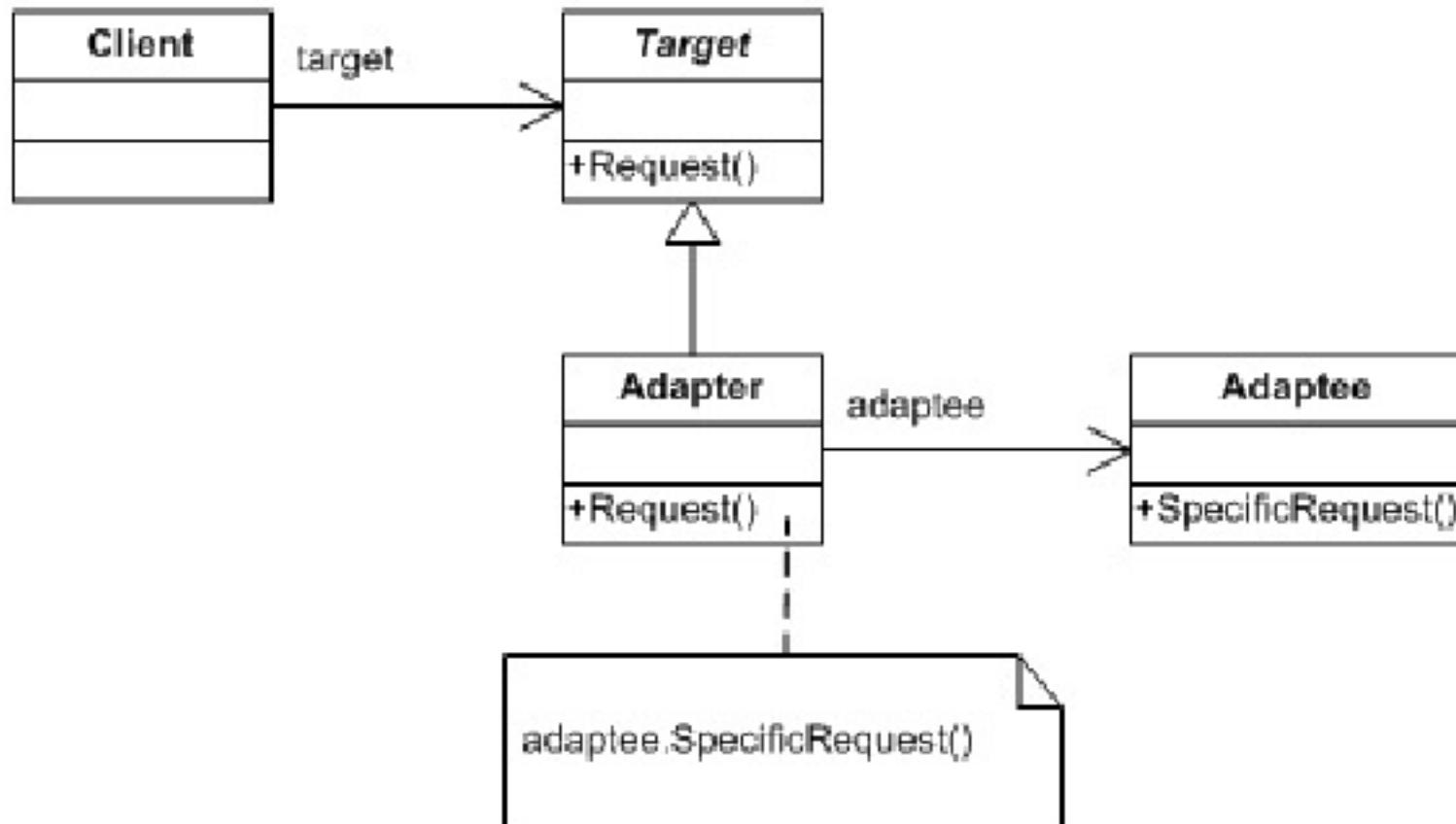
# Model protip - implementare de caz

```
import abc
import copy
class Shape(metaclass=abc.ABCMeta):
 def __init__(self):
 self.id = None
 self.type = None
 @abc.abstractmethod
 def draw(self):
 pass
 def get_type(self):
 return self.type
 def get_id(self):
 return self.id
 def set_id(self, sid):
 self.id = sid
 def clone(self):
 return copy.deepcopy(self)
class Rectangle(Shape):
 def __init__(self):
 super().__init__()
 self.type = "Rectangle"
 def draw(self):
 print("Inside Rectangle:draw() method.")
class Square(Shape):
 def __init__(self):
 super().__init__()
 self.type = "Square"
 def draw(self):
 print("Inside Square::draw() method.")
```

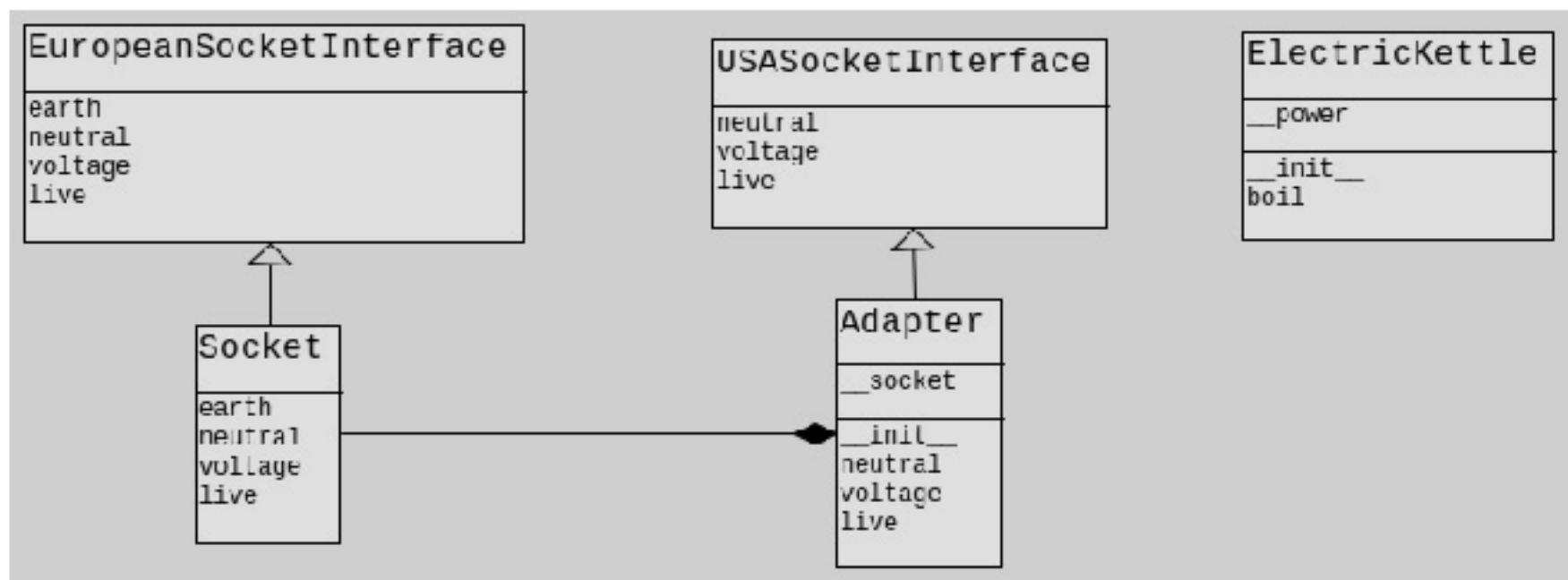
```
class Circle(Shape):
 def __init__(self):
 super().__init__()
 self.type = "Circle"
 def draw(self):
 print("Inside Circle::draw() method.")
class ShapeCache:
 cache = {}
 @staticmethod
 def get_shape(sid):
 shape = ShapeCache.cache.get(sid, None)
 return shape.clone()
 @staticmethod
 def load():
 circle = Circle()
 circle.set_id("1")
 ShapeCache.cache[circle.get_id()] = circle
 square = Square()
 square.set_id("2")
 ShapeCache.cache[square.get_id()] = square
 rectangle = Rectangle()
 rectangle.set_id("3")
 ShapeCache.cache[rectangle.get_id()] = rectangle
 if __name__ == '__main__':
 ShapeCache.load()
 circle = ShapeCache.get_shape("1")
 print(circle.get_type())
 square = ShapeCache.get_shape("2")
 print(square.get_type())
 rectangle = ShapeCache.get_shape("3")
 print(rectangle.get_type())
```

# Modele structurale

# Modelul Adaptor



# Model Adaptor - caz de utilizare



# Model Adaptor - implementare

```
class EuropeanSocketInterface:
 def voltage(self): pass
 def live(self): pass
 def neutral(self): pass
 def earth(self): pass

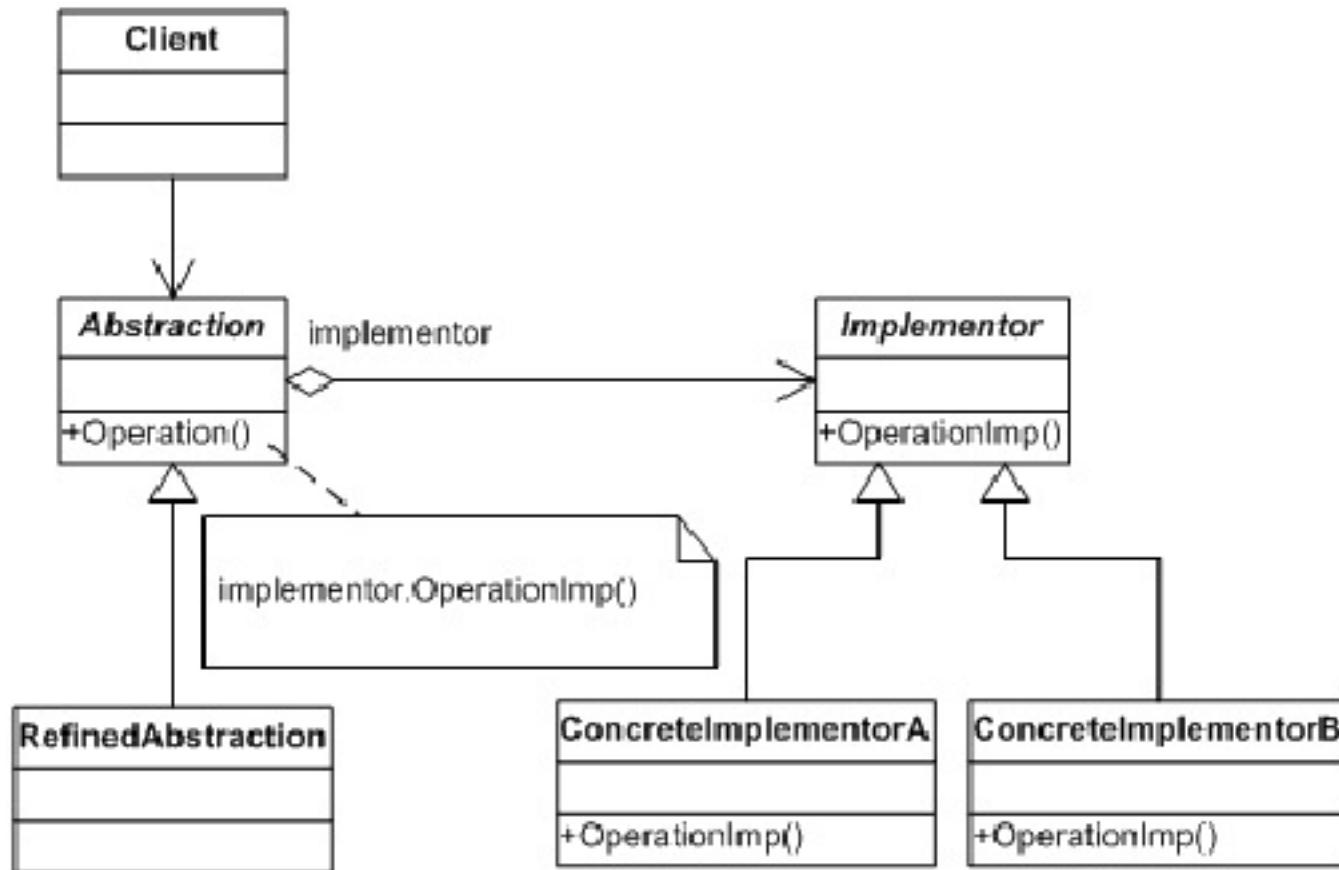
class Socket(EuropeanSocketInterface):# Adaptee
 def voltage(self):
 return 230
 def live(self):
 return 1
 def neutral(self):
 return -1
 def earth(self):
 return 0

class USASocketInterface:#interfata tinta
 def voltage(self): pass
 def live(self): pass
 def neutral(self): pass

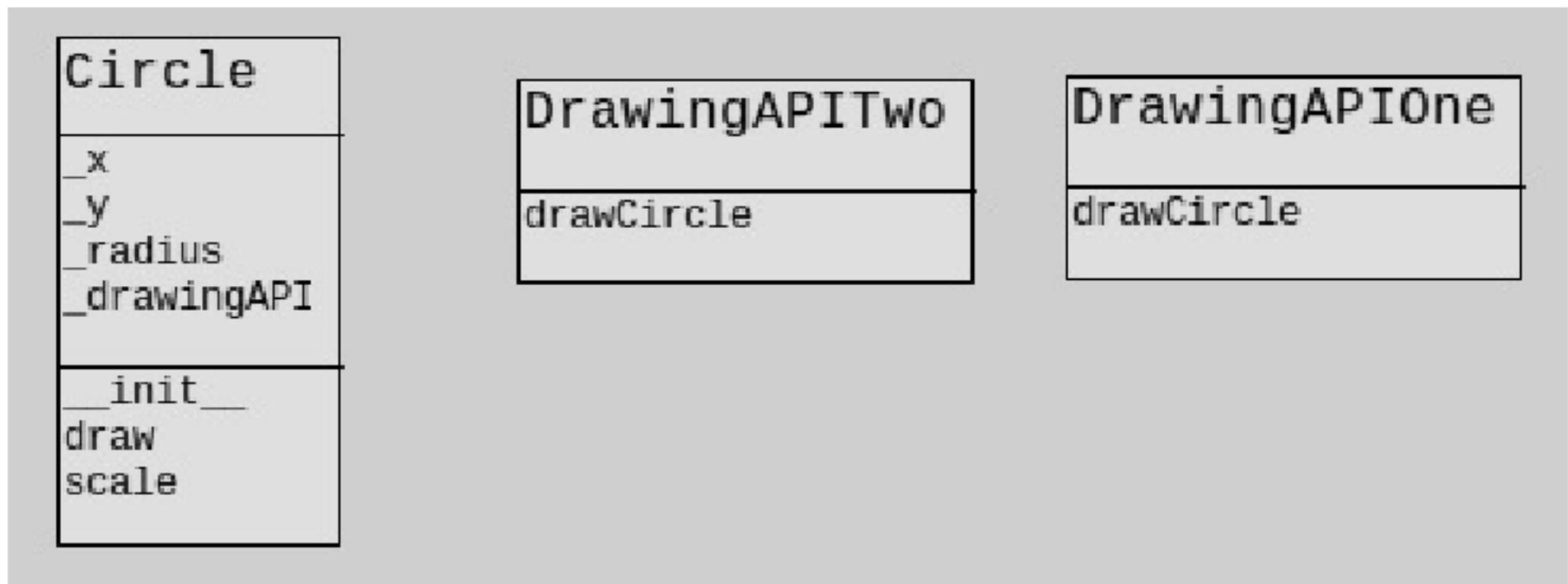
class Adapter(USASocketInterface):# The Adapter
 __socket = None
 def __init__(self, socket):
 self.__socket = socket
 def voltage(self):
 return 110
 def live(self):
 return self.__socket.live()
 def neutral(self):
 return self.__socket.neutral()
```

```
class ElectricKettle:# Client
 __power = None
 def __init__(self, power):
 self.__power = power
 def boil(self):
 if self.__power.voltage() > 110:
 print("Kettle on fire!")
 else:
 if self.__power.live() == 1 and \
 self.__power.neutral() == -1:
 print("Coffee time!")
 else:
 print("No power.")
 def main():
 # bagam in priza cu adaptor
 socket = Socket()
 adapter = Adapter(socket)
 kettle = ElectricKettle(adapter)
 # facem cafea
 kettle.boil()
 return 0
```

# Modelul Pod



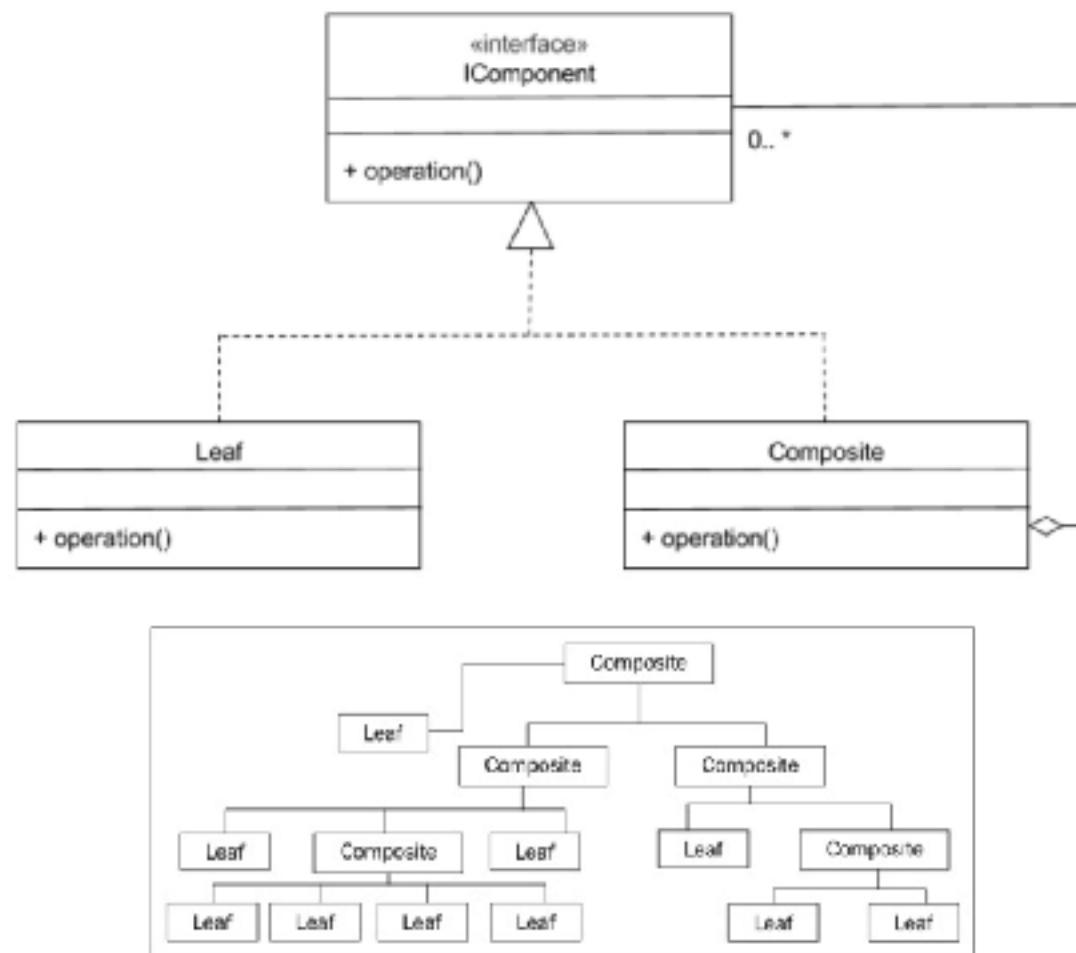
# Modelul Pod - caz de utilizare



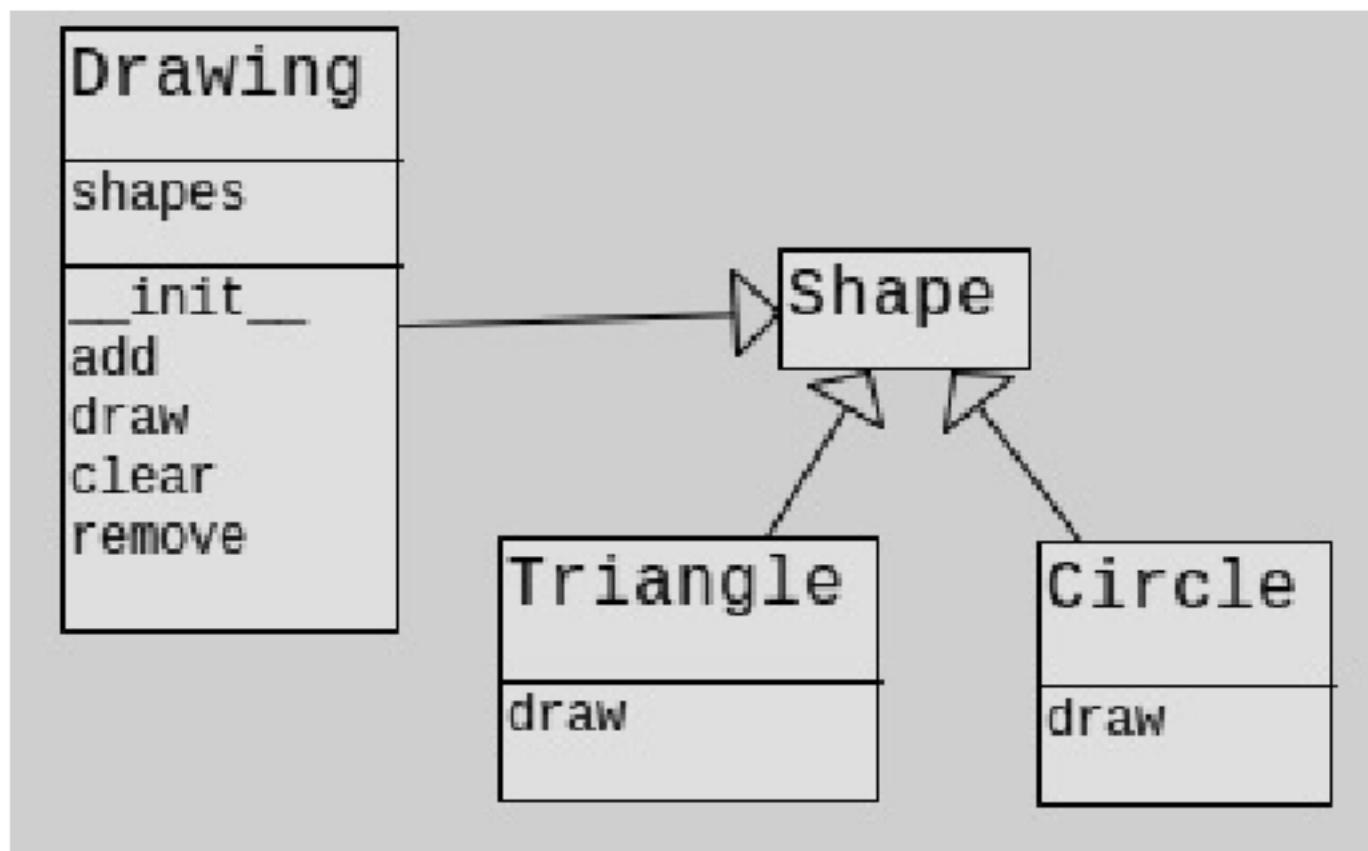
# Modelul Pod - implementare

```
class DrawingAPIOne:# concret
 def drawCircle(self, x, y, radius):
 print("API 1 deseneaza un cerc la ({}, {}) cu raza {}".format(x, y, radius))
class DrawingAPITwo:#concret
 def drawCircle(self, x, y, radius):
 print("API 2 deseneaza un cerc la ({}, {}) cu raza {}".format(x, y, radius))
class Circle: #abstract
 def __init__(self, x, y, radius, drawingAPI):
 self._x = x
 self._y = y
 self._radius = radius
 self._drawingAPI = drawingAPI
 def draw(self):#apelul unei implementari concrete specifice
 self._drawingAPI.drawCircle(self._x, self._y, self._radius)
 def scale(self, percent):
 self._radius *= percent
circle1 = Circle(0, 0, 2, DrawingAPIOne())
circle1.draw()
circle2 = Circle(1, 3, 3, DrawingAPITwo())
circle2.draw()
```

# Model Compus - forma generală



## Model Compus - caz de utilizare



# Compus - caz de utilizare - implementare

```
import abc
class Shape(metaclass=abc.ABCMeta):
 @ abc.abstractmethod
 def draw(self, color):
 pass
class Triangle(Shape):
 def draw(self, color):
 print("Desenez un triunghi cu culoarea " + color)
class Circle(Shape):
 def draw(self, color):
 print("Desenez un cerc cu culoarea " + color)
class Drawing(Shape):
 def __init__(self):
 self.shapes = []
 def draw(self, color):
 for sh in self.shapes:
 sh.draw(color)
 def add(self, sh):
 self.shapes.append(sh)
 def remove(self, sh):
 self.shapes.remove(sh)
 def clear(self):
 print("Sterg toate formele din desen")
 self.shapes = []
```

```
if __name__ == '__main__':
 tri1 = Triangle()
 tri2 = Triangle()
 cir = Circle()

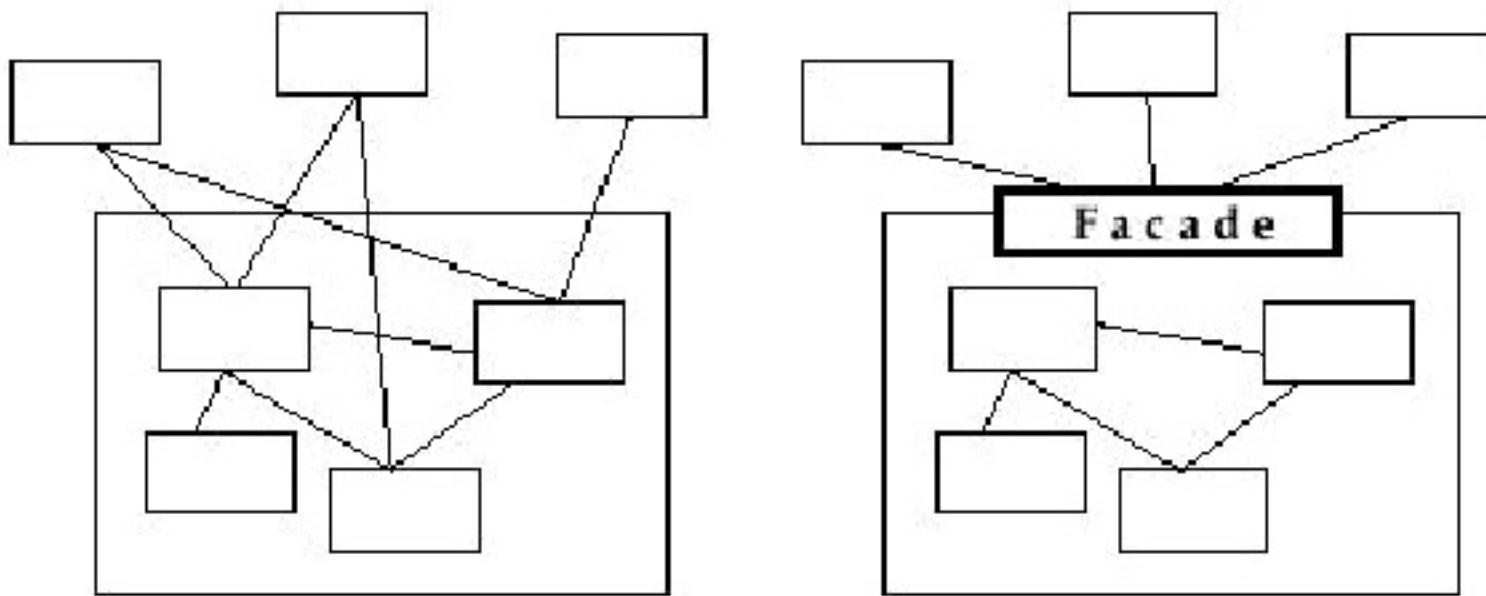
 drawing = Drawing()
 drawing.add(tri1)
 drawing.add(tri2)
 drawing.add(cir)

 drawing.draw("rosu")

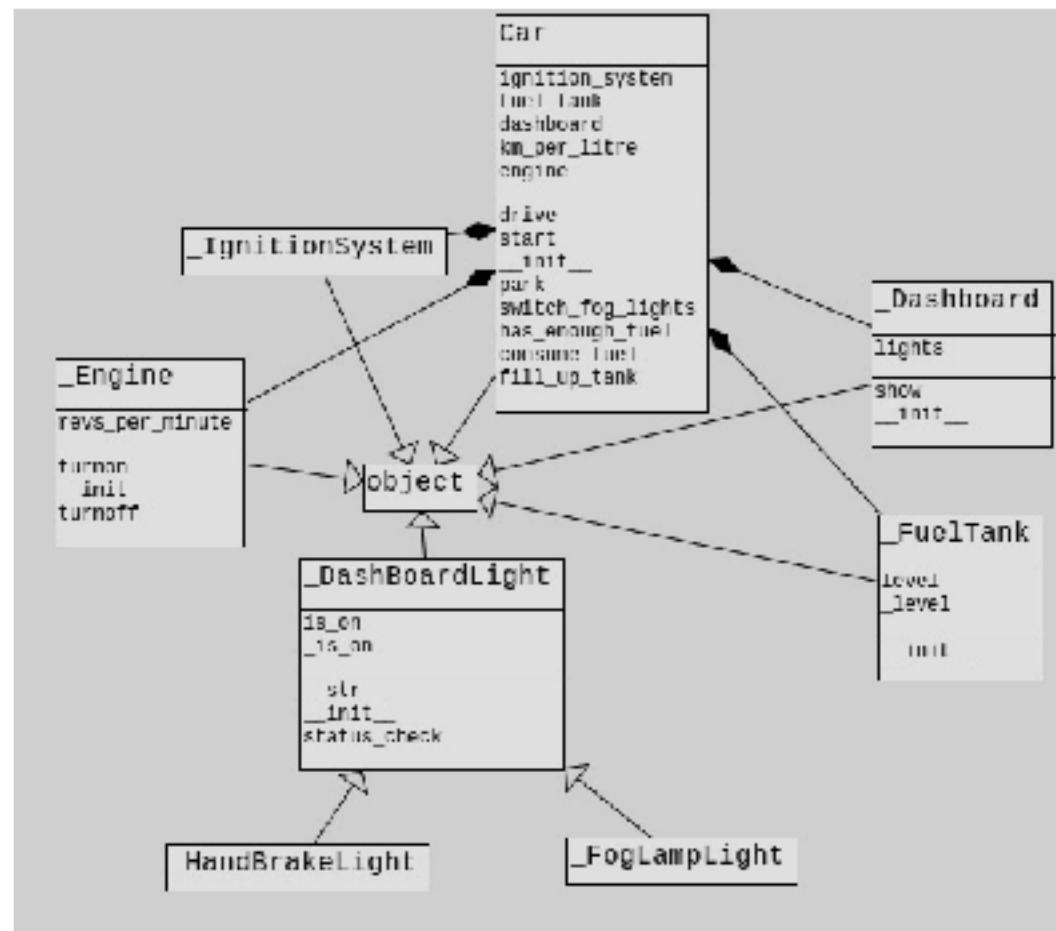
 drawing.clear()

 drawing.add(tri1)
 drawing.add(cir)
 drawing.draw("verde")
```

# Model Fațadă



# Fațadă - Bord



# Model Fațadă - implementare

```
class _IgnitionSystem(object):
 @staticmethod
 def produce_spark():
 return True
class _Engine(object):
 def __init__(self):
 self.revs_per_minute = 0
 def turnon(self):
 self.revs_per_minute = 2000
 def turnoff(self):
 self.revs_per_minute = 0
class _FuelTank(object):
 def __init__(self, level=30):
 self._level = level
 @property
 def level(self):
 return self._level
 @level.setter
 def level(self, level):
 self._level = level
```

```
class _DashBoardLight(object):
 def __init__(self, is_on=False):
 self._is_on = is_on
 def __str__(self):
 return self.__class__.__name__
 @property
 def is_on(self):
 return self._is_on
 @is_on.setter
 def is_on(self, status):
 self._is_on = status
 def status_check(self):
 if self._is_on:
 print("{}: Pornit".format(str(self)))
 else:
 print("{}: Oprit".format(str(self)))
class _HandBrakeLight(_DashBoardLight):
 pass
class _FogLampLight(_DashBoardLight):
 pass
```

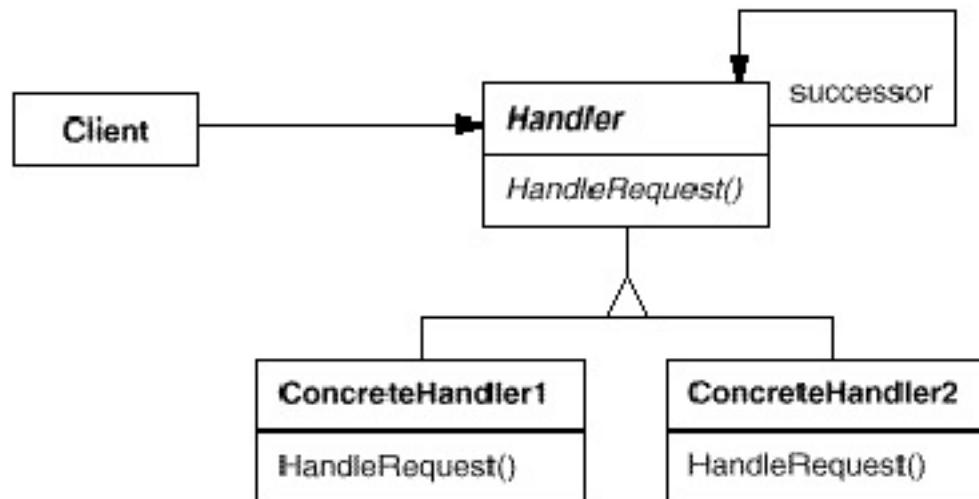
```

class _Dashboard(object):
 def __init__(self):
 self.lights = {"frana de mana": _HandBrakeLight(), "ceata": _FogLampLight()}
 def show(self):
 for light in self.lights.values():
 light.status_check()
class Car(object):#Fata de
 def __init__(self):
 self.ignition_system = _IgnitionSystem()
 self.engine = _Engine()
 self.fuel_tank = _FuelTank()
 self.dashboard = _Dashboard()
 @property
 def km_per_litre(self):
 return 17.0
 def consume_fuel(self, km):
 litres = min(self.fuel_tank.level, km / self.km_per_litre)
 self.fuel_tank.level -= litres
 def start(self):
 print("\nPornire...")
 self.dashboard.show()
 if self.ignition_system.produce_spark():
 self.engine.turnon()
 else:
 print("NU pot porni. Eroare la sistemul de aprindere")
 def has_enough_fuel(self, km, km_per_litre):
 litres_needed = km / km_per_litre
 if self.fuel_tank.level > litres_needed:
 return True
 else:
 return False
 def drive(self, km=100):
 print("\n")
 if self.engine.revs_per_minute > 0:
 while self.has_enough_fuel(km, self.km_per_litre):
 self.consume_fuel(km)
 print("S-a undus {} km".format(km))
 print("au mai ramas {:.2f} l de combustibil".format(self.fuel_tank.level))
 else:
 print("Nu se poate merge. Motorul este oprit!")
 def park(self):
 print("\nParcare...")
 self.dashboard.lights["frana de mana"].is_on = True
 self.dashboard.show()
 self.engine.turnoff()
 def switch_fog_lights(self, status):
 print("\nPornire {} lumini ceata...".format(status))
 boolean = True if status == "ON" else False
 self.dashboard.lights["ceata"].is_on = boolean
 self.dashboard.show()
 def fill_up_tank(self):
 print("\nRezervorul a fost umplut!")
 self.fuel_tank.level = 100
def main():#functia main reprezinta clientul
 car = Car()
 car.start()
 car.drive()
 car.switch_fog_lights("ON")
 car.switch_fog_lights("OFF")
 car.park()
 car.fill_up_tank()
 car.drive()
 car.start()
 car.drive()
if __name__ == "__main__":
 main()

```

# Modele comportamentale

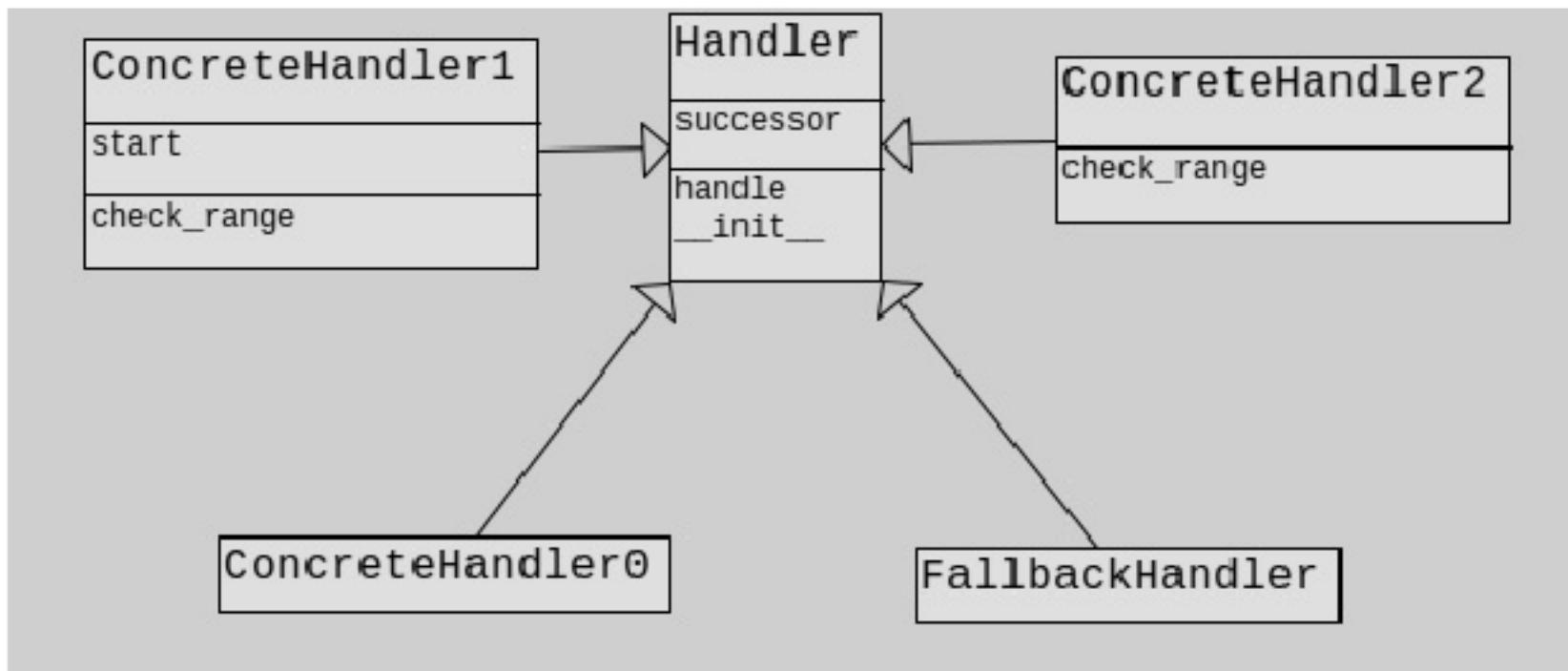
# Modelul lanț de responsabilități



Unde o structură tipică de înlănțuire de obiecte ar fi



# Caz concret cu trei gestionari diferenți



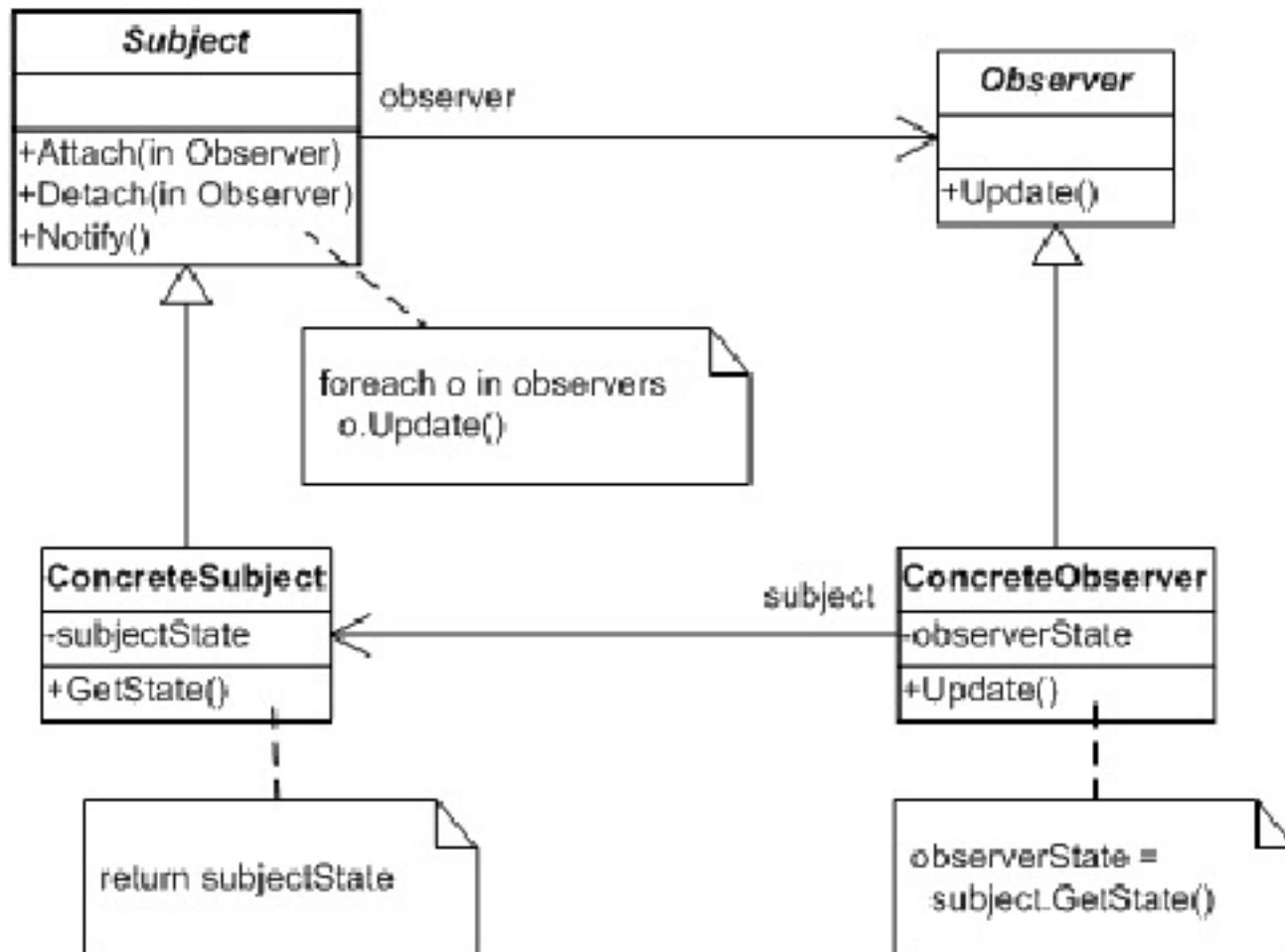
# Modelul lanț de responsabilități - implementare

```
import abc
class Handler(metaclass=abc.ABCMeta):
 def __init__(self, successor=None):
 self.successor = successor
 def handle(self, request):
 res = self.check_range(request)
 if not res and self.successor:
 self.successor.handle(request)
 @abc.abstractmethod
 def check_range(self, request):
 """compara valoarea primită cu un interval predefinită"""
class ConcreteHandler0(Handler):
 @staticmethod
 def check_range(request):
 if 0 <= request < 10:
 print("cererea {} tratată în gestionarul 0".format(request))
 return True
class ConcreteHandler1(Handler):# are propria stare internă
 start, end = 10, 20
 def check_range(self, request):
 if self.start <= request < self.end:
 print("cererea {} tratată de gestionarul 1".format(request))
 return True
```

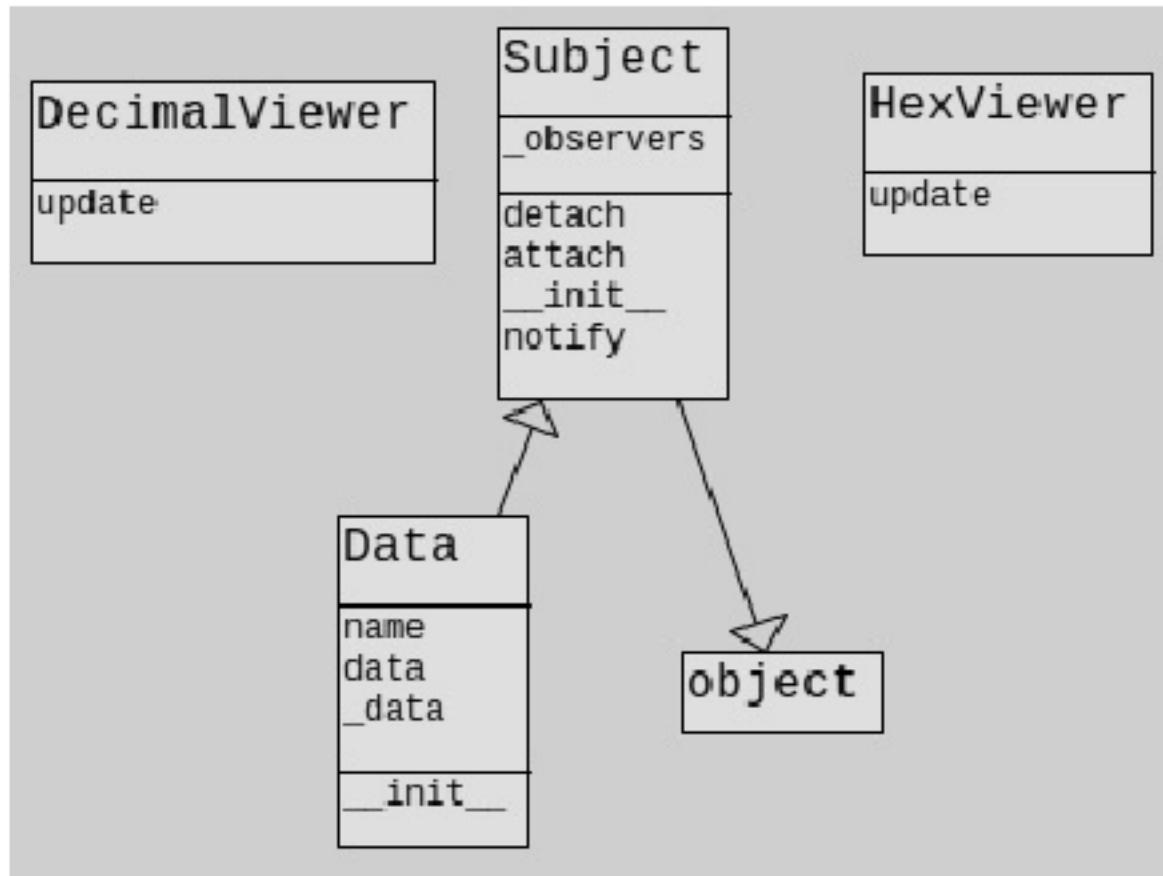
```
class ConcreteHandler2(Handler):#utilizează metode ajutătoare
 def check_range(self, request):
 start, end = self.get_interval_from_db()
 if start <= request < end:
 print("cererea {} tratată de gestionarul 2".format(request))
 return True
 @staticmethod
 def get_interval_from_db():
 return (20, 30)
class FallbackHandler(Handler):
 @staticmethod
 def check_range(request):
 print("am terminat de parcurs lantul - nu există tratare pentru cazul {}".
 format(request))
 return False
h0 = ConcreteHandler0() #creez gestionarii
h1 = ConcreteHandler1()
h2 = ConcreteHandler2(FallbackHandler())
h0.successor = h1 #creez lantul
h1.successor = h2
requests = [2, 5, 14, 22, 18, 3, 35, 27, 20]
for request in requests:
 h0.handle(request)
```

# Modelul Observator

# Modelul Observator



# Observator - caz de utilizare



# Model Observator - Caz de utilizare

```
class Subject(object):
 def __init__(self):
 self._observers = []
 def attach(self, observer):
 if observer not in self._observers:
 self._observers.append(observer)
 def detach(self, observer):
 try:
 self._observers.remove(observer)
 except ValueError:
 pass
 def notify(self, modifier=None):
 for observer in self._observers:
 if modifier != observer:
 observer.update(self)
class Data(Subject):# exemplu de utilizare
 def __init__(self, name=''):
 Subject.__init__(self)
 self.name = name
 self._data = 0
 @property
 def data(self):
 return self._data
 @data.setter
 def data(self, value):
 self._data = value
 self.notify()
class HexViewer:
 def update(self, subject):
 print(u'Format Hexa: Subiectul %s are valoarea 0x%x' % (subject.name, subject.data))
class DecimalViewer:
 def update(self, subject):
 print(u'Format Zecimal: Subiectul %s are valoarea %d' % (subject.name, subject.data))
```

```
def main():#utilizare
 data1 = Data('Data 1')
 data2 = Data('Data 2')
 view1 = DecimalViewer()
 view2 = HexViewer()
 data1.attach(view1)
 data1.attach(view2)
 data2.attach(view2)
 data2.attach(view1)
 print(u"Stabilim valoarea 11 = 10")
 data1.data = 10
 print(u"Stabilim valoarea 12 = 15")
 data2.data = 15
 print(u"Stabilim valoarea 11 = 3")
 data1.data = 3
 print(u"Stabilim valoarea 12 = 5")
 data2.data = 5
 print(u"NU mai utilizam Afisarea Hexa pentru data1 si data2.")
 data1.detach(view2)
 data2.detach(view2)
 print(u"Stabilim valoarea 11 = 10")
 data1.data = 10
 print(u"Stabilim valoarea 12 = 15")
 data2.data = 15
if __name__ == '__main__':
 main()
```

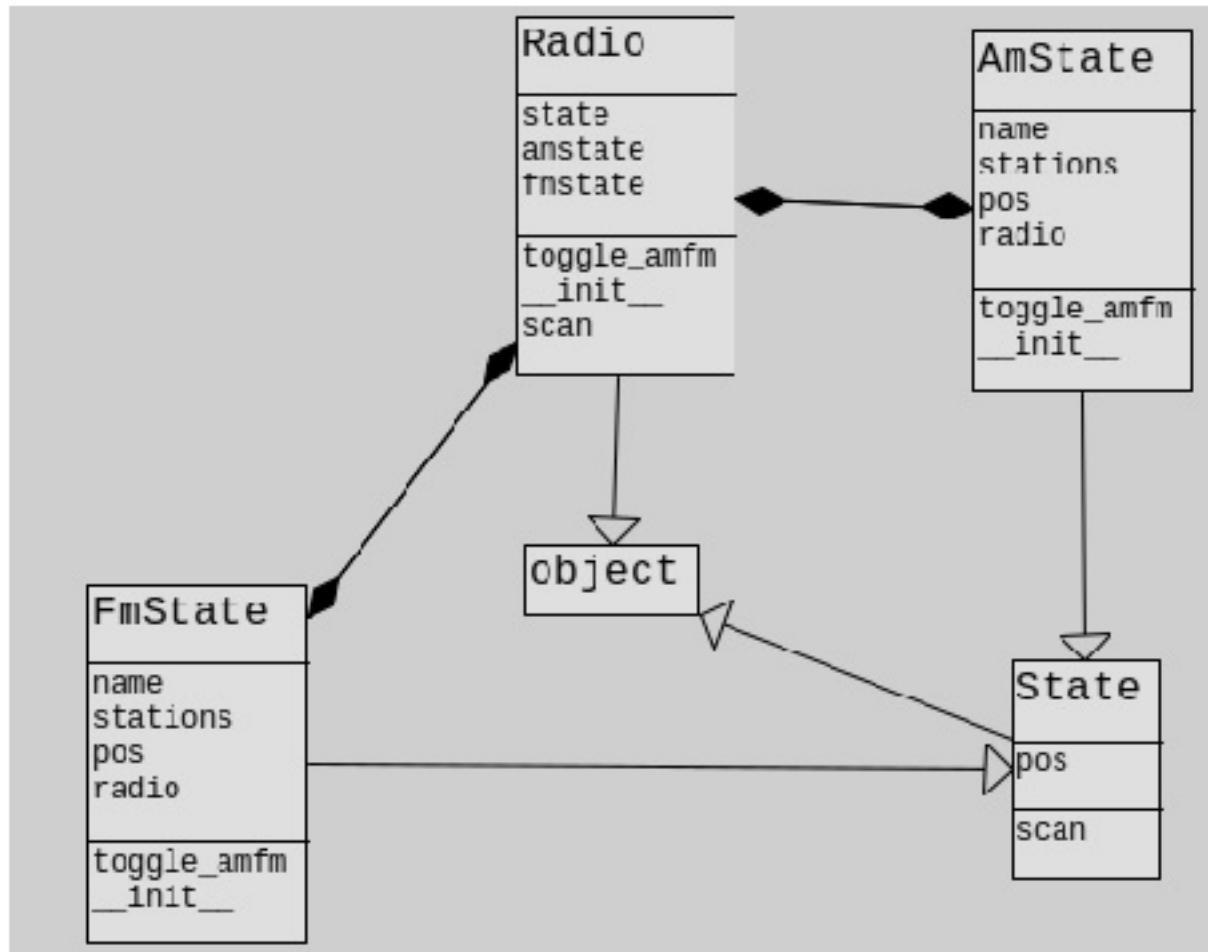
# Varianta publish - subscribe

```
class Provider:
 def __init__(self):
 self.msg_queue = []
 self.subscribers = {}
 def notify(self, msg):
 self.msg_queue.append(msg)
 def subscribe(self, msg, subscriber):
 self.subscribers.setdefault(msg, []).append(subscriber)
 def unsubscribe(self, msg, subscriber):
 self.subscribers[msg].remove(subscriber)
 def update(self):
 for msg in self.msg_queue:
 for sub in self.subscribers.get(msg, []):
 sub.run(msg)
 self.msg_queue = []
class Publisher:
 def __init__(self, msg_center):
 self.provider = msg_center
 def publish(self, msg):
 self.provider.notify(msg)
class Subscriber:
 def __init__(self, name, msg_center):
 self.name = name
 self.provider = msg_center
 def subscribe(self, msg):
 self.provider.subscribe(msg, self)

def unsubscribe(self, msg):
 self.provider.unsubscribe(msg, self)
def run(self, msg):
 print("{} got {}".format(self.name, msg))
def main():
 message_center = Provider()
 fftv = Publisher(message_center)
 jim = Subscriber("jim", message_center)
 jim.subscribe("cartoon")
 jack = Subscriber("jack", message_center)
 jack.subscribe("music")
 gee = Subscriber("gee", message_center)
 gee.subscribe("movie")
 vani = Subscriber("vani", message_center)
 vani.subscribe("movie")
 vani.unsubscribe("movie")
 fftv.publish("cartoon")
 fftv.publish("music")
 fftv.publish("ads")
 fftv.publish("movie")
 fftv.publish("cartoon")
 fftv.publish("cartoon")
 fftv.publish("movie")
 fftv.publish("blank")
 message_center.update()
if __name__ == "__main__":
 main()
```

# Modelul automatului finit State

# Automatul - caz concret

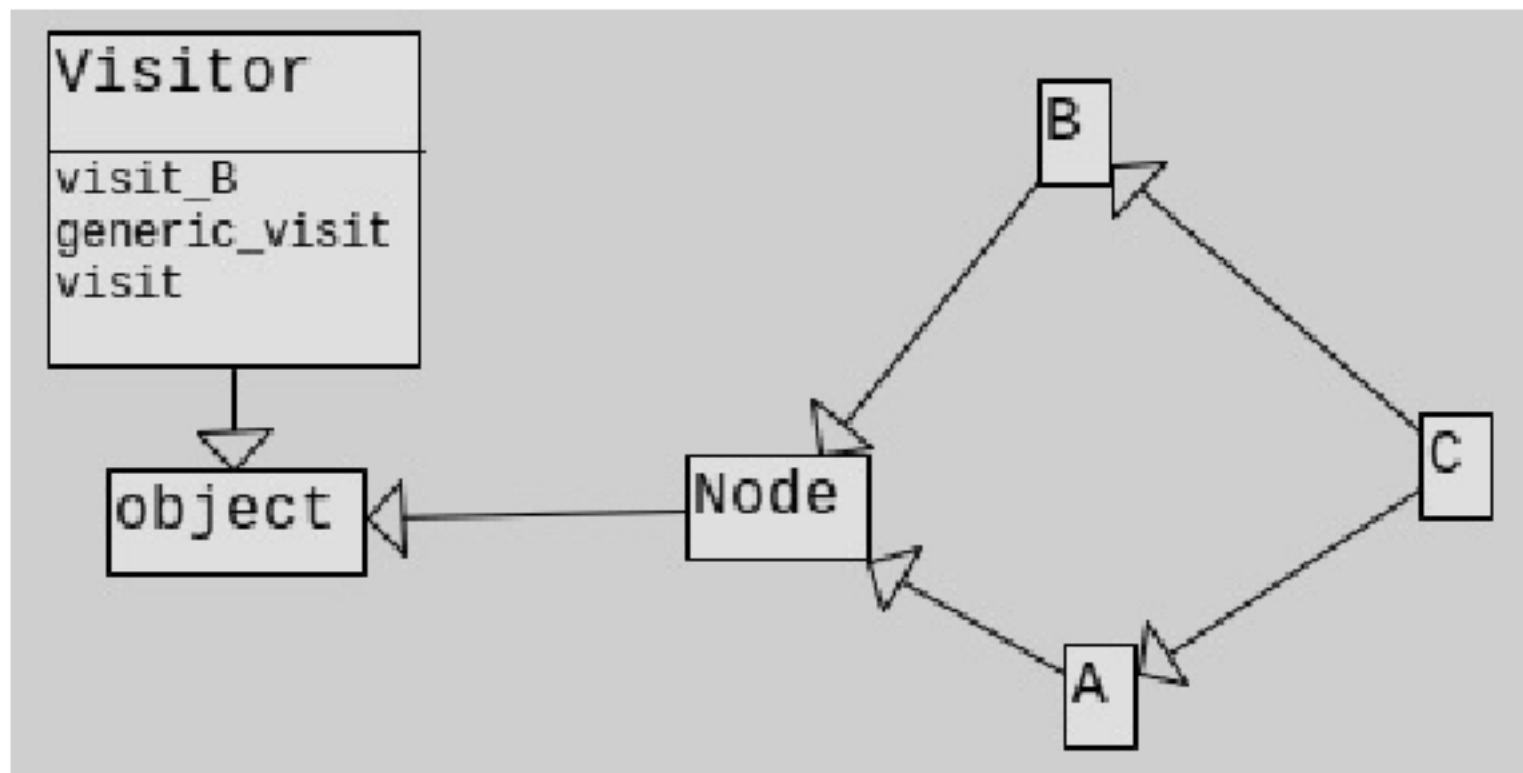


# Modelul automatului finit - caz de utilizare

```
class State(object):
 def scan(self): #stare de baza pentru a pune in comun o functionalitate
 self.pos += 1
 if self.pos == len(self.stations):
 self.pos = 0
 print(u"Caut... Am gasit o statie la %s in banda %s" %
 (self.stations[self.pos], self.name))
class AmState(State):
 def __init__(self, radio):
 self.radio = radio
 self.stations = ["1250", "1380", "1510"]
 self.pos = 0
 self.name = "AM"
 def toggle_amfm(self):
 print(u"Comutare in banda FM")
 self.radio.state = self.radio.fmstate
class FmState(State):
 def __init__(self, radio):
 self.radio = radio
 self.stations = ["81.3", "89.1", "103.9"]
 self.pos = 0
 self.name = "FM"
 def toggle_amfm(self):
 print(u"Comutare in banda AM")
 self.radio.state = self.radio.amstate
```

```
class Radio(object):
 def __init__(self): # are un buton pentru cautare post si unul #
 for action in actions:
 self.actions.append(action)
 def __name__ == '__main__':
 main()
 def __init__(self):
 self.am state = Am State(self)
 self.fm state = Fm State(self)
 self.state = self.am state
 def toggle_amfm(self):
 self.state.toggle_amfm()
 def scan(self):
 self.state.scan()
def main(): #testare radio
 radio = Radio()
 actions = [radio.scan] * 2 + [radio.toggle_amfm] + [radio.scan] * 2
 actions *= 2
 for action in actions:
 action()
```

# Modelul Vizitator?



# Vizitator - Exemplu de tratare

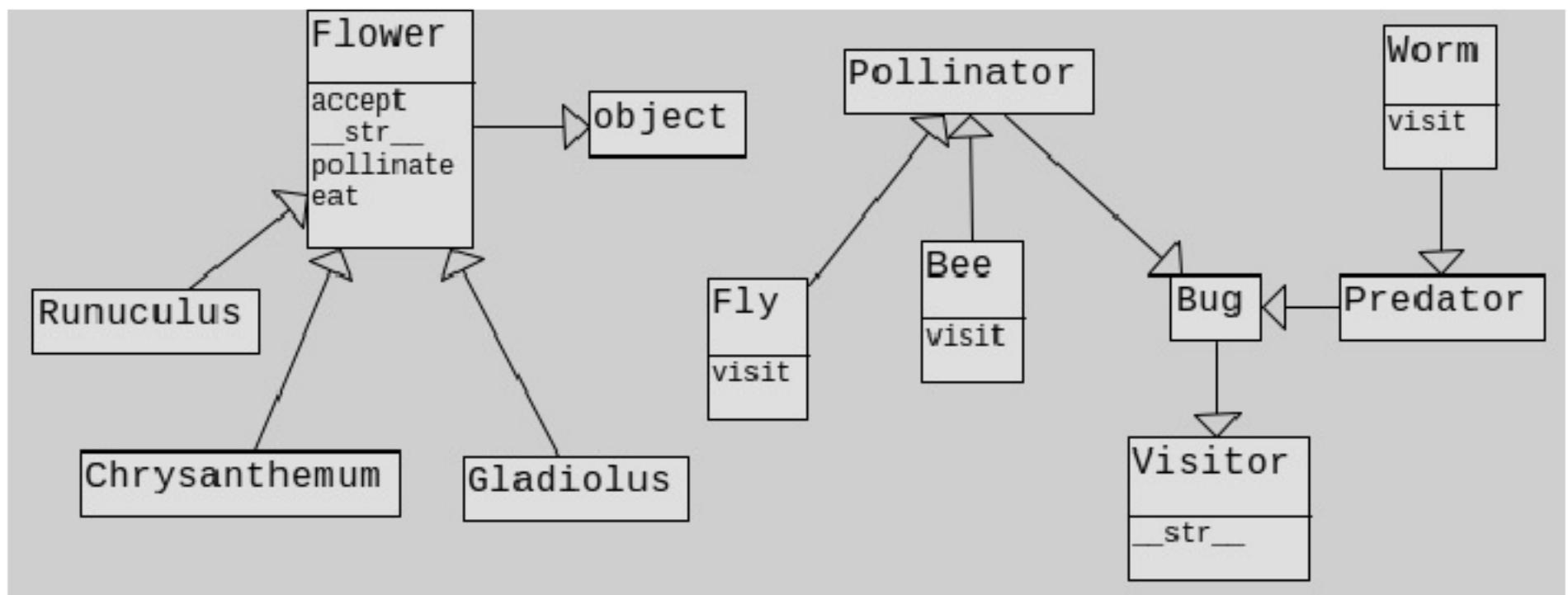
```
class Node(object):
 pass
class A(Node):
 pass
class B(Node):
 pass
class C(A, B):
 pass
class Visitor(object):
 def visit(self, node, *args, **kwargs):
 meth = None
 for cls in node.__class__.__mro__:
 meth_name = 'visit_' + cls.__name__
 meth = getattr(self, meth_name, None)
 if meth:
 break
 if not meth:
 meth = self.generic_visit
 return meth(node, *args, **kwargs)
```

```
def generic_visit(self, node, *args, **kwargs):
 print('generic_visit ' + node.__class__.__name__)
def visit_B(self, node, *args, **kwargs):
 print('visit_B ' + node.__class__.__name__)

def main():
 a = A()
 b = B()
 c = C()
 visitor = Visitor()
 visitor.visit(a)
 visitor.visit(b)
 visitor.visit(c)

if __name__ == "__main__":
 main()
```

# Vizitatori în grădină



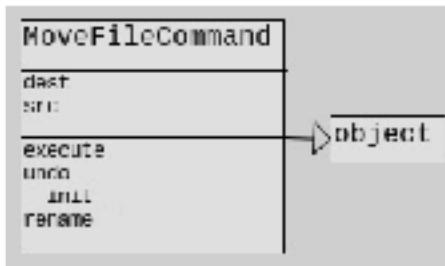
## Exemplu 2 de vizitator

```
import random
class Flower(object):
 def accept(self, visitor):
 visitor.visit(self)
 def pollinate(self, pollinator):
 print(self, "polenizata de ", pollinator)
 def eat(self, eater):
 print(self, "mancata de ", eater)
 def __str__(self):
 return self.__class__.__name__ #intorc numele clasei
class Gladiolus(Flower): pass
class Runuculus(Flower): pass
class Chrysanthemum(Flower): pass
class Visitor:
 def __str__(self):
 return self.__class__.__name__
class Bug(Visitor): pass
class Pollinator(Bug): pass
class Predator(Bug): pass
class Bee(Pollinator): # ce face o albina
 def visit(self, flower):
 flower.pollinate(self)
class Fly(Pollinator): #ce face o musca
 def visit(self, flower):
 flower.pollinate(self)
class Worm(Predator): # ce face un vierme
 def visit(self, flower):
 flower.eat(self)
def flowerGen(n):
 flwrs = Flower.__subclasses__()
 for i in range(n):
 yield random.choice(flwrs)()
bee = Bee()
fly = Fly()
worm = Worm()
for flower in flowerGen(10):
 flower.accept(bee)
 flower.accept(fly)
 flower.accept(worm)
```

# Modelul comandă

```
import os
from os.path import lexists
class MoveFileCommand(object):
 def __init__(self, src, dest):
 self.src = src
 self.dest = dest
 def execute(self):
 self.rename(self.src, self.dest)
 def undo(self):
 self.rename(self.dest, self.src)
 def rename(self, src, dest):
 print(u"Redenumesc %s ca %s" % (src, dest))
 os.rename(src, dest)

def main():
 command_stack = []
 # se introduc comenzile în stivă
 command_stack.append(MoveFileCommand('bugs.txt', 'bunny.txt'))
 command_stack.append(MoveFileCommand('bunny.txt',
'stimpy.txt'))
```

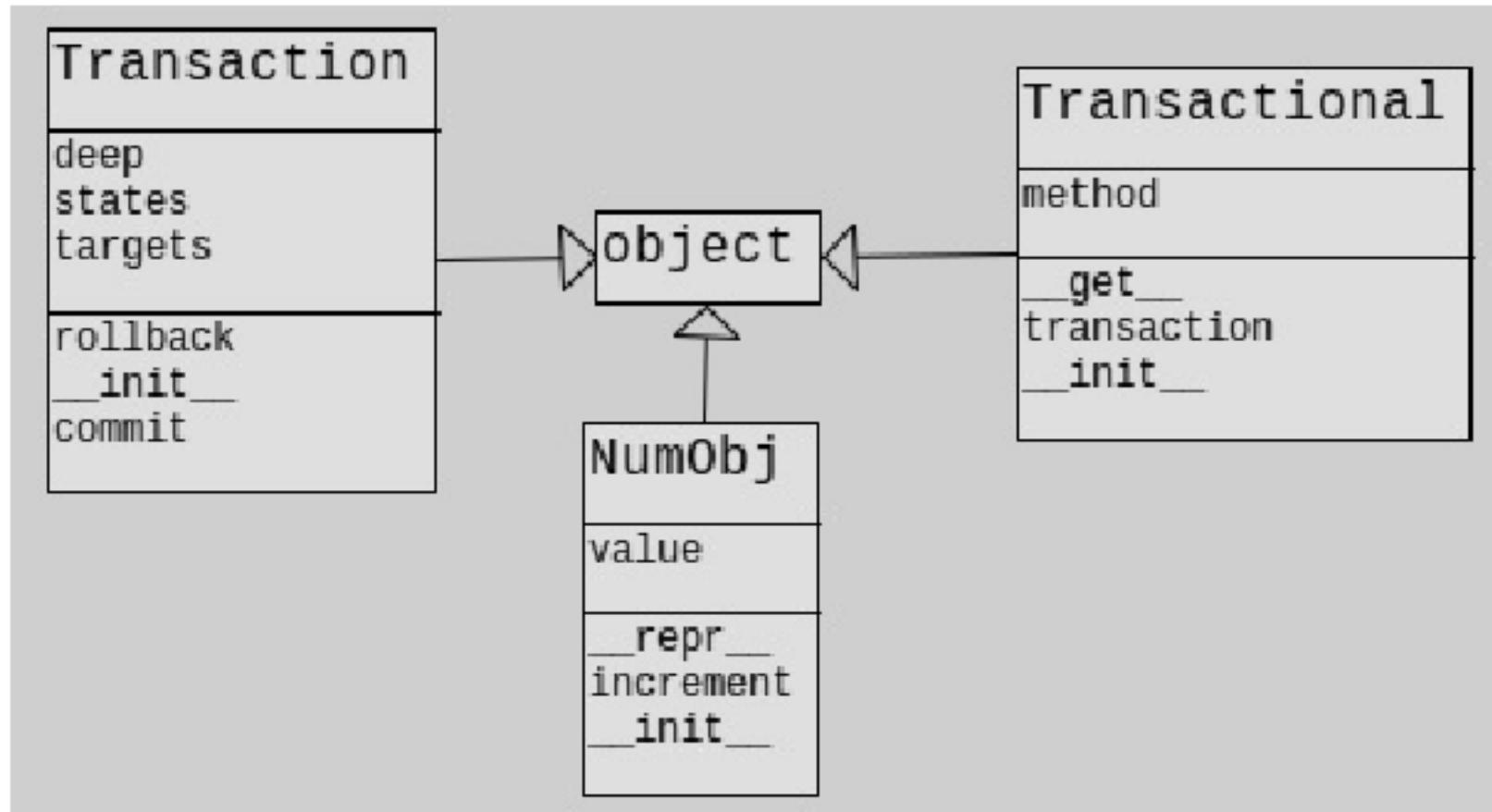


```
assert not lexists("bugs.txt")
assert not lexists("bunny.txt")
assert not lexists("stimpy.txt")
try:
 with open("bugs.txt", "w"): # Creez fisier
 pass # și cam atât
 for cmd in command_stack:# pot fi executate ulterior
 cmd.execute()

 for cmd in reversed(command_stack):
 cmd.undo() # poate fi anulat efectul
finally:# curățăm urma mea
 os.unlink("bugs.txt")
```

```
if __name__ == "__main__":
 main()
```

## modelul restaurare/reamintire (latină: memento)



## si implementarea

```
ifrom copy import copy
from copy import deepcopy
def memento(obj, deep=False):
 state = deepcopy(obj.__dict__) if deep else
copy(obj.__dict__)
 def restore():
 obj.__dict__.clear()
 obj.__dict__.update(state)
 return restore
class Transaction(object):# o tranzactie cu restaurare
 deep = False
 states = []
 def __init__(self, deep, *targets):
 self.deep = deep
 self.targets = targets
 self.commit()
 def commit(self):
 self.states = [memento(target, self.deep) for target in
self.targets]
 def rollback(self):
 for a_state in self.states:
 a_state()
```

```
class Transactional(object):
 def __init__(self, method):
 self.method = method

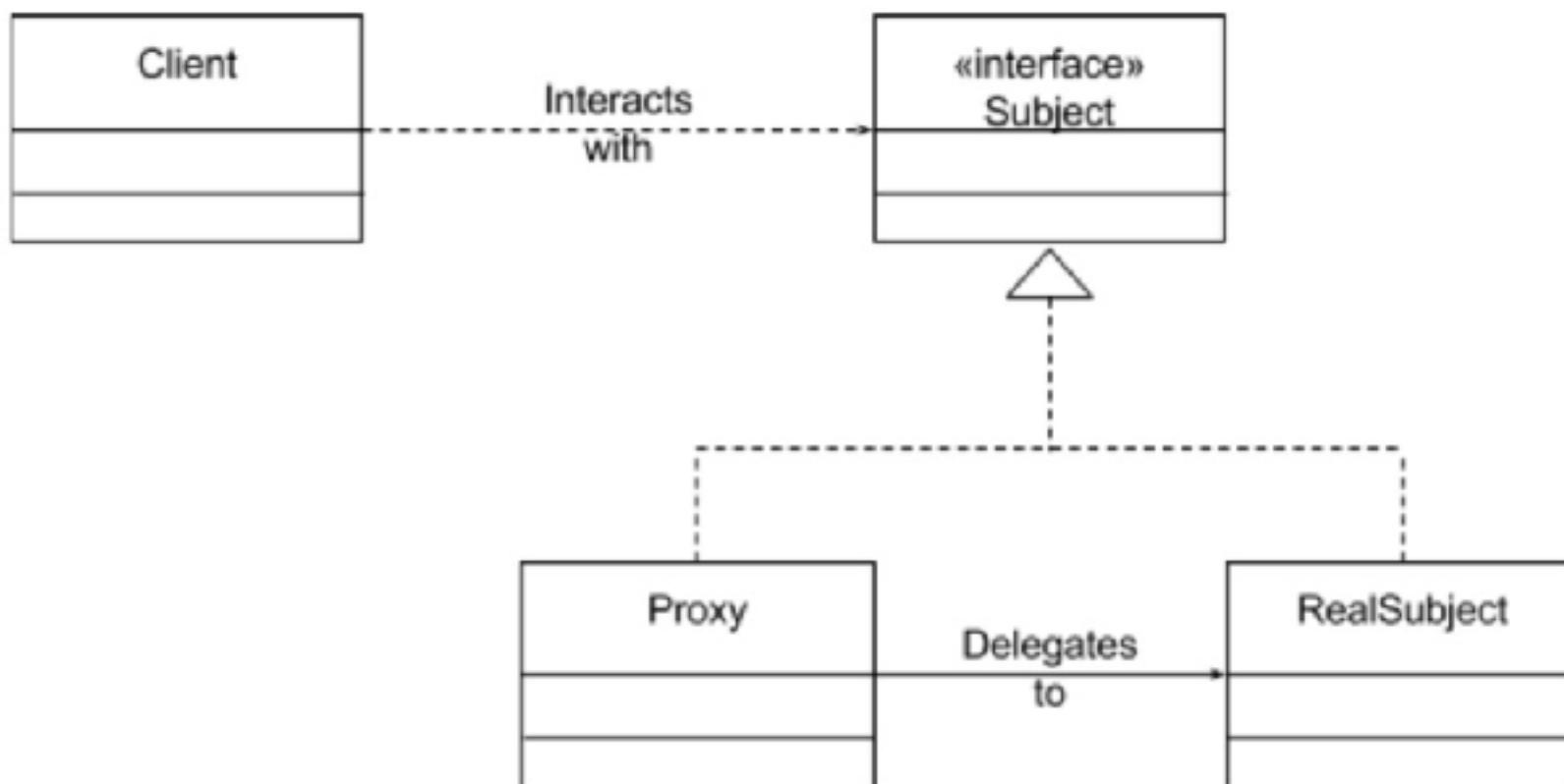
 def __get__(self, obj, T):
 def transaction(*args, **kwargs):
 state = memento(obj)
 try:
 return self.method(obj, *args, **kwargs)
 except Exception as e:
 state()
 raise e
 return transaction
class NumObj(object):
 def __init__(self, value):
 self.value = value
 def __repr__(self):
 return '<%s: %r>' % (self.__class__.__name__,
self.value)
```

# continuare

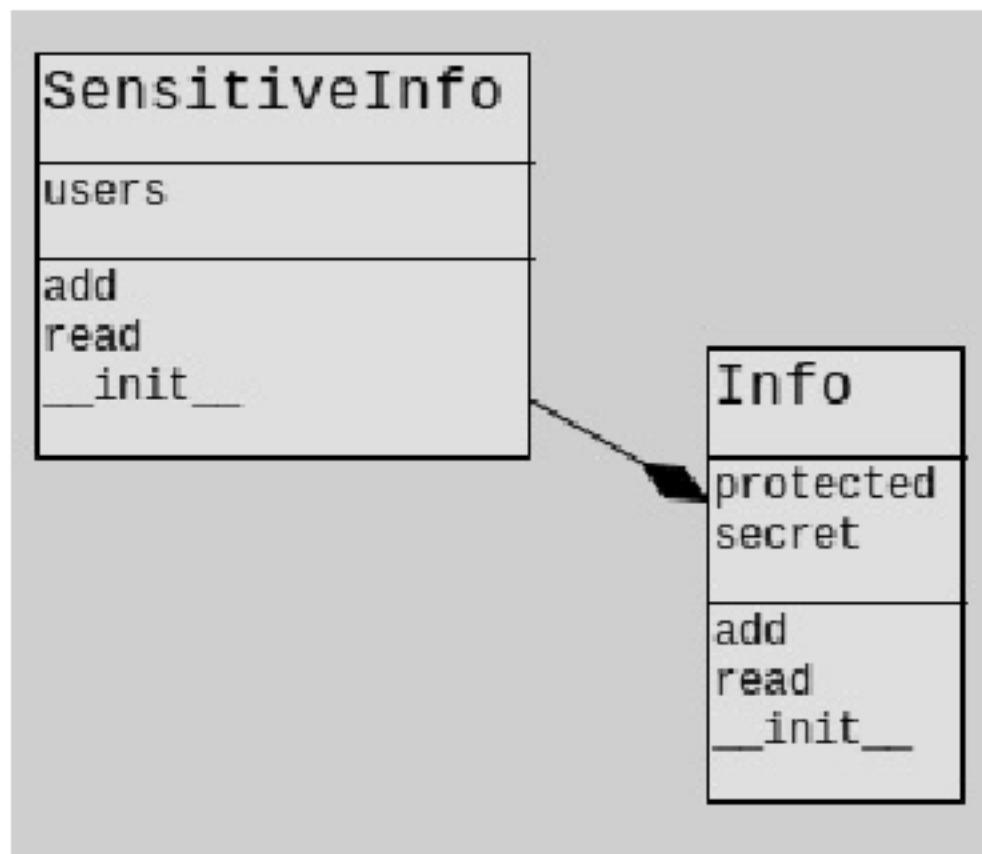
```
def increment(self):
 self.value += 1
@Transactional
def do_stuff(self): #tot ca sa crape
 self.value = '1111' # <- valoare incorecta
 self.increment() # <- crapa si face restaurare
#main
num_obj = NumObj(-1)
print(num_obj)
a_transaction = Transaction(True, num_obj)
try:
 for i in range(3):
 num_obj.increment()
 print(num_obj)
 a_transaction.commit()
 print('-- tranzactie finalizata')
 num_obj.value += 'x' # incorcet va crapa
 print(num_obj)
except Exception:
 a_transaction.rollback()
 print('-- tranzactie esuata - restauram la situatia anterioara')
```

```
print(num_obj)
print('-- alte instructiuni gresite ...')
try:
 num_obj.do_stuff()
except Exception:
 print('-> do_stuff a crapat!')
 import sys
 import traceback
 traceback.print_exc(file=sys.stdout)
#afisez explicit erorile de executie
print(num_obj)
```

## Intermediar - forma generală



## Intermediar - caz de utilizare



# Intermediar - caz de utilizare - implementare

```
class SensitiveInfo:
 def __init__(self):
 self.users = ['bula', 'strula', 'bugs', 'mike']

 def read(self):
 nb = len(self.users)
 print(f"Sunt {nb} utilizatori: {' '.join(self.users)}")

 def add(self, user):
 self.users.append(user)
 print(f'Adauga loser {user}')

class Info:
 def __init__(self):
 self.protected = SensitiveInfo()
 self.secret = '0xdeadbeef'

 def read(self):
 self.protected.read()

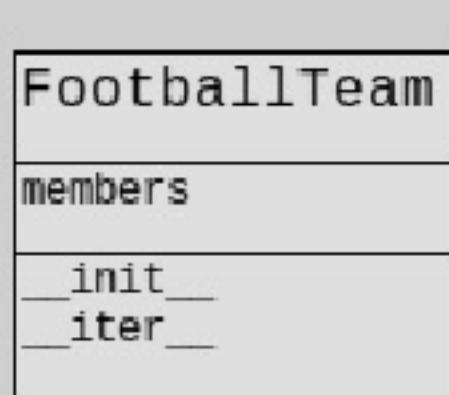
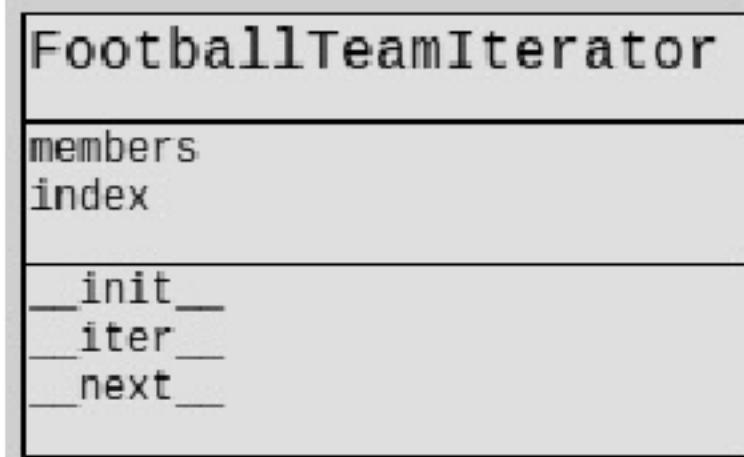
 def add(self, user):
 sec = input('dati parola? ')
 self.protected.add(user) if sec == self.secret else print("Mai incercam!")

def main():
 info = Info()

 while True:
 print('1. afiseaza lista |==| 2. adauga loser |==| 3. iesire')
 key = input('Alegeti o optiune: ')
 if key == '1':
 info.read()
 elif key == '2':
 name = input('Dati numele utilizatorului: ')
 info.add(name)
 elif key == '3':
 exit()
 else:
 print(f'Optiune invalida: {key}')

if __name__ == '__main__':
 main()
```

# Model iterator



# Modelul iterator - implementare

```
class FootballTeamIterator:
 def __init__(self, members):# lista de jucatori si
 antroni
 self.members = members
 self.index = 0
 def __iter__(self):
 return self
 def __next__(self):
 if self.index < len(self.members):
 val = self.members[self.index]
 self.index += 1
 return val
 else:
 raise StopIteration()
class FootballTeam:
 def __init__(self, members):
 self.members = members
 def __iter__(self):
 return FootballTeamIterator(self.members)
```

```
def main():
 members = []
 for x in range(1, 23):
 members.append(f'jucator_nr_{str(x)}')
 members = members + ['antrenor principal',
 'antrenor secund', 'antrenorul cu cafelele']
 team = FootballTeam(members)
 team_it = iter(team)

 while True:
 try:
 print(next(team_it))
 except StopIteration:
 break

if __name__ == '__main__':
 main()
```

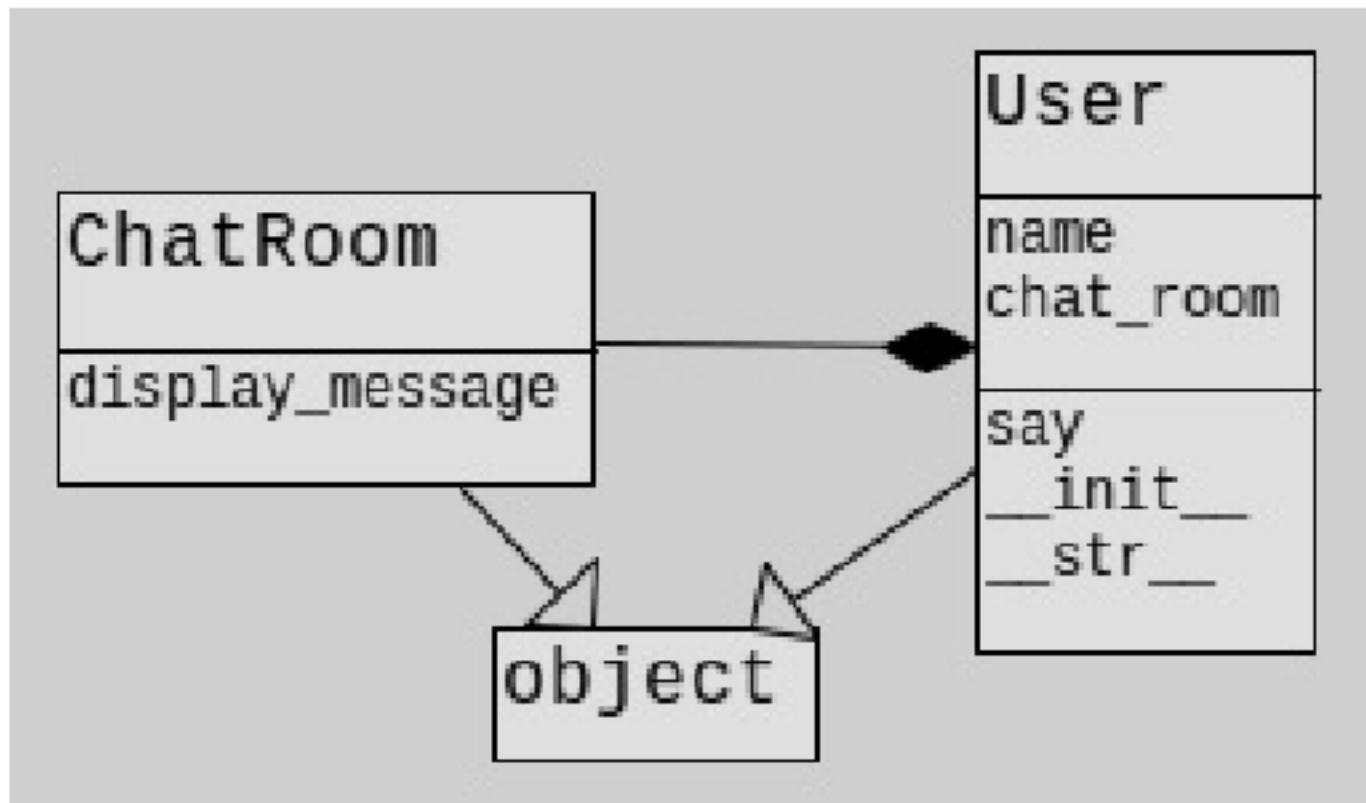
# Modelul Strategie

```
import time
def pairs(seq):
 n = len(seq)
 for i in range(n):
 yield seq[i], seq[(i + 1) % n]
SLOW = 3 # in secunde
LIMIT = 5 # in caractere
WARNING = 'nu este bine ai ales un algoritm lent :('
def allUniqueSort(s):
 if len(s) > LIMIT:
 print(WARNING)
 time.sleep(SLOW)
 srtStr = sorted(s)
 for (c1, c2) in pairs(srtStr):
 if c1 == c2:
 return False
 return True
def allUniqueSet(s):
 if len(s) < LIMIT:
 print(WARNING)
 time.sleep(SLOW) #pentru a simula un alg lent
 return True if len(set(s)) == len(s) else False
def allUnique(word, strategy):
 return strategy(word)
```

```
def main():
 WORD_IN_DESC = 'Introducei un cuvant (papa pentru iesire)\n>'
 STRAT_IN_DESC = 'Alegeti o strategie: [1] bazta pe un set, [2]\nsorteaza si imperecheaza >'
 while True:
 word = None
 while not word:
 word = input(WORD_IN_DESC)
 if word == 'papa':
 print('pa!!!')
 return
 strategy_picked = None
 strategies = {'1': allUniqueSet, '2': allUniqueSort}
 while strategy_picked not in strategies.keys():
 strategy_picked = input(STRAT_IN_DESC)
 try:
 strategy = strategies[strategy_picked]
 result = allUnique(word, strategy)
 print(f'allUnique({word}): {result}')
 except KeyError as err:
 print(f'Selectie gresita!: {strategy_picked}')
 if __name__ == "__main__":
 main()
```

# Modelul Mediator

## Versiune simplificată



# Model Mediator - caz de utilizare

```
class ChatRoom(object):#clasa mediator

 def display_message(self, user, message):
 print("{} zice: {}".format(user, message))

class User(object):
 def __init__(self, name):
 self.name = name
 self.chat_room = ChatRoom()

 def say(self, message):
 self.chat_room.display_message(self, message)

 def __str__(self):
 return self.name

def main():
 vasale = User('Vasale')
 tica = User('Tica2')
 altul = User('Altul')

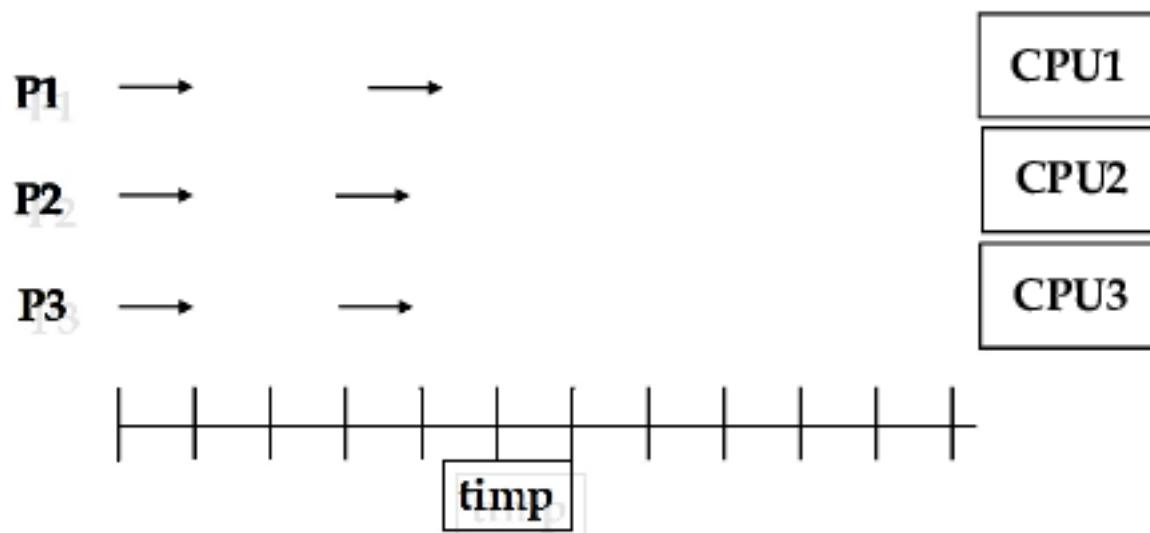
 vasale.say("Echipa adunarea la ora 3 dupa
amiaza!")
 tica.say("Da sefu pot sa astept pana atunci in
fata usii?")
 altul.say("da' eu pot?")

if __name__ == '__main__':
 main()
```

# *Paradigma Sequentiala versus Concurrency*

Cursul nr. 10  
Mihai Zaharia

## **Ce este calculul paralel**



**Numarul de programe/procese active**

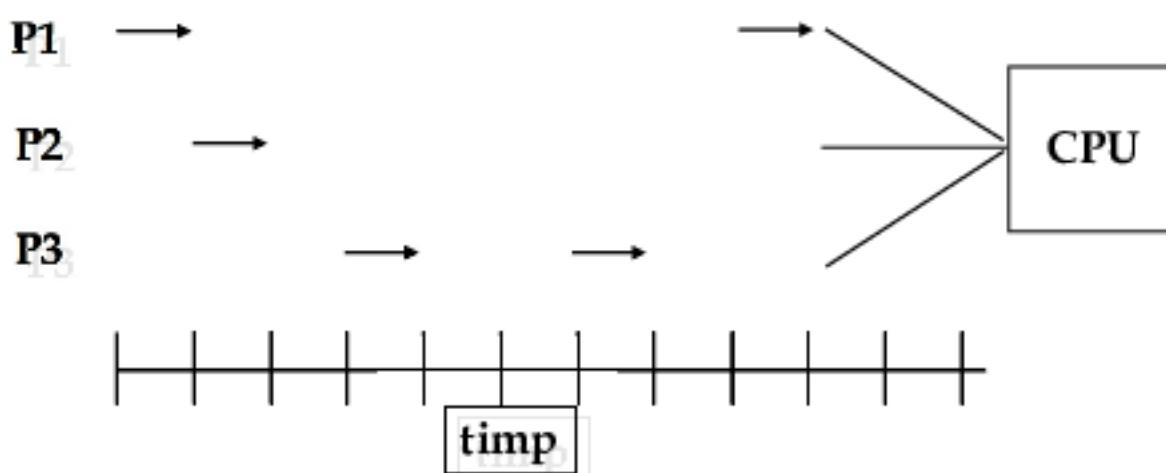
=

**numărul de procesoare**

# Ce este concurență?

- Concurență Vs Paralelism

Concurență

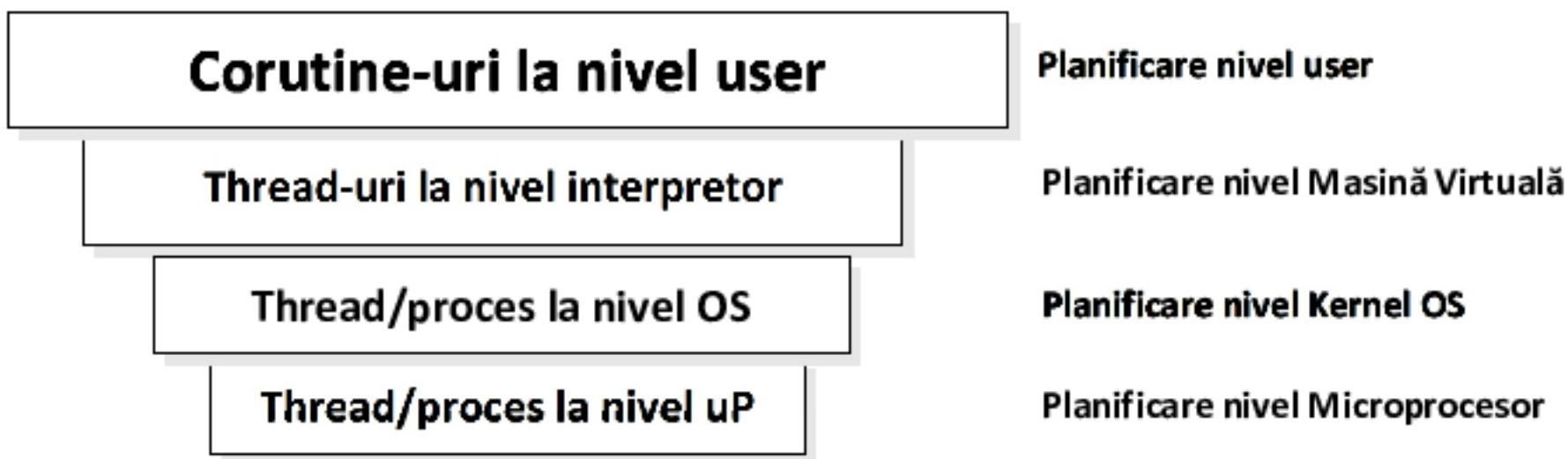


Numărul de entități care efectuează ceva

>

numărul de procesoare

## Hierarhia de control la nivel Kotlin

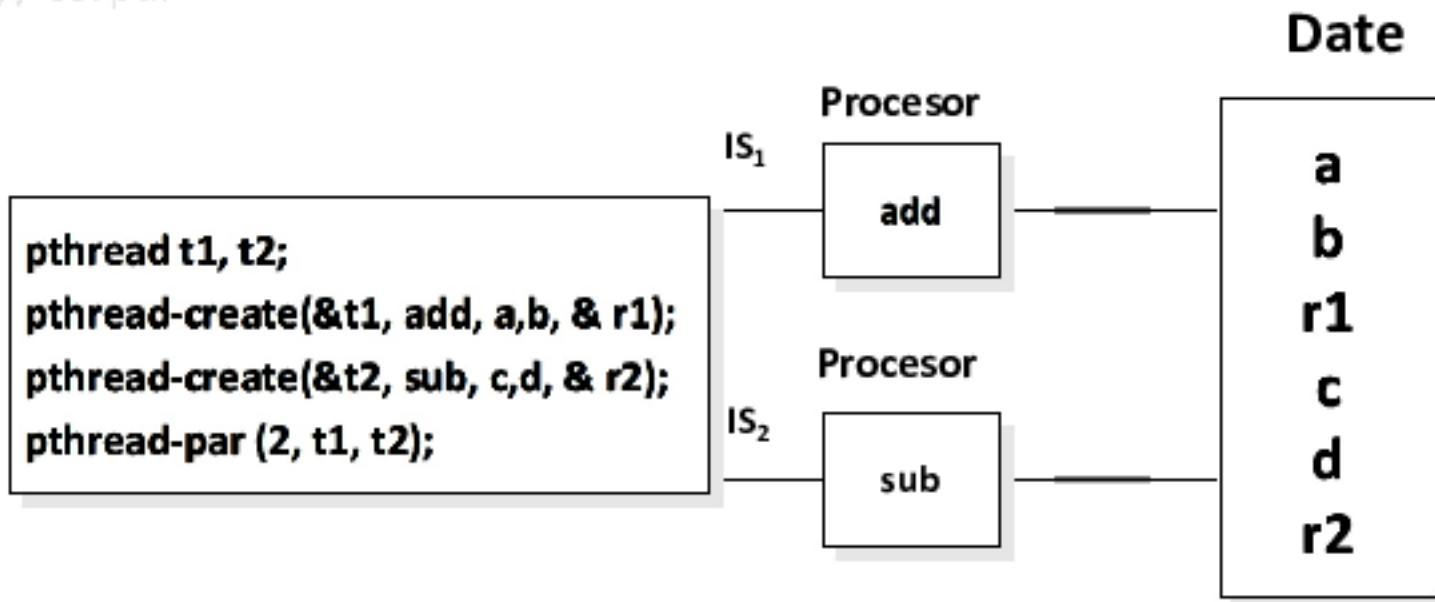


Execuție :

- Concurență de coroutine la nivel thread-uri din mașina virtuală
- Concurență de procese/thread-uri la nivel de OS
- Paralelism real: maparea procese / thread-uri : procesor = 1:1

# Concurență la nivel de date

```
int add (int a, int b, int & result)
//corpu
int sub(int a, int b, int & result)
//corpu
```



# Paralelismul la nivel datelor

```
sort(int *array, int count)
```

```
//.....
```

```
//.....
```

```
pthread-t, thread1, thread2;
```

```
"
```

```
"
```

```
pthread-create(& thread1, sort, array, N/2);
pthread-create(& thread2, sort, array, N/2);
pthread-par(2, thread1, thread2);
```

Date

do

"

"

$d_{n/2}$

$d_{n/2+1}$

"

"

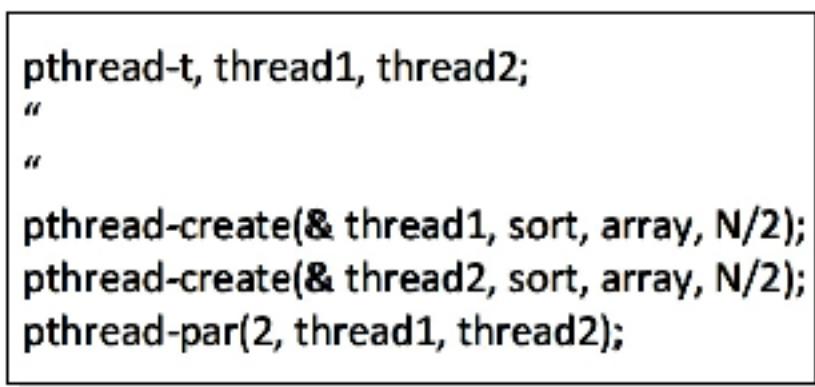
$d_n$

Procesor

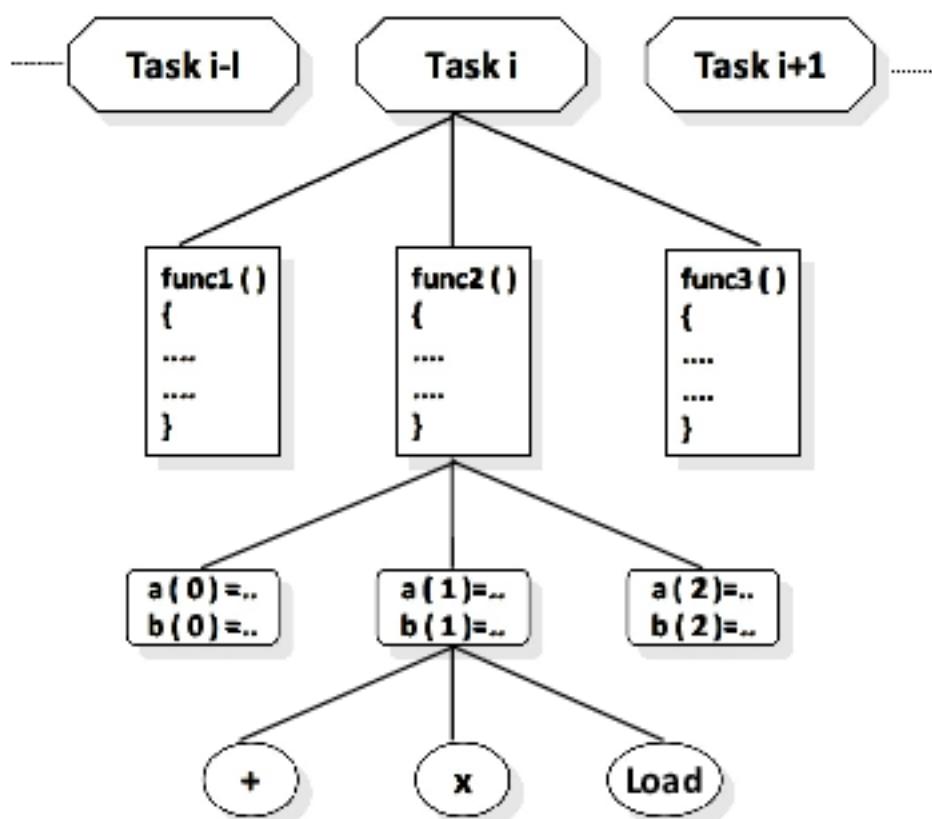
Sortare

Procesor

Sortare



# Granularitatea



**Granularitate cod**

**Entitate Cod**

**Granularitate mare**

**(nivel task)**

**Program**

**Granularitate medie**

**(nivel control)**

**Funcție (coruină/thread)**

**Granularitate fină (nivel date)**

**Bucă**

**Granularitate foarte fină**

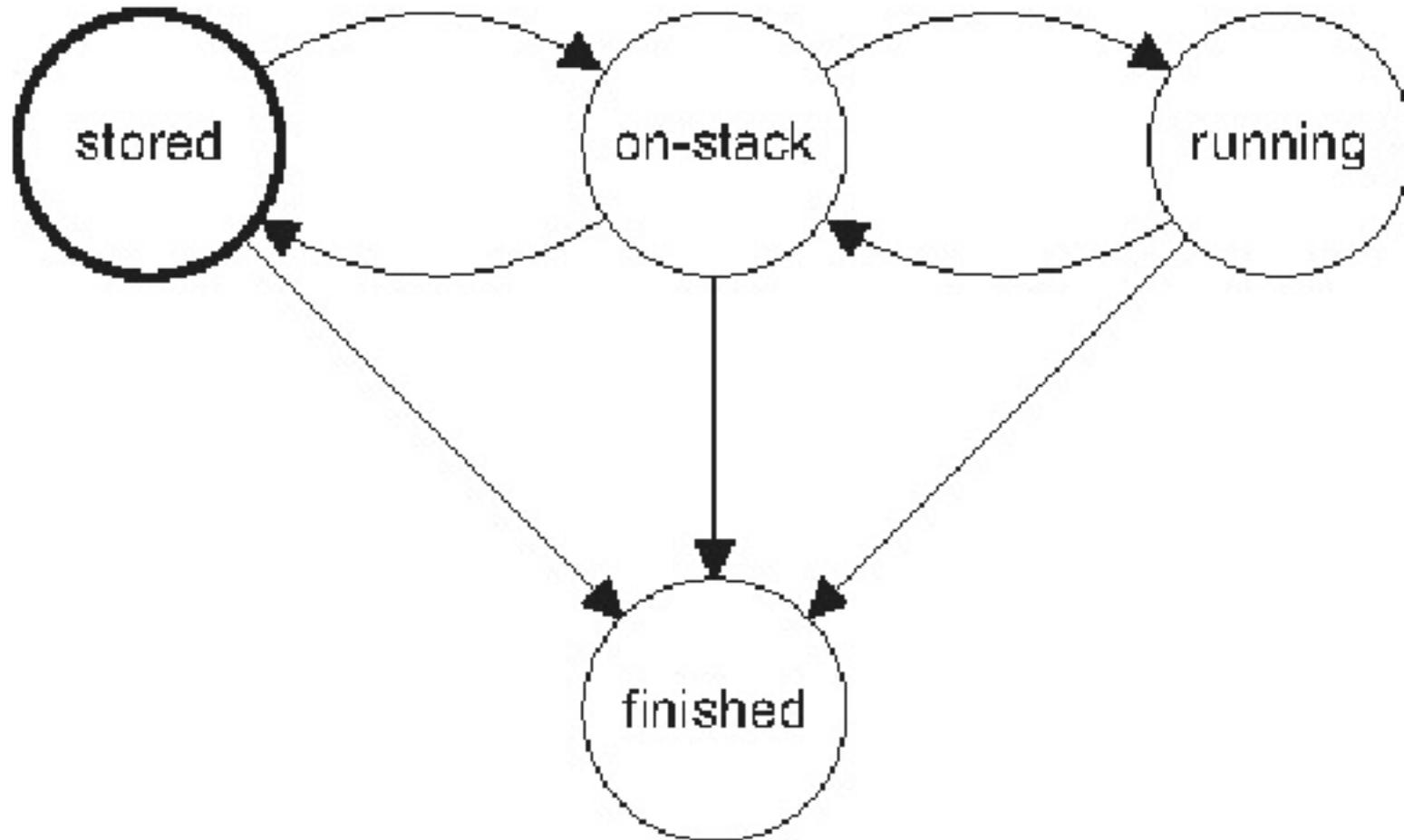
**(alegeri multiple )**

**Cu suport hard**

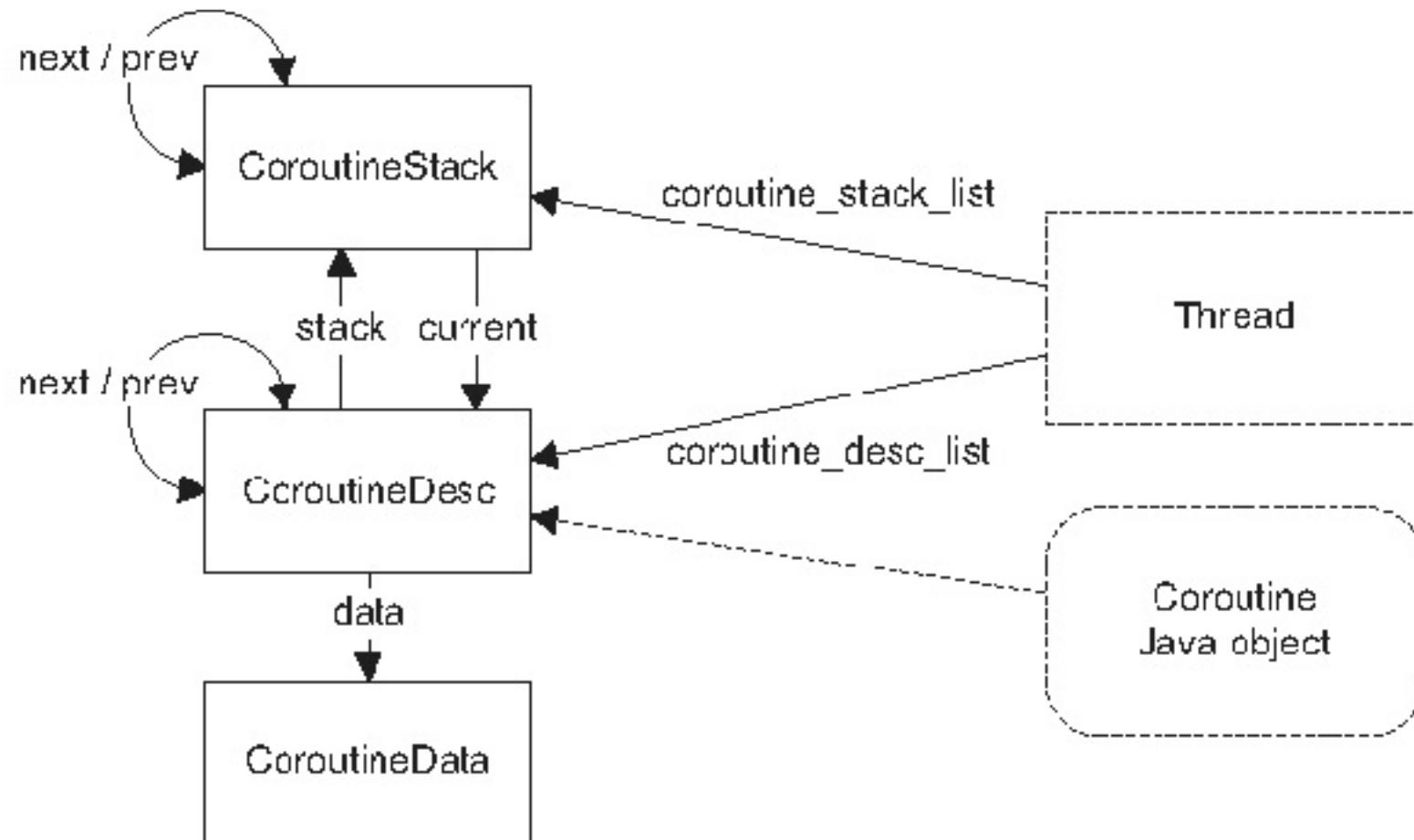
# **Ce sunt corutinele?**

- ceva vechi
- ceva nou

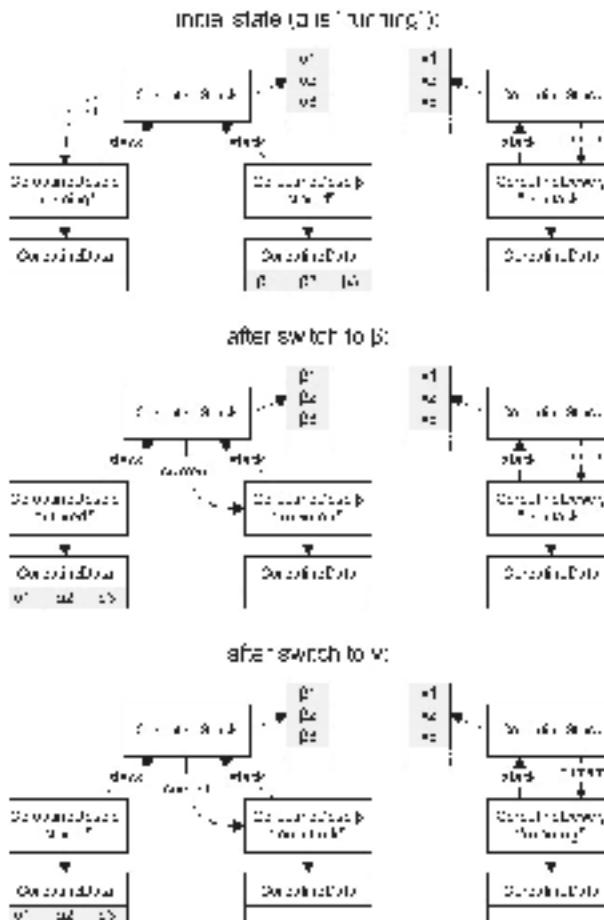
## Ciclul de viață al corutinelor



# Relația cu JVM



# Cum se schimbă stările



# Mapare coroutine pe thread-uri

```
import kotlin.system.*
import kotlinx.coroutines.*
fun main(args: Array<String>) =
 runBlocking {
 println("${Thread.activeCount()} fire
de executie active la pornire")
 val time = measureTimeMillis {
 createCoroutines(10_000)
 }
 println("${Thread.activeCount()} fire
de executie active la sfarsit")
 println("Procesul a durat $time ms")
 }
```

```
suspend fun createCoroutines(amount: Int) {
 val jobs = ArrayList<Job>()
 for (i in 1..amount) {
 jobs += GlobalScope.launch {
 println("Am pornit $i in ${Thread.currentThread().name}")
 delay(1000)
 println("S-a terminat $i din
${Thread.currentThread().name}")
 }
 }
 jobs.forEach {
 it.join()
 }
}
//S-a terminat 9998 din DefaultDispatcher-worker-5
//11 fire de executie active la sfarsit
//Procesul a durat 1186 ms - pe sistemul meu
```

# Relația corutină-thread

2 fire de executie active la pornire  
Pornit 3 in DefaultDispatcher-worker-1  
Pornit 5 in DefaultDispatcher-worker-5  
Pornit 4 in DefaultDispatcher-worker-3  
Pornit 2 in DefaultDispatcher-worker-2  
Pornit 6 in DefaultDispatcher-worker-6  
Pornit 1 in DefaultDispatcher-worker-4  
Pornit 7 in DefaultDispatcher-worker-7  
Pornit 8 in DefaultDispatcher-worker-8  
Pornit 9 in DefaultDispatcher-worker-4

.....  
Terminat 8 din DefaultDispatcher-worker-7  
Terminat 6 din DefaultDispatcher-worker-8  
Terminat 5 din DefaultDispatcher-worker-5  
Terminat 7 din DefaultDispatcher-worker-2  
Terminat 2 din DefaultDispatcher-worker-1  
Terminat 4 din DefaultDispatcher-worker-4  
Terminat 1 din DefaultDispatcher-worker-6  
Terminat 3 din DefaultDispatcher-worker-3

# Creare diverse tipuri de thread

```
import kotlinx.coroutines.*
fun main() = runBlocking<Unit> {

 launch { // contextul parinte - corutina functiei main cu runBlocking
 println("Corutina principala runBlocking : Sunt in thread ${Thread.currentThread().name}")
 }
 launch(Dispatchers.Unconfined) { // not confined -- va lucra cu thread-ul principal
 println("Independenta : Sunt in thread ${Thread.currentThread().name}")
 }
 launch(Dispatchers.Default) { // gestionata de DefaultDispatcher
 println("Implicita : Sunt in thread ${Thread.currentThread().name}")
 }
 launch(newSingleThreadContext("Threadul Meu")) { // va primi propriul thread
 println("newSingleThreadContext: Sunt in thread ${Thread.currentThread().name}")
 }
 si exemplu de executie
 Independenta : Sunt in thread main
 Implicita : Sunt in thread DefaultDispatcher-worker-1
 Corutina principala runBlocking : Sunt in thread main
 newSingleThreadContext: Sunt in thread Threadul Meu
```

# Exemplu oprire forțată a unei corutine

```
import kotlinx.coroutines.*

fun main() = runBlocking {
 val job = launch {
 // Emulate some batch processing
 repeat(30) { i ->
 println("Calculam ceva și ...")
 delay(300L)
 }
 }
 delay(1000L)
 println("main: Utilizatorul a cerut oprirea calculelor")
 job.cancelAndJoin() // da comanda de terminare si asteptata efectuarea ei
 println("main: Operatiunea in curs a fost abandonata")
}
```

**și rezultatul executiei**

Calculam ceva 0 ...

Calculam ceva 1 ...

Calculam ceva 2 ...

Calculam ceva 3 ...

main: Utilizatorul a cerut oprirea calculelor

main: Operatiunea in curs a fost abandonata

## Exemplu de oprire după depasirea limitei de timp

```
import kotlinx.coroutines.*
fun main()
{
 puturos()
}
fun puturos()
{
 runBlocking
 {
 val job = launch
 {
 try
 {
 withTimeout(1000L)
 {
 repeat(30) { i->
 println("Calculez $i ...")
 delay(300L)
 }
 }
 }catch(e: TimeoutCancellationException){println("sunt un lenes si ma opresc")}
 }
 }
}
```

# Depășire de timp fără excepții

```
import kotlinx.coroutines.*
import kotlinx.coroutines.withTimeoutOrNull as withTimeoutOrNull1
fun main()
{ if(null == lenes())println("ma opresc din lene") }
fun lenes(): String? {
 var status:String?=""
 runBlocking {
 val status1= withTimeoutOrNull1(1000L) {
 repeat(30) { i ->
 println("Calcul numarul $i ...")
 delay(300L)
 }
 "Gata" //incercati sa-l stergeti
 }
 status=status1
 }
 return status;
}
```

# Distrugere la ordin

```
import kotlinx.coroutines.*

class Activity : CoroutineScope by CoroutineScope(Dispatchers.Default) {
 fun destroy() {
 cancel() // se realizeaza o extindere a scopului corutinei CoroutineScope
 }
 fun doSomething() {
 repeat(10) { i ->
 launch {
 delay((i + 1) * 200L) // 200ms, 400ms, ... etc
 println("Coroutina $i s-a terminat")
 }
 }
 }
}

fun main() = runBlocking<Unit> {
 val activity = Activity()
 activity.doSomething() // run test function
 println("pornim corutinele")
 delay(500L)
 println("Distrug activitatile!")
 activity.destroy() // le omor pe toate
}
```

**si exemplu executie**  
pornim corutinele  
Coroutina 0 s-a terminat  
Coroutina 1 s-a terminat  
Distrug activitatile!

# Thread-local data

```
import kotlinx.coroutines.*

val threadLocal = ThreadLocal<String?>() // se declara referinta catre thread-ul local
fun main() = runBlocking<Unit> {
 threadLocal.set("thread-ul cu prisina:")
 println("Pre-main, current thread: ${Thread.currentThread()}, numit: '${threadLocal.get()}'")
 val job = launch(Dispatchers.Default + threadLocal.asContextElement(value = "launch")) {
 println("Sunt acum in: ${Thread.currentThread()}, numit: '${threadLocal.get()}'")
 yield()
 println("Dupa yield, sunt in: ${Thread.currentThread()}, numit: '${threadLocal.get()}'")
 }
 job.join()
 println("Dupa ce am oprit thread-urile interne sunt in: ${Thread.currentThread()}, numit:
'${threadLocal.get()}'")
}

si executia codului
Pre-main, current thread: Thread[main,5,main], numit: 'thread-ul cu prisina:'
Sunt acum in: Thread[DefaultDispatcher-worker-2,5,main], numit: 'launch'
Dupa yield, sunt in: Thread[DefaultDispatcher-worker-2,5,main], numit: 'launch'
Dupa ce am oprit thread-urile interne sunt in: Thread[main,5,main], numit: 'thread-ul cu prisina:'
```

# Asigurarea coerentei datelor

```
import kotlinx.coroutines.*
import kotlin.system.*
suspend fun CoroutineScope.massiveRun(action: suspend () -> Unit) {
 val n = 100 // numar coroutine care vor fi lansate in executie
 val k = 1000 // numar de repetari a fiecarei corutine
 val time = measureTimeMillis {
 val jobs = List(n)
 { launch { repeat(k) { action() } } }
 jobs.forEach { it.join() }
 }
 println("S-au efectuat ${n * k} operatii in $time ms")
}
val mtContext = newFixedThreadPoolContext(2, "mtPool") // se defineste un context explicit numai cu 2 fire
var counter = 0
fun main() = runBlocking<Unit> {
 CoroutineScope(mtContext).massiveRun {
 // se va folosi mt... in loc de Dispatchers.Default pentru a forta aparitia fenomenului
 counter++ //variabila comună unde vor apărea erori
 }
 println("Numarator = $counter")
}
```

**Si un exemplu de executie**

S-au efectuat 100000 operatii in 28 ms  
Numarator = 90497

# Soluții specifice

```
import java.util.concurrent.atomic.*
.....
var counter = AtomicInteger() si rezultatul executiei
.....
GlobalScope.massiveRun {
 counter.incrementAndGet()
}
println("Numarator = ${counter.get()}")
Am efectuat 100000 sarcini in 29 ms
Numarator = 100000
```

## Izolare cu granularitate mică/fină a firelor

```
GlobalScope.massiveRun {
 // desi fiecare corutina este executata cu DefaultDispatcher
 withContext(counterContext) {
 // fiecare operatie pe variabila este limitata la firul unic dedicat
 counter++
 }
}
println("Numarator = $counter")
```

**si rezultatul executiei**  
Am terminat 100000 sarcini in 569 ms  
Numarator = 100000

## Izolarea cu granularitate mare a firelor

```
CoroutineScope(counterContext).massiveRun {
 // se executa fiecare corutina intr-un context cu thread unic
 counter++
}
println("Numarator = $counter")
```

**si rezultatul executiei**

Am terminat 100000 sarcini in 27 ms  
Numarator = 100000

## Soluția bazată pe excluziunea mutuală

```
GlobalScope.massiveRun {
 mutex.withLock {
 counter++
 }
}
```

withLock echivalenta cu

```
mutex.lock();
```

```
try {
```

```
...
```

```
}
```

```
finally { mutex.unlock() }
```

**si rezultatul executiei**

Am terminat 100000 sarcini in 218 ms

Numarator = 100000

# Actori

```
//tipuri de mesaj pentru counterActor
sealed class CounterMsg
object IncCounter : CounterMsg() // mesaj pentru incrementare
class GetCounter(val response: CompletableDeferred<Int>) : CounterMsg() // o cerere cu raspuns
acum vom defini o functie care va lansa un actor prin intermediul unui constructie specific
fun CoroutineScope.counterActor() = actor<CounterMsg> {
 var counter = 0 // actor state
 for (msg in channel) { // iterate over incoming messages
 when (msg) {
 is IncCounter -> counter++
 is GetCounter -> msg.response.complete(counter)
 }
 }
}
iar in codul de baza
val counter = counterActor() // creez the actor
GlobalScope.massiveRun { counter.send(IncCounter) }
// send a message to get a counter value from an actor
val response = CompletableDeferred<Int>()
counter.send(GetCounter(response))
println("Counter = ${response.await()}")
counter.close() // termin actorul
```

## Si rezultatul executiei

Am terminat 100000 operatii in 248 ms  
Numarator = 100000

# Async

```
fun <T> CoroutineScope.async(
 context: CoroutineContext = EmptyCoroutineContext,
 start: CoroutineStart = CoroutineStart.DEFAULT,
 block: suspend CoroutineScope.() -> T
) : Deferred<T> (source)
```

# Async - exemplu utilizare

```
import kotlinx.coroutines.*
import java.text.SimpleDateFormat
import java.util.*
fun main() = runBlocking {
 val deferred1 = async { computation1() }
 val deferred2 = async { computation2() }
 printCurrentTime("Astept efectuarea calculelor...")
 val result = deferred1.await() + deferred2.await()
 printCurrentTime("Valoarea calculata este $result")
}

suspend fun computation1(): Int {
 delay(1000L) // simulam durata primei operatii
 printCurrentTime("Am terminat de calculat prima valoare")
 return 131
}

suspend fun computation2(): Int {
 delay(2000L)//simulam durata celui de-al doilea calcul
 printCurrentTime("Am terminat al doilea calcul")
 return 9
}

fun printCurrentTime(message: String) {
 val time = (SimpleDateFormat("hh:mm:ss")).format(Date())
 println("[${time}] $message")
}
```

## și rezultatul programului

```
[07:49:59] Astept efectuarea calculelor...
[07:50:00] Am terminat de calculat prima valoare
[07:50:01] Am terminat al doilea calcul
[07:50:01] Valoarea calculata este 140
```

## **Produce - este încă în dezvoltare**

```
@ExperimentalCoroutinesApi fun <E> CoroutineScope.produce(
 context: CoroutineContext = EmptyCoroutineContext,
 capacity: Int = 0,
 block: suspend ProducerScope<E>.() -> Unit
) : ReceiveChannel<E> (source)
```

# Deadlock

```
import kotlinx.coroutines.*

lateinit var jobA : Job
lateinit var jobB : Job

fun main(args: Array<String>) = runBlocking {
 jobA = launch {
 println("Sunt in A")
 jobB.join()
 println("S-a terminat B")
 }
 jobB = launch {
 println("Sunt in B")
 jobA.join()
 println("Sa terminat A")
 }
}
```

## Si exemplu de executie

Sunt in A

Sunt in B

Process finished with exit code 130 (interrupted by signal 2:  
SIGINT) (oprit manual)

## Determinarea stării unui job

| Starea    | isActive | isCompleted | isCanceled |
|-----------|----------|-------------|------------|
| Created   | false    | false       | false      |
| Active    | true     | false       | false      |
| Canceled  | false    | true        | true       |
| Completed | false    | true        | false      |

# Optimizare în funcție de numărul core

```
import kotlin.system.*
import kotlinx.coroutines.*
fun main(args: Array<String>) = runBlocking {
 println("${Thread.activeCount()} fire de executie active la pornire")
 val time = measureTimeMillis{
 createCoroutines(10_0)
 }
 println("${Thread.activeCount()} fire de executie active la sfarsit")
 println("Procesul a durat $time ms")
}

suspend fun createCoroutines(amount: Int) {
 val backgroundPool: CoroutineDispatcher by lazy {
 val numProcessors = Runtime.getRuntime().availableProcessors()
 when {
 numProcessors <= 2 -> newFixedThreadPoolContext(2, "background")
 else -> newFixedThreadPoolContext(numProcessors, "background")
 }
 }
 val jobs = ArrayList<Job>()
 for (i in 1..amount) {
 jobs += GlobalScope.launch(backgroundPool) {
 println("Am pornit $i in
${Thread.currentThread().name}")
 delay(1000)
 println("S-a terminat $i din
${Thread.currentThread().name}")
 }
 }
 jobs.forEach {
 it.join()
 }
}
```

**si rezultat executie**

2 fire de executie active la pornire  
Am pornit 1 in background-1  
Am pornit 2 in background-2  
....  
S-a terminat 69 din background-8  
S-a terminat 98 din background-3  
S-a terminat 100 din background-2  
10 fire de executie active la sfarsit  
Procesul a durat 1019 ms

## Canale

```
val channel = RendezvousChannel<Int>()
```

- fara parametru

```
val rendezvousChannel = Channel<Int>()
```

- similară cu

```
val rendezvousChannel = Channel<Int>(0)
```

- ca efect dar aceasta din urmă poate avea o altă capacitate a tamponului

```
val rendezvousChannel = Channel<Int>(30)
```

# Utilizare canale de comunicare

```
import kotlin.system.*
import kotlinx.coroutines.*
import kotlinx.coroutines.channels.*

fun main(args: Array<String>) = runBlocking {
 val time = measureTimeMillis {
 val channel = Channel<Int>()
 val sender = launch {
 repeat(10) {
 channel.send(it) //am trimis 10 bucati pe canal
 println("Am trimis $it")
 }
 }
 for (i in 1..10) {
 channel.receive() // am primit 10 bucati din canal
 }
 }
 println("Procesul a durat ${time}ms")
}
```

**si exemplul de executie**

Am trimis 0  
Am trimis 1  
Am trimis 2  
Am trimis 3  
Am trimis 4  
Am trimis 5  
Am trimis 6  
Am trimis 7  
Am trimis 8  
Procesul a durat 13ms  
Am trimis 9

Process finished with exit code 0

# Canale neblocante

```
import kotlin.system.*
import kotlinx.coroutines.*
import kotlinx.coroutines.channels.*

fun main(args: Array<String>) = runBlocking {
 val time = measureTimeMillis {
 val channel = Channel<Int>(Channel.UNLIMITED)
 val sender = launch {
 repeat(10) {
 channel.send(it) //am trimis 10 bucati pe canal
 println("Am trimis $it")
 }
 }
 for (i in 1..8) {
 println(channel.receive()) // am primit 8 bucati din canal
 }
 }
 println("Procesul a durat ${time}ms")
}
```

si rezultatul executiei

Am trimis 0  
Am trimis 1  
Am trimis 2  
Am trimis 3  
Am trimis 4  
Am trimis 5  
Am trimis 6  
Am trimis 7  
Am trimis 8  
Am trimis 9 //nepreluate  
0  
1  
2  
3  
4  
5  
6  
7  
Procesul a durat 13ms  
Process finished with exit code 0

# ConflatedChannel

```
import kotlin.system.*
import kotlinx.coroutines.*
import kotlinx.coroutines.channels.*

fun main(args: Array<String>) = runBlocking {
 val time = measureTimeMillis {
 val channel = Channel<Int>(Channel.CONFLATED)
 launch {
 repeat(5) {
 channel.send(it)
 println("Am trimis $it")
 }
 }
 for (i in 1..5) {
 val element = channel.receive()
 println("Am primit $element")
 } //comentati for-ul si nu se va bloca
 }
 println("Procesul a durat ${time}ms")
}
```

rezultat cu for activ

```
Am trimis 0
Am trimis 1
Am trimis 2
Am trimis 3
Am trimis 4
Am primit 0
Am primit 4
```

Process finished with exit  
code 130 (interrupted by  
signal 2: SIGINT)

cu for comentat

```
Am trimis 0
Am trimis 1
Am trimis 2
Am trimis 3
Am trimis 4
Am primit 0
Procesul a durat 12ms
```

Process finished with exit  
code 0

# Thread-uri Kotlin

```
fun thread(
 start: Boolean = true,
 isDaemon: Boolean = false,
 contextClassLoader: ClassLoader? = null,
 name: String? = null,
 priority: Int = -1,
 block: () -> Unit
): Thread //definiția din bibliotecă
• și un program de test:
import kotlin.concurrent.*
fun main(args: Array<String>){
 thread(start = true) {
 println("Thread Kotlin ${Thread.currentThread()} s-a executat.")
 }
}
```

# Controlul thread-urilor din Java

```
fun main(args: Array<String>){
 object : Thread() {
 override fun run() {
 println("Sunt in thread-ul singleton ${Thread.currentThread()}")
 }
 }.start()
 val t1=SimpleThread()
 t1.run()
 val t2=SimpleRunnable()
 t2.run()
 val thread = Thread {
 println("Thread lambda ${Thread.currentThread()} s-a executat.")
 }
 thread.start()
}
class SimpleThread: Thread() {
 public override fun run() {
 println("Instanta clasei derivate din Thread ${Thread.currentThread()} s-a executat.")
 }
}
class SimpleRunnable: Runnable {
 public override fun run() {
 println("Instanta clasei care implementeaza Runnable ${Thread.currentThread()} s-a executat.")
 }
}
```

Sunt in thread-ul singleton Thread[Thread-0,5,main]  
Instanta clasei derivate din Thread Thread[main,5,main] s-a executat.  
Instanta clasei care implementeaza Runnable Thread[main,5,main] s-a executat.  
Thread lambda Thread[Thread-2,5,main] s-a executat.

Process finished with exit code 0

# Metodă elegantă de apel thread

```
import java.lang.Thread.*\n\nfun main(args: Array<String>){\n val thread1 = thread(start = true, name = "Speedy", priority = MAX_PRIORITY) {\n println("Threadul ${Thread.currentThread()} s-a executat.")\n }\n val thread2 = thread(start = true, name = "Turtle", priority = MIN_PRIORITY) {\n println("Threadul ${Thread.currentThread()} s-a executat.")\n }\n}\n\npublic fun thread(start: Boolean = true, isDaemon: Boolean = false, contextClassLoader: ClassLoader? = null, name: String? = null, priority: Int = -1, block: () -> Unit): Thread {\n val thread = object : Thread() {\n public override fun run(){\n block()\n }\n }\n if (isDaemon)\n thread.isDaemon = true\n if (priority > 0)\n thread.priority = priority\n if (name != null)\n thread.name = name\n if (contextClassLoader != null)\n thread.contextClassLoader = contextClassLoader\n if (start)\n thread.start()\n return thread\n}
```

## si rezultatul executiei

Threadul Thread[Speedy,10,main] s-a executat.  
Threadul Thread[Turtle,1,main] s-a executat.

Process finished with exit code 0

# Utilizarea funcțiilor/blocurilor cu excluziune mutuală

```
import java.util.concurrent.*
fun main(args: Array<String>){
 val g=gigel()
 val executor = Executors.newFixedThreadPool(5)
 for(i in 0..9){
 val worker= Runnable{
 println("Sunt in firul "+i)
 g.synchronizedMethod()
 g.methodWithSynchronizedBlock()
 }
 executor.execute(worker)
 }
 executor.shutdown()
 while (!executor.isTerminated){
 }
 println("S-au terminat toate firele din piscina")
}
class gigel{
 @Synchronized
 fun synchronizedMethod(){
 println("Sunt in metoda sincronizata ${Thread.currentThread()}")
 }
 fun methodWithSynchronizedBlock(){
 println("Zona fara sincronizare: ${Thread.currentThread()}")
 synchronized(this){
 println("Sectiune cu sincronizare: ${Thread.currentThread()}")
 }
 }
}
```

## si exemplu parcial de iesire

Sunt in firul 0  
Sunt in firul 1  
Sunt in metoda sincronizata Thread[pool-1-thread-1,5,main]  
Zona fara sincronizare: Thread[pool-1-thread-1,5,main]  
Sunt in metoda sincronizata Thread[pool-1-thread-2,5,main]  
Zona fara sincronizare: Thread[pool-1-thread-2,5,main]  
Sectiune cu sincronizare: Thread[pool-1-thread-1,5,main]  
Sunt in firul 2  
Sunt in firul 3  
Sunt in firul 4  
Sunt in metoda sincronizata Thread[pool-1-thread-5,5,main]  
Sunt in firul 5  
Zona fara sincronizare: Thread[pool-1-thread-5,5,main]  
Sunt in metoda sincronizata Thread[pool-1-thread-4,5,main]  
Zona fara sincronizare: Thread[pool-1-thread-4,5,main]  
Sunt in metoda sincronizata Thread[pool-1-thread-3,5,main]  
Zona fara sincronizare: Thread[pool-1-thread-3,5,main]  
Sectiune cu sincronizare: Thread[pool-1-thread-2,5,main]  
Sectiune cu sincronizare: Thread[pool-1-thread-3,5,main]  
....  
S-au terminat toate firele din piscina

# Variabile comună inter-thread - @Volatile

```
import java.util.concurrent.*
fun main(args: Array<String>){
 val executor = Executors.newFixedThreadPool(5)
 for (i in 0..4) {
 val worker = Runnable {
 println("Sunt in firul " + i)
 println(faceceva.inc())
 }
 executor.execute(worker)
 }
 executor.shutdown()
 while (!executor.isTerminated) {
 }
 println("S-au terminat toate firele din piscina")
}
object faceceva {
 @Volatile
 private var i = 0
 fun inc(): Int {
 i+=1
 return i
 }
}
```

**si exemplul de executie**

|                                        |  |
|----------------------------------------|--|
| Sunt in firul 0                        |  |
| Sunt in firul 1                        |  |
| Sunt in firul 2                        |  |
| Sunt in firul 3                        |  |
| 1                                      |  |
| 4                                      |  |
| 2                                      |  |
| 3                                      |  |
| Sunt in firul 4                        |  |
| 5                                      |  |
| S-au terminat toate firele din piscina |  |
| Process finished with exit code 0      |  |

# Metodele Wait & Notify

- **wait()**

- Apelată de un obiect

- **notifyAll()**

- Apelată de un obiect

- Trebuie să aibă deja controlul asupra lock-ului respectivului obiect.

# wait(), notify() and notifyAll() la piață

```
import java.util.*
import kotlin.concurrent.thread
class TaraNoasean(private val maxItems: Int) {
 @Volatile private var items = 0
 private val rand = Random()
 private val lock = java.lang.Object()
 fun produce() = synchronized(lock) {
 while (items >= maxItems) {
 lock.wait()
 }
 items++
 println("Am produs $items alimente în ${Thread.currentThread()}")
 lock.notifyAll()
 }
 fun consume() = synchronized(lock) {
 while (items <= 0) {
 lock.wait()
 }
 items--
 println("Am utilizat $items alimente în ${Thread.currentThread()}")
 lock.notifyAll()
 }
}
fun main(args: Array<String>) {
 println("Starting: ${Thread.currentThread()}")

 val example = TaraNoasean(5)

 for (i in 0..14) {
 thread {
 if (i < 5) {
 example.consume()
 } else {
 example.produce()
 }
 }
 println("S-a închis piață: ${Thread.currentThread()}")
 }
}
```

## **Exemplu simplu reflecție Java**

```
fun main(args: Array<String>){
 val s = "Hello world"
 val length = s.javaClass.getMethod("length")
 val x = length.invoke(s) as Int
 println(x)
}
```

## Exemplu de reflexie generică Kotlin - proprietăți

```
fun main(args: Array<String>){
 val prop = Person::name
 print(prop)
}

class Person(val name: String, var age: Int) {
 fun present() = "Sunt $name, si am $age ani"
 fun greet(other: String) = "Salut, $other, sunt $name"
}

Immutable KProperty1<R, V>,
mutable KMutableProperty1<R, V>.

si iesirea
val Person.name: kotlin.String
Process finished with exit code 0
```

# Reflexie la nivel de instanță

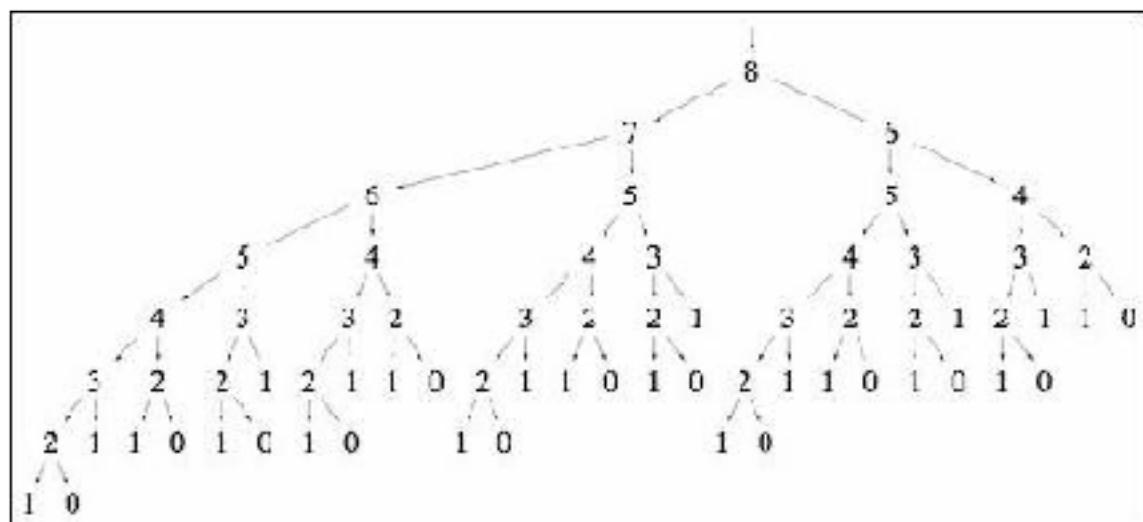
```
fun main(args: Array<String>){//inspectie instanta Kotlin
 val person = Person("Lisa", 23)
 println(person.present())
 printProperty(person, Person::name)
 incrementProperty(person, Person::age)
 println(person.present())
}

class Person(val name: String, var age: Int) {
 fun present() = "Sunt $name, si am $age ani"
 fun greet(other: String) = "Salut, $other, sunt $name"
}
fun <T> printProperty(instance: T, prop: KProperty1<T, *>) {
 println("${prop.name} = ${prop.get(instance)}")
}
fun <T> incrementProperty(instance: T, prop: KMutableProperty1<T, Int>) {
 val value = prop.get(instance)
 prop.set(instance, value + 1)
}
```

**Si ieșirea programului**  
Sunt Lisa, si am 23 ani  
name = Lisa  
Sunt Lisa, si am 24 ani  
Process finished with exit code 0

## Memoization

```
import kotlin.system.*\n\nfun main(args: Array<String>) {\n val time = measureTimeMillis {\n println(fib(30))\n }\n println("Procesul a durat ${time}ms")\n}\n\nfun fib(k: Int): Long = when (k) {\n 0 -> 1\n 1 -> 1\n else -> fib(k - 1) + fib(k - 2)\n}
```



# Memoizare Fibbonaci cu map

```
import kotlin.system.*
fun main(args: Array<String>) {
 val k=8
 val time = measureTimeMillis {
 println("Fibbonaci($k)="+ memfib(k).toString())
 }
 println("Procesul a durat ${time}ms")
 println(map)
}
val map = mutableMapOf<Int, Long>()
fun memfib(k: Int): Long {
 return map.getOrPut(k) {
 when (k) {
 0 -> 1
 1 -> 1
 else -> memfib(k - 1) + memfib(k - 2)
 }
 }
}
```

**si rezultatul executiei**  
Fibbonaci(8)=34  
Procesul a durat 1ms  
{1=1, 0=1, 2=2, 3=3, 4=5, 5=8, 6=13, 7=21, 8=34}  
Process finished with exit code 0

# Memoizare generalizată

```
fun <A, R> memoize(fn: (A) -> R): (A) -> R {
 val map = ConcurrentHashMap<A, R>()
 return { a ->
 map.getOrPut(a) { fn(a) }
 }
}
```

și o versiune înbunătățită:

```
fun <A, R> Function1<A, R>.memoized(): (A) -> R {
 val map = ConcurrentHashMap<A, R>()
 return {
 a -> map.getOrPut(a) { this.invoke(a) }
 }
}
val memquery = ::query.memoized()
```

## **Alias de tip**

```
typealias Cache = HashMap<String, Boolean>
```

- sau

```
fun process(exchange: Exchange<HttpRequest, HttpResponseMessage>):
Exchange<HttpRequest, HttpResponseMessage>
```

- se poate înlocui cu:

```
typealias HttpExchange = Exchange<HttpRequest, HttpResponseMessage>
```

```
fun process2(exchange: HttpExchange): HttpExchange
```

- sau

```
typealias Width = Int
```

```
typealias Length = Int
```

```
typealias Height = Int
```

```
fun volume(width: Width, length: Length, height: Height): Int
```

## **Either**

```
sealed class Either<out L, out R>
 class Left<out L>(value: L) : Either<L, Nothing>()
 class Right<out R>(value: R) : Either<Nothing, R>()
```

## **Fold**

```
sealed class Either<out L, out R> {
 fun <T> fold(lfn: (L) -> T, rfn: (R) -> T): T = when (this) {
 is Left -> lfn(this.value)
 is Right -> rfn(this.value)
 }
}
```

# *Paradigma Sequentiala versus Concurrency*

Cursul nr. 11  
Mihai Zaharia

# Cum se face un logging mai serios

```
def alg_complacat(items):# ex1
 for i, item in enumerate(items):
 # corpul alg
 logger.debug('%s iteration, item=%s', i, item)
def handle_request(request):#ex2
 logger.info('Gestionez cererea %s', request)
 # tratare cerere
 result = 'result'
 logger.info('Rezultatul este: %s', result)
def start_service():
 logger.info('Pornesc serviciul pe portul %s ...', port)
 service.start()
 logger.info('Serviciul a pornit')
def authenticate(user_name, password, ip_address):# ex 3
 if user_name != USER_NAME and password != PASSWORD:
 logger.warn('Incercare esuata de intrare in sistem utilizator %s de la IP %s', user_name, ip_address)
 return False
 # executarea autentificarii
def get_user_by_id(user_id):
 user = db.read_user(user_id)
 if user is None:
 logger.error('Nu hasesc utilizatorul cu user_id=%s', user_id)
 return user
 return user
```

# Cum se face un logging mai serios

```
try: #ex 1
 open('/path/to/does/not/exist', 'rb')
except (SystemExit, KeyboardInterrupt):
 raise
except Exception, e:
 logger.error('Nu am putut deschide fisierul', exc_info=True)
import logging

def foo():#ex 2
 logger = logging.getLogger(__name__)
 logger.info('Hi, foo')
class Bar(object):
 def __init__(self, logger=None):
 self.logger = logger or logging.getLogger(__name__)
 def bar(self):
 self.logger.info('Hi, bar')
```

# Cum se face un logging mai serios

```
logging.json
{
 "version": 1,
 "disable_existing_loggers": false,
 "formatters": {
 "simple": {
 "format": "%(asctime)s - %(name)s - %(levelname)s -\n%(message)s"
 }
 },
 "handlers": {
 "console": {
 "class": "logging.StreamHandler",
 "level": "DEBUG",
 "formatter": "simple",
 "stream": "ext://sys.stdout"
 },
 "info_file_handler": {
 "class": "logging.handlers.RotatingFileHandler",
 "level": "INFO",
 "formatter": "simple",
 "filename": "info.log",
 "maxBytes": 10485760,
 "backupCount": 20,
 "encoding": "utf8"
 }
 }
}
```

```
 "error_file_handler": {
 "class": "logging.handlers.RotatingFileHandler",
 "level": "ERROR",
 "formatter": "simple",
 "filename": "errors.log",
 "maxBytes": 10485760,
 "backupCount": 20,
 "encoding": "utf8"
 }
},
"loggers": {
 "my_module": {
 "level": "ERROR",
 "handlers": ["console"],
 "propagate": false
 }
},
"root": {
 "level": "INFO",
 "handlers": ["console", "info_file_handler", "error_file_handler"]
}
}
pentru a incarca acest fisier dintr-o cale prestabilita
LOG_CFG=my_logging.json python my_server.py
```

## **Python threading module**

- pentru thread
- pentru Lock
- pentru RLock
- pentru semafoare
- pentru condiții
- pentru evenimente

# Analiza comparativă - diverse biblioteci pentru paralelism

```
import threading
import multiprocessing
from concurrent.futures import ThreadPoolExecutor
import time

def countdown():
 x = 100000000
 while x > 0:
 x -= 1

def ver_1():#pseudoparalelism
 thread_1 = threading.Thread(target=countdown)
 thread_2 = threading.Thread(target=countdown)
 thread_1.start()
 thread_2.start()
 thread_1.join()
 thread_2.join()

def ver_2():#sequential
 countdown()
 countdown()

def ver_3():#paralelism cu multiprocesing
 process_1 = multiprocessing.Process(target=countdown)
 process_2 = multiprocessing.Process(target=countdown)
 process_1.start()
 process_2.start()
 process_1.join()
 process_2.join()

def ver_4():#paralelism cu concurrent.futurez
 with ThreadPoolExecutor(max_workers=2) as executor:
 future = executor.submit(countdown())
 future = executor.submit(countdown())
```

```
if __name__ == '__main__':
 start = time.time()
 ver_1()
 end = time.time()
 print("\n Timp executie pseudoparalelism cu GIL")
 print(end - start)
 start = time.time()
 ver_2()
 end = time.time()
 print("\n Timp executie sequential")
 print(end - start)
 start = time.time()
 ver_3()
 end = time.time()
 print("\n Timp executie paralela cu multiprocessing")
 print(end - start)
 start = time.time()
 ver_4()
 end = time.time()
 print("\n Timp executie paralela cu concurrent.futures")
 print(end - start)
```

## si rezultatul executiei

Timp executie pseudoparalelism cu GIL - 13.273755550384521

Timp executie sequential - 10.081993579864502

**Timp executie paralela cu multiprocessing - 5.0672242641448975**

Timp executie paralela cu concurrent.futures - 13.14623498916626

## **Un fir de execuție**

```
class threading.Thread(group=None,
 target=None,
 name=None,
 args=(),
 kwargs={})
```

## **Exemplu utilizare parametri funcție în fir**

```
import threading
```

```
def function(i):
 print('Functia este apelata de firul %i\n' % i)

threads = []
for i in range(5):
 t = threading.Thread(target=function, args=(i,))
 threads.append(t)
 t.start()
 t.join()
```

# Determinarea firului curent

```
import threading
import logging
logging.basicConfig(level=logging.INFO)
def first_function():
 logging.info(threading.currentThread().getName() + str(' porneste...'))
 logging.info(threading.currentThread().getName() + str(' se opreste...'))
 return
def second_function():
 logging.info(threading.currentThread().getName() + str(' porneste...'))
 logging.info(threading.currentThread().getName() + str(' se opreste...'))
 return
def third_function():
 logging.info(threading.currentThread().getName() + str(' porneste...'))
 logging.info(threading.currentThread().getName() + str(' se opreste...'))
 return
if __name__ == '__main__':
 t1 = threading.Thread(name='prima_functie', target=first_function)
 t2 = threading.Thread(name='a doua functie', target=second_function)
 t3 = threading.Thread(name='a treia functie', target=third_function)
 t1.start()
 t2.start()
 t3.start()
 logging.debug('Pauza')
 t1.join()
 t2.join()
 t3.join()
 logging.info(threading.currentThread().getName() + str(' - main thread...'))
```

## si rezultatul executiei

```
INFO:root:prima_functie porneste...
INFO:root:prima_functie se opreste...
INFO:root:a doua functie porneste...
INFO:root:a treia functie porneste...
INFO:root:a doua functie se opreste...
INFO:root:a treia functie se opreste...
INFO:root:MainThread - main thread...
```

Process finished with exit code 0

# Utilizarea unui fir într-o subclasă

```
import threading
import time
EXIT_FLAG = 0
class Firisor(threading.Thread):
 def __init__(self, thread_id, name, counter):
 threading.Thread.__init__(self)
 self.thread_id = thread_id
 self.name = name
 self.counter = counter
 def run(self):
 print('Sunt %s și am pornit' % self.name)
 print_time(self.name, self.counter, 5)
 print('Sunt %s și am terminat\n' % self.name)
 def print_time(thread_name, delay, counter):
 while counter:
 if EXIT_FLAG:
 thread.exit()
 time.sleep(delay)
 print('%s: %s' % (thread_name, time.ctime(time.time())))
 counter -= 1
thread1 = Firisor(1, 'Firul 1', 1)
thread2 = Firisor(2, 'Firul 2', 2)
thread1.start()
thread2.start()
thread1.join()
thread2.join()
print('S-a terminat firul principal')
```

## și rezultatul executiei

```
/home/bugs/PycharmProjects/fir in subclasa/venv/bin/python
"/home/bugs/PycharmProjects/fir in subclasa/fir in subclasa.py"
Sunt Firul 1 și am pornit
Sunt Firul 2 și am pornit
Firul 1: Sun Apr 7 13:30:44 2019
Firul 1: Sun Apr 7 13:30:45 2019
Firul 2: Sun Apr 7 13:30:45 2019
Firul 1: Sun Apr 7 13:30:46 2019
Firul 2: Sun Apr 7 13:30:47 2019
Firul 1: Sun Apr 7 13:30:47 2019
Firul 1: Sun Apr 7 13:30:48 2019
Sunt Firul 1 și am terminat
Firul 2: Sun Apr 7 13:30:49 2019
Firul 2: Sun Apr 7 13:30:51 2019
Firul 2: Sun Apr 7 13:30:53 2019
Sunt Firul 2 și am terminat
S-a terminat firul principal
Process finished with exit code 0
```

# Exemplu de utilizare lock()

```
import threading
contor_cu_lock = 0
contor_fara_lock = 0
COUNT = 1000000
lock_contor = threading.Lock()
def safe_inc():
 global contor_cu_lock
 for _ in range(COUNT):
 lock_contor.acquire()
 contor_cu_lock += 1
 lock_contor.release()
def safe_dec():
 global contor_cu_lock
 for _ in range(COUNT):
 lock_contor.acquire()
 contor_cu_lock -= 1
 lock_contor.release()
def unsafe_inc():
 global contor_fara_lock
 for _ in range(COUNT):
 contor_fara_lock += 1
```

```
def unsafe_dec():
 global contor_fara_lock
 for _ in range(COUNT):
 contor_fara_lock += 1
if __name__ == '__main__':
 t1 = threading.Thread(target=safe_inc)
 t2 = threading.Thread(target=safe_dec)
 t3 = threading.Thread(target=unsafe_dec)
 t4 = threading.Thread(target=unsafe_inc)
 t1.start()
 t2.start()
 t3.start()
 t4.start()
 t1.join()
 t2.join()
 t3.join()
 t4.join()
 print('variabila comună gestionată cu lock', contor_cu_lock)
 print('variabila comună gestionată fără lock', contor_fara_lock)
```

**si rezultatul executiei**

|            |                                               |
|------------|-----------------------------------------------|
| t4.start() | variabila comună gestionată cu lock 0         |
| t1.join()  | variabila comună gestionată fără lock 1322023 |

# Exemplu utilizare RLock()

```
import threading
import time
class Cutiechibrituri(object):
 lock = threading.RLock()
 def __init__(self):
 self.total_chibrituri = 0
 def execute(self, n):
 Cutiechibrituri.lock.acquire()
 self.total_chibrituri += n
 Cutiechibrituri.lock.release()
 def pun(self):
 Cutiechibrituri.lock.acquire()
 self.execute(1)
 Cutiechibrituri.lock.release()
 def scot(self):
 Cutiechibrituri.lock.acquire()
 self.execute(-1)
 Cutiechibrituri.lock.release()
 def pune(Cutiechibrituri, chibrituri):
 while chibrituri > 0:
 print('Pun un chibrit in Cutiechibrituri')
 Cutiechibrituri.pun()
 time.sleep(1)
 chibrituri -= 1
def scot(Cutiechibrituri, chibrituri):
 while chibrituri > 0:
 print('Scot un chibrit din Cutiechibrituri')
 Cutiechibrituri.scot()
 time.sleep(1)
 chibrituri -= 1
 if __name__ == '__main__':
 chibrituri = 5
 print('Pun', chibrituri, 'chibrituri in Cutiechibrituri')
 Cutiechibrituri = Cutiechibrituri()
 t1 = threading.Thread(target=pune, args=(Cutiechibrituri,
 chibrituri))
 t2 = threading.Thread(target=scot, args=(Cutiechibrituri,
 chibrituri))
 t1.start()
 t2.start()
 t1.join()
 t2.join()
 print('mai sunt', Cutiechibrituri.total_chibrituri, 'chibrituri in
 Cutiechibrituri')
```

# Exemplu semafoare

```
import threading
import time
import random
semafor = threading.Semaphore(0)
def consumator():
 print('Consumatorul in asteptare')
 semafor.acquire()
 print('Consumatorul in asteptare')
 print('Consumatorul in asteptare si a produs ', element, ' elemente')
def producator():
 global element
 time.sleep(1)#simulare complexitate operatiuni in caz real
 element = random.randint(0, 1000)
 print('Producatorul in asteptare si a produs ', element, ' elemente')
 semafor.release()
if __name__ == '__main__':
 for i in range(5):
 t1 = threading.Thread(target=producator)
 t2 = threading.Thread(target=consumator)
 t1.start()
 t2.start()
 t1.join()
 t2.join()
```

**si un exemplu de executie**

Consumatorul in asteptare  
Producatorul a fost anuntat si a produs 398 elemente  
Consumatorul in asteptare  
....  
Consumatorul in asteptare  
Producatorul a fost anuntat si a produs 701 elemente  
Consumatorul in asteptare  
....  
Consumatorul in asteptare  
Producatorul a fost anuntat si a produs 701 elemente

# Fir cu Condiție

```
from threading import Thread, Condition
import time
elemente = []
conditie = Condition()
class Consumator(Thread):
 def __init__(self):
 Thread.__init__(self)
 def consumator(self):
 global conditie#utilizarea variabilelor globale
NERECOMANDATA in caz real
 global elemente
 conditie.acquire()
 if len(elemente) == 0:
 conditie.wait()
 print('mesaj de la consumator: nu am nimic disponibil')
 elemente.pop()
 print('mesaj de la consumator : am utilizat un element')
 print('mesaj de la consumator : mai am disponibil',
len(elemente), 'elemente')
 conditie.notify()
 conditie.release()
 def run(self):
 for i in range(5):
 self.consumator()
```

```
class Producator(Thread):
 def __init__(self):
 Thread.__init__(self)
 def producator(self):
 global conditie
 global elemente
 conditie.acquire()
 if len(elemente) == 10:
 conditie.wait()
 print('mesaj de la producator : am disponibile',
len(elemente), 'elemente')
 print('mesaj de la producator : am oprit productia')
 elemente.append(1)
 print('mesaj de la producator : am produs',
len(elemente), 'elemente')
 conditie.notify()
 conditie.release()
 def run(self):
 for i in range(5):
 self.producator()
if __name__ == '__main__':
 producator = Producator()
 consumator = Consumator()
 producator.start()
 consumator.start()
 producator.join()
 consumator.join()
```

# Fir cu eveniment

```
import time
from threading import Thread, Event
import random
elemente = []
eveniment = Event()
class Consumator(Thread):
 def __init__(self, elemente, eveniment):
 Thread.__init__(self)
 self.elemente = elemente
 self.eveniment = eveniment
 def run(self):
 for i in range(5):
 self.eveniment.wait()
 try:
 item = self.elemente.pop()
 except IndexError:
 print('Nu pot scoate dintr-o coada goala!')
 print('\nMesaj de la consumator : %d a fost generat de %s' % (item, self.name))
```

```
class Producator(Thread):
 def __init__(self, elemente, eveniment):
 Thread.__init__(self)
 self.elemente = elemente
 self.eveniment = eveniment
 def run(self):
 for i in range(5):
 item = random.randint(0, 256)
 self.elemente.append(item)
 print('\nMesaj de la producator : elementul # %d a fost adaugat la lista de %s' % (item, self.name))
 print('Mesaj de la producator : eveniment generat de %s' % self.name)
 self.eveniment.set()
 print('Mesaj de la producator : eveniment anulat de %s' % self.name)
 self.eveniment.clear()
 if __name__ == '__main__':
 t1 = Producator(elemente, eveniment)
 t2 = Consumator(elemente, eveniment)
 t1.start()
 t2.start()
 t1.join()
 t2.join()
```

# Utilizarea 'with'

```
import threading
import logging
logging.basicConfig(
 level=logging.DEBUG,
 format='%(threadName)-8s %(message)s',
)
def thread_cu_with(statement):
 with statement:
 logging.debug('%s achiziționat cu with' % statement)
def thread_fara_with(statement):
 statement.acquire()
 try:
 logging.debug('%s achiziționat direct' % statement)
 finally:
 statement.release()
if __name__ == '__main__':
 lock = threading.Lock()
 rlock = threading.RLock()
 conditie = threading.Condition()
 mutex = threading.Semaphore(1)
 threading_synchronisation_list = [lock, rlock, conditie, mutex]
 for statement in threading_synchronisation_list:
 t1 = threading.Thread(target=thread_cu_with, args=(statement,))
 t2 = threading.Thread(target=thread_fara_with, args=(statement,))
 t1.start()
 t2.start()
 t1.join()
 t2.join()
```

## si rezultatul executiei

```
(Thread-1) <locked _thread.lock object at 0x7fc8cce9dcb0> achiziționat cu with
(Thread-2) <locked _thread.lock object at 0x7fc8cce9dcb0> achiziționat direct
(Thread-3) <locked _thread.RLock object owner=140500325627648 count=1 at
0x7fc8ccdbb390> achiziționat cu with
(Thread-4) <locked _thread.RLock object owner=140500325627648 count=1 at
0x7fc8ccdbb390> achiziționat direct
(Thread-5) <Condition(<locked _thread.RLock object owner=140500325627648 count=1
at 0x7fc8ccdbb420>, 0)> achiziționat cu with
(Thread-6) <Condition(<locked _thread.RLock object owner=140500406716160 count=1
at 0x7fc8ccdbb420>, 0)> achiziționat direct
(Thread-7) <threading.Semaphore object at 0x7fc8ccd56320> achiziționat cu with
(Thread-8) <threading.Semaphore object at 0x7fc8ccd56320> achiziționat direct
```

# Comunicare inter-thread utilizând cozi

```
from threading import Thread
from queue import Queue
import time
import random
class Producator(Thread):
 def __init__(self, queue):
 Thread.__init__(self)
 self.queue = queue
 def run(self):
 for i in range(10):
 element = random.randint(0, 256)
 self.queue.put(element)
 print('Mesaj de la producator : element N%d adaugat la coada de %s\n' %
 (element, self.name))
 time.sleep(1)
class Consumator(Thread):
 def __init__(self, queue):
 Thread.__init__(self)
 self.queue = queue
```

```
def run(self):
 while True:
 element = self.queue.get()
 print('Mesaj de la consumator : %d scos din coada de %s' %
 (element, self.name))
 self.queue.task_done()
if __name__ == '__main__':
 queue = Queue()
 t1 = Producator(queue)
 t2 = Consumator(queue)
 t3 = Consumator(queue)
 t4 = Consumator(queue)
 t1.start()
 t2.start()
 t3.start()
 t4.start()
 t1.join()
 t2.join()
 t3.join()
 t4.join()
```

# Paralelism real - multiprocessing

```
import multiprocessing
import time
def proces_gol():
 nume = multiprocessing.current_process().name
 print('\nPornesc un proces numit: %s' % nume)
 time.sleep(3)#simulez o executie
 print('Am terminat procesul numit: %s' % nume)
if __name__ == '__main__':
 proces_demon = multiprocessing.Process(
 name='proces demon', target=proces_gol)
 proces_demon.daemon = True
 proces_normal = multiprocessing.Process(
 name='proces normal', target=proces_gol)
 proces_normal.daemon = False
 proces_demon.start()
 proces_normal.start()
 print('am terminat procesul normal')
```

**si rezultatul executiei**

am terminat procesul normal

Pornesc un proces numit: proces demon

Pornesc un proces numit: proces normal

Am terminat procesul numit: proces normal

Process finished with exit code 0

# gestiunea stării curente a unui proces

```
import multiprocessing
import time
import signal
def proces_gol():
 print('Pornesc executia procesului')
 time.sleep(0.1)
 print('S-a terminat executia procesului')
if __name__ == '__main__':
 proces_test = multiprocessing.Process(target=proces_gol)
 print('Starea procesului inainte de lansarea in executie:', proces_test, proces_test.is_alive())
 proces_test.start()
 print('Procesul se executa:', proces_test, proces_test.is_alive())
 proces_test.terminate()
 try:
 print('Procesul s-a terminat:', proces_test, proces_gol().is_alive())
 except AttributeError:
 print('Nu exista informatii dupa comanda terminare')
 proces_test.join()
 try:
 print('Procesul dupa join:', proces_test, proces_gol().is_alive())
 except AttributeError:
 print('Nu am informatii dupa join')
 if signal.SIG_DFL == proces_test.exitcode:
 print('Procesul dupa un exit code')
```

**si rezultatul executiei**

Starea procesului inainte de lansarea in executie:  
<Process(Process-1, initial)> False

Procesul se executa: <Process(Process-1, started)> True

Pornesc executia procesului

S-a terminat executia procesului

Nu exista informatii dupa comanda terminare

Pornesc executia procesului

S-a terminat executia procesului

Nu am informatii dupa join

Process finished with exit code 0

# utilizarea unui proces in subclasă

```
import multiprocessing
class ProcesTest(multiprocessing.Process):
 def run(self):
 print ('am apelat metoda run() in procesul: %s' %self.name)
 return

 if __name__ == '__main__':
 jobs = []

 for i in range(5):
 p = ProcesTest()
 jobs.append(p)
 p.start()
 p.join()

 si rezultatul executiei
 am apelat metoda run() in procesul: ProcesTest-1
 am apelat metoda run() in procesul: ProcesTest-2
 am apelat metoda run() in procesul: ProcesTest-3
 am apelat metoda run() in procesul: ProcesTest-4
 am apelat metoda run() in procesul: ProcesTest-5

 Process finished with exit code 0
```

# Cozi pentru comunicare interproces

```
import multiprocessing
import random
class Producator(multiprocessing.Process):
 def __init__(self, queue):
 multiprocessing.Process.__init__(self)
 self.queue = queue
 def run(self):
 for _ in range(10):
 element = random.randint(0, 256)
 self.queue.put(element)
 print('Proces Producator : elementul %d s-a
addaugat in coada %s' % (element, self.name))
 print('Dimensiunea cozii este %s' %
self.queue.qsize())
class Consumator(multiprocessing.Process):
 def __init__(self, queue):
 multiprocessing.Process.__init__(self)
 self.queue = queue
```

```
def run(self):
 while True:
 if self.queue.empty():
 print('Coada este goala')
 break
 else:
 element = self.queue.get()
 print('Proces Consumator : elementul %d a
fost scos din %s\n' % (element, self.name))

if __name__ == '__main__':
 queue = multiprocessing.Queue()
 proces_producator = Producator(queue)
 proces_consumator = Consumator(queue)
 proces_producator.start()
 proces_consumator.start()
 proces_producator.join()
 proces_consumator.join()
```

# Comunicare utilizând pipe

```
import multiprocessing

def creare_elemente(pipe):
 pipe_iesire, _ = pipe
 for element in range(4):
 pipe_iesire.send(element)
 pipe_iesire.close()

def multiply_elements(pipe1, pipe2):
 close, pipe_intrare = pipe1
 close.close()
 pipe_iesire, _ = pipe2
 try:
 while True:
 element = pipe_intrare.recv()
 print('am primit în pipe1:', element)
 x = element * element
 pipe_iesire.send(x)
 print('am trimis în pipe2:', x)
 except EOFError:
 pipe_iesire.close()

if __name__ == '__main__':
 # primul pipe cu elemente de la 0 la 9
 pipe1 = multiprocessing.Pipe(True)
 process_pipe1 = multiprocessing.Process(
 target=creare_elemente, args=(pipe1,))
 process_pipe1.start()
 # al doilea pipe
 pipe2 = multiprocessing.Pipe(True)
 process_pipe2 = multiprocessing.Process(
 target=multiply_elements, args=(pipe1, pipe2))
 process_pipe2.start()
 pipe1[0].close()
 pipe2[0].close()
 try:
 while True:
 print('Am scos elementul:', pipe2[1].recv())
 except EOFError:
 print('End')
```

# Sincronizarea proceselor

- Lock
- Event
- Condition
- Semaphore
- RLock
- Barrier

# Exemplu simplu de apel barieră

```
import multiprocessing
from multiprocessing import Barrier, Lock,
Process
from time import time
import datetime as dt
def test_bariera(barrier, lock):
 name =
multiprocessing.current_process().name
 barrier.wait()
 now = time()
 with lock:
 print('Procesul %s ----> %s' % (name,
dt.datetime.fromtimestamp(now)))
def test_fara_bariera():
 name =
multiprocessing.current_process().name
 now = time()
 print('Procesul %s ----> %s' % (name,
dt.datetime.fromtimestamp(now)))
```

```
if __name__ == '__main__':
 barrier = Barrier(2)
 lock = Lock()
 Process(name='p1 - test_cu_bariera',
 target=test_bariera,
 args=(barrier, lock)).start()
 Process(name='p2 - test_cu_bariera',
 target=test_bariera,
 args=(barrier, lock)).start()
 Process(name='p3 - test_fara_bariera',
 target=test_fara_bariera).start()
 Process(name='p4 - test_fara_bariera',
 target=test_fara_bariera).start()
```

## si rezultatul executiei

```
Procesul p2 - test_cu_bariera ----> 2019-04-10 07:38:19.407982
Procesul p1 - test_cu_bariera ----> 2019-04-10 07:38:19.408019
Procesul p3 - test_fara_bariera ----> 2019-04-10 07:38:19.408314
Procesul p4 - test_fara_bariera ----> 2019-04-10 07:38:19.408933
Process finished with exit code 0
```

# Gestiunea stărilor între procese

```
import multiprocessing as mp
def worker(dictionary, cheie, element, contor):
 lock=mp.Lock()#trebuie?
 with lock:
 contor[0]=contor[0]+1
 dictionary[cheie] = element
 print('Cheie:', cheie, 'laloare:', element, 'sunt la al', contor[0], '-lea apel')
if __name__ == '__main__':
 manager = mp.Manager() # handler de variabila comună
 dictionary = manager.dict()
 contor = manager.list([0])
 contor[0] = 0
 sarcini = [mp.Process(target=worker, args=(dictionary, i, i * 2, contor))
 for i in range(5)]
 for treaba in sarcini:
 treaba.start()
 for treaba in sarcini:
 treaba.join()
 print('Rezultate:', dictionary)
```

## si rezultatul executiei

```
Cheie: 0 laloare: 0 sunt la al 1 -lea apel
Cheie: 1 laloare: 2 sunt la al 2 -lea apel
Cheie: 2 laloare: 4 sunt la al 2 -lea apel
Cheie: 3 laloare: 6 sunt la al 3 -lea apel
Cheie: 4 laloare: 8 sunt la al 4 -lea apel
Rezultate: {0: 0, 1: 2, 2: 4, 3: 6, 4: 8}
```

# Utilizarea unui pool de procese

```
import multiprocessing as mp
import time as tm
import signal
import sys
def la_patrat(data):
 #tm.sleep(10000)
 val = data*data
 return val
def signal_handler(signal, frame):
 print('a aparut o operatie externa', signal)
 pool.terminate()
 pool.join()
 print('am terminat fortat procesul')
 sys.exit(0)
if __name__ == '__main__':
 intrari = list(range(10))
 pool = mp.Pool(processes=4)
 signal.signal(signal.SIGINT, signal_handler)
 calcul_pool = pool.map(la_patrat, intrari)
 pool.close()
 pool.join()
 print('Pool:', calcul_pool)
```

**si rezultatul executiei**

Pool: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

Process finished with exit code 0

# Cozi în multiprocessing

```
import multiprocessing
import os
from multiprocessing import Queue
q = Queue()
def proc_pid(n):
 q.put(os.getpid())#linux pid
 print("\n[{0}] Salut!".format(n))
procese = []
for i in range(5):
 t = multiprocessing.Process(target=proc_pid, args=(i,))
 procese.append(t)#creez un pool de procese
 t.start()
for un_proces in procese:
 print(un_proces.name) #utilizez referinta la fiecare proces
 un_proces.join()
olista = []
while not q.empty():
 olista.append(q.get())#scot din coada
print(olista,'de lungimea',len(olista))
```

si rezultatul executiei

```
[0] Salut!
[1] Salut!
[2] Salut!
Process-1
Process-2
Process-3
[3] Salut!
Process-4
[4] Salut!
Process-5
[28746, 28747, 28748, 28750, 28753]
de lungimea 5
Process finished with exit code 0
```

## **Concurrent.future**

- concurrent.futures.Executor:
- submit (function ,argument):
- map (function,argument):
- shutdown (Wait = True):
- concurrent.futures.Future:

## **Executors - Gestionari ai execuției**

- concurrent.futures.ThreadPoolExecutor(max\_workers)
- concurrent.futures.ProcessPoolExecutor(max\_workers)

# Reanalizăm performanțele

```
import concurrent.futures as cf
import time
lista_numere = list(range(1, 5))
def numara(numar):
 i = 0
 for i in range(10**7):
 i += 1
 return i*numar
def evaluare(element):
 element_rezultat = numara(element)
 print('element %s, rezultat este %s' % (element,
element_rezultat))
if __name__ == '__main__':
 # secential
 start = time.time()
 tpornire = time.time()
 for element in lista_numere:
 evaluare(element)
 print('Executia secentiala a durat %s secunde' %
(time.time() - tpornire))
```

```
cu pool fire
tpornire = time.time()
with cf.ThreadPoolExecutor(max_workers=5) as
executor:
 for element in lista_numere:
 executor.submit(evaluare, element)
 print('Executia pool-ului de fire a durat %s secunde' %
(time.time() - tpornire))
cu pool procese
tpornire = time.time()
with cf.ProcessPoolExecutor(max_workers=5) as
executor:
 for element in lista_numere:
 executor.submit(evaluare, element)
 print('Executia pool-ului de procese a durat %s
secunde' % (time.time() - tpornire))

extras din rezultat de executie
Executia secentiala a durat 2.10807728767395 secunde
Executia pool-ului de fire a durat 3.307506799697876 secunde
Executia pool-ului de procese a durat 0.5447263717651367 secunde
Process finished with exit code 0
```

## **Gestiunea evenimentelor cu Asyncio**

- Event loop:
- Coroutines:
- Futures:
- Tasks:

## **Metode specifice gestiunii buclei de evenimente**

- `loop = get_event_loop()`:
- `loop.call_later(time_delay,callback,argument)`:
- `loop.call_soon(callback,argument)`:
- `loop.time()`:
- `asyncio.set_event_loop()`:
- `asyncio.new_event_loop()`:
- `loop.run_forever()`:

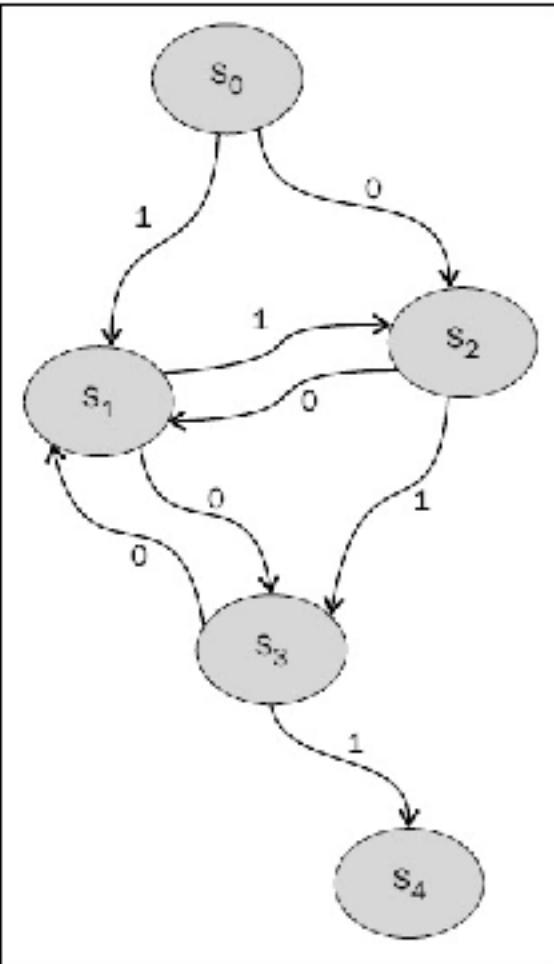
# Un prim exemplu de utilizare asyncio

```
import asyncio as asy
def functia_1(tstop, bucla):
 print('functia 1 apelata')
 if (bucla.time() + 1.0) < tstop:
 bucla.call_later(1, functia_2, tstop, bucla)
 else:
 bucla.stop()
def functia_2(tstop, bucla):
 print('functia 2 apelata')
 if (bucla.time() + 1.0) < tstop:
 bucla.call_later(1, functia_3, tstop, bucla)
 else:
 bucla.stop()
```

```
def functia_3(tstop, bucla):
 print('functia 3 apelata')
 if (bucla.time() + 1.0) < tstop:
 bucla.call_later(1, functia_1, tstop, bucla)
 else:
 bucla.stop()
if __name__ == '__main__':
 bucla_asincrona = asy.get_event_loop()
 sfarsit_bucla = bucla_asincrona.time() + 6.0
 bucla_asincrona.call_soon(functia_1,
 sfarsit_bucla, bucla_asincrona)
 bucla_asincrona.run_forever()
 bucla_asincrona.close()
```

```
functia 1 apelata
functia 2 apelata
functia 3 apelata
functia 1 apelata
functia 2 apelata
functia 3 apelata
Process finished with exit code 0
```

# Coroutine



Să testăm următorul program cu  
sevența de intrări prezentate mai jos

**Start**  $\rightarrow S_0 \rightarrow I_0=0$

$\downarrow$

**S<sub>2</sub>**  $\rightarrow I_2=0$

$\downarrow$

**S<sub>1</sub>**  $\rightarrow I_1=0$

$\downarrow$

**S<sub>3</sub>**  $\rightarrow I_3=1$

$\downarrow$

**S<sub>4</sub>**  $\rightarrow \text{Stop}$

# Și programul ...

```
import asyncio as asy
@asy.coroutine
def stare0_start():
 print('Start din S0 \n')
 valoare_intrare = int(input('Valoare Intrare in S0='))
 if valoare_intrare == 0:
 rezultat = yield from stare2(valoare_intrare)
 else: rezultat = yield from stare1(valoare_intrare)
@asy.coroutine
def stare1(valoare_tranzitie):
 valoare_jesire = 'stare S1 cu valoare de intrare = %s\n' %
 valoare_tranzitie
 valoare_intrare = int(input('Valoare Intrare in S1='))
 print('...S1 - calculez...')
 if valoare_intrare == 0:
 rezultat = yield from stare3(valoare_intrare)
 else: rezultat = yield from stare2(valoare_intrare)
 return valoare_jesire + 'apel stare S1 cu %s' % rezultat
@asy.coroutine
def stare2(valoare_tranzitie):
 valoare_jesire = 'stare S2 cu valoare de tranzitie = %s\n' %
 valoare_tranzitie
 valoare_intrare = int(input('Valoare Intrare in S2='))
 print('...S2 - calculez...')
```

```
if valoare_intrare == 0:
 rezultat = yield from stare1(valoare_intrare)
else: rezultat = yield from stare3(valoare_intrare)
return valoare_jesire + 'apel stare 2 cu %s' % rezultat
@asy.coroutine
def stare3(valoare_tranzitie):
 valoare_jesire = 'stare S3 cu valoare de tranzitie = %s\n' %
 valoare_tranzitie
 valoare_intrare = int(input('Valoare Intrare in S3='))
 print('...S3 - calculez...')
 if valoare_intrare == 0:
 rezultat = yield from stare1(valoare_intrare)
 else: rezultat = yield from stare4_stop(valoare_intrare)
 return valoare_jesire + 'apel stare 3 cu %s' % rezultat
@asy.coroutine
def stare4_stop(valoare_tranzitie):
 print('...S4 - calculez...')
 valoare_jesire = 'sfarsit stare with tranzitie value = %s\n' %
 valoare_tranzitie
 print('...Oprire...')
 return valoare_jesire
if __name__ == '__main__':
 print('Executie FSM utilizand asyncio si Corutine [I
reprezinta intrarea in stare]')
 buda = asy.get_event_loop()
 buda.run_until_complete(stare0_start())
```

# Gestiune task-uri cu asyncio

```
import asyncio as asy
@asy.coroutine
def factorial(number):
 fact = 1
 for i in range(2, number + 1):
 print('Calculez factorial(%s)' % i)
 yield from asy.sleep(1)
 fact *= i
 print(' factorial(%s) = %s' % (number, fact))
@asy.coroutine
def fibonacci(number):
 a, b = 0, 1
 for i in range(number):
 print('Calculez fibonacci(%s)' % i)
 yield from asy.sleep(1)
 a, b = b, a + b
 print(' fibonacci(%s) = %s' % (number, a))
```

```
@asy.coroutine
def coeficient_binomial(n, k):
 rezultat = 1
 for i in range(1, k + 1):
 rezultat = rezultat*(n - i + 1)/i
 print('Calculez coeficientul binomial(%s)' % i)
 yield from asy.sleep(1)
 print(' coeficientul binomial(%s, %s) = %s' % (n,
k, rezultat))
if __name__ == '__main__':
 tasks = [asy.Task(factorial(7)),
 asy.Task(fibonacci(7)),
 asy.Task(coeficient_binomial(14, 7))]
 bucla = asy.get_event_loop()
 bucla.run_until_complete(asy.wait(tasks))
 bucla.close()
```

# Asyncio și Futures

- instanțiere obiect future

```
import asyncio
```

```
future = asyncio.Future()
```

- metodele acestei clase sunt următoarele:

- cancel():
- result():
- exception():
- add\_done\_callback(fn):
- remove\_done\_callback(fn):
- set\_result(result):
- set\_exception(exception):



shutterstock.com • 109991456

# Şi un exemplu de utilizare

```
import asyncio as asy
@asy.coroutine
def prima_corutina(future, numar):
 contor = 0
 for i in range(1, numar + 1):
 contor += 1
 yield from asy.sleep(1)
 future.set_result('In prima corutina calculez
suma a %s numere = %s' % (numar, contor))
@asy.coroutine
def a_doua_corutina(future, numar):
 contor = 1
 for i in range(2, numar + 1):
 contor *= i
 future.set_result('In a doua corutina calculez
factorial(%s) = %s' % (numar, contor))
def preiau_rezultatul(future):
 print(future.result())
```

```
if __name__ == '__main__':
 numar1 = int(input('Numarul 1 = '))
 numar2 = int(input('Numarul 2 = '))
 bucla = asy.get_event_loop()
 future1 = asy.Future()
 future2 = asy.Future()
 tasks = [prima_corutina(future1, numar1),
 a_doua_corutina(future2, numar2)]
 future1.add_done_callback(preiau_rezultatul)
 future2.add_done_callback(preiau_rezultatul)
 bucla.run_until_complete(asy.wait(tasks))
 bucla.close()
```

## Şi exemplu de utilizare

```
Numarul 1 = 10
Numarul 2 = 10
In a doua corutina calculez factorial(10) = 3628800
In prima corutina calculez suma a 10 numere = 10
Process finished with exit code 0
```

# Tratarea canalelor de comunicare cu socket

- Deschiderea unui socket la server pentru recepționare date

```
canal_comunicare = socket(AF_INET, SOCK_STREAM)
canal_comunicare.bind((serverHost, serverPort))
canal_comunicare.listen(3)
conn, addr = canal_comunicare.accept()
date_prumite = conn.recv(1024)
```

- Pentru început se crează canalul de comunicare pe server cu  
socket(family, type [, proto]),
- care ne oferă un canal de comunicare\_din\_categorii *family*.
- Categorii de canale de comunicare specifice bibliotecii socket din python:

| Categorie | Descriere                |
|-----------|--------------------------|
| AF_INET   | Protocol Ipv4 (TCP, UDP) |
| AF_INET6  | Protocol Ipv6 (TCP, UDP) |
| AF_UNIX   | Protocoale de domeniu    |

## Tratarea canalelor de comunicare

- Tipuri de canale de comunicare din biblioteca socket

| Tip            | Descriere                                                                          |
|----------------|------------------------------------------------------------------------------------|
| SOCK_STREAM    | Deschide un fisier existent pentru citire.                                         |
| SOCK_DGRAM     | Deschide un fisier pentru scriere. Cu suprascriere.                                |
| SOCK_RAW       | Deschide un fisier existent pentru adăugări sau modificări                         |
| SOCK_RDM       | Deschide un fisier atât pentru scriere cât și pentru citire, nu face suprascriere. |
| SOCK_SEQPACKET | Deschide un fisier atât pentru scriere cât și pentru citire, face suprascriere.    |

# Gestiune simplă a rețelei

```
import socket
def afla_informatii_despre_masina_locala():
 nume_masina = socket.gethostname()
 adresa_ip = socket.gethostbyname(nume_masina)
 print("Numele sistemului local: %s" % nume_masina)
 print("Adresa de IP a sistemului local: %s" % adresa_ip)
def afla_informatii_despre_masina_la_distanta(nume):
 try:
 print("Adresa IP a masinii cu numele %s: %s" % (nume, socket.gethostbyname(nume)))
 except socket.error as err_msg:
 print("%s: %s" % (nume, err_msg))
if __name__ == '__main__':
 afla_informatii_despre_masina_locala()
 afla_informatii_despre_masina_la_distanta('www.tuiasi.ro')
```

## și rezultatul executiei

Numele sistemului local: home  
Adresa de IP a sistemului local: 127.0.1.1  
Adresa IP a masinii cu numele www.tuiasi.ro:  
81.180.223.65

Process finished with exit code 0

# Caut servicii

```
import socket as sk

def caut_servicii(nume_protocol, port_list):
 for port in port_list:
 try:
 s=sk.getservbyport(port, nume_protocol)
 print("Pe portul: %s am serviciul cu numele %s care utilizeaza protocolul %s" %(port,
s,nume_protocol))
 except: continue

if __name__ == '__main__':
 port_list =[1,80,8080]
 nume_protocol = 'udp'
 caut_servicii(nume_protocol,port_list)
 nume_protocol = 'tcp'
 caut_servicii(nume_protocol,port_list)
```

# Măruntișuri

```
import socket as sk
DIM_BUFF_SEND = 4096
DIM_BUFF_RECV = 4096
def test_socket_timeout():
 s = sk.socket(sk.AF_INET, sk.SOCK_STREAM)
 print("Timp maxim de asteptare: %s" % s.gettimeout())
 s.settimeout(100)
 print("Timpul de asteptare curent: %s" % s.gettimeout())
def modific_dim_buffer(trimitere,receptie):
 sock = sk.socket(sk.AF_INET, sk.SOCK_STREAM)
 bufsize = sock.getsockopt(sk.SOL_SOCKET, sk.SO_SNDBUF)
 print("Dimensiune tampon inainte de modificare:%d" % bufsize)
 sock.setsockopt(sk.SOL_TCP, sk.TCP_NODELAY, 1)
 sock.setsockopt(sk.SOL_SOCKET, sk.SO_SNDBUF, trimitere)
 sock.setsockopt(sk.SOL_SOCKET, sk.SO_RCVBUF, receptie)
 bufsize = sock.getsockopt(sk.SOL_SOCKET, sk.SO_SNDBUF)
 print("Dimensiune tampon dupa de modificare:%d" % bufsize)
if __name__ == '__main__':
 test_socket_timeout()
 modific_dim_buffer(DIM_BUFF_SEND,DIM_BUFF_RECV)
```

## si rezultatul executiei

Timp maxim de asteptare: None  
Timpul de asteptare curent: 100.0  
Dimensiune tampon inainte de modificare:16384  
Dimensiune tampon dupa de modificare:8192

Process finished with exit code 0

# Aplicații client server – serverul

```
from socket import *
serverHost = "" # asculta pe toate interfetele de retea
serverPort = 8888
canal_comunicare_server = socket(AF_INET, SOCK_STREAM)
canal_comunicare_server.bind((serverHost, serverPort))
canal_comunicare_server.listen(3)
while True:
 conexiune,addr = canal_comunicare_server.accept()
 print("Conexiune cu un client:", addr)
 while True:
 data = conexiune.recv(1024)
 if not data:
 break
 print('Serverul a primit:', repr(data))
 conexiune.sendall(data)
 conexiune.close()
```

## **Aplicatii client server – clientul**

```
import sys
from socket import *
serverHost = ""
serverPort = 8888
mesaj = ['Mesajul unu de la client', 'Mesajul doi']
if len(sys.argv) > 1:
 serverHost = sys.argv[1]
canal_comunicare_client = socket(AF_INET, SOCK_STREAM)
canal_comunicare_client.connect((serverHost, serverPort))
for element in mesaj:
 canal_comunicare_client.sendall(element.encode())
 date_receptionate = socketul_client.recv(1024)
 print("Clientul a primit:", date_receptionate)
canal_comunicare_client.close()
```

## **Aplicatii client server – server cu socketserver**

```
import socketserver
HOST, PORT = "", 8889
```

```
class MyTCPRequestHandler(socketserver.StreamRequestHandler):
 def handle (self):
 self.date_primit = self.request.recv(1024).strip()
 print("{} a trimis:".format(self.client_address[0]))
 print(self.date_primit)
 self.request.sendall(self.date_primit.upper())
server = socketserver.TCPServer((HOST,PORT),MyTCPRequestHandler)

server.serve_forever()
```

## Şi un client pentru serverul anterior

```
import socket
HOST, PORT = "", 8889
date_test = "Test de emisie receptie\nsi a doua linie"
canal_comunicare = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
 canal_comunicare.connect((HOST, PORT))
 canal_comunicare.sendall(bytes(date_test + "\n", "utf-8"))
 date_receptionate = str(canal_comunicare.recv(1024), "utf-8")
finally:
 canal_comunicare.close()
print("Am trimis la sever: {}".format(date_test))
print("Am primit de la server: {}".format(date_receptionate))
```

# Server Ecou utilizând TCP-IP bazat pe asyncio

```
import asyncio as asy
class ServerEcou(asy.Protocol):
 def connection_made(self, transport):
 nume_client =
 transport.get_extra_info('nume client')
 print("Conexiune acceptata cu
{}".format(nume_client))
 self.transport = transport
 def data_received(self, date):
 mesaj = date.decode()
 print("Date Receptionate:
{}".format(mesaj))
 print("Trimt: {!r}".format(mesaj))
 self.transport.write(date)
 print("Inchid socket-ul clientului")
 self.transport.close()
```

```
bucla = asy.get_event_loop()
c1 = bucla.create_server(ServerEcou, '127.0.0.1',
8888)
server = bucla.run_until_complete(c1)
try:
 bucla.run_forever()
except KeyboardInterrupt:
 pass
server.close()
bucla.run_until_complete(server.wait_closed())
bucla.close()
```

# Client Ecou utilizând TCP-IP bazat pe asyncio

```
import asyncio as asy
class ClientEcou(asy.Protocol):
 def __init__(self, mesaj, bucla):
 self.mesaj = mesaj
 self.bucla = bucla
 def connection_made(self, transport):
 transport.write(self.mesaj.encode())
 print('Date trimise:
{!r}'.format(self.mesaj))
 def data_received(self, data):
 print("Date primite:
{!r}".format(data.decode()))
 def connection_lost(self, exc):
 print("serverul a terminat conexiunea")
 print("Opreire bucla de evenimente")
 self.bucla.stop()
bucla = asy.get_event_loop()
mesaj = 'Am trimis ceva'
c = bucla.create_connection(lambda:
ClientEcou(mesaj, bucla), '127.0.0.1',
8888)
bucla.run_until_complete(c)
bucla.run_forever()
bucla.close()
```

# *Paradigma Calcului Funcțional*

Cursul nr. 12  
Mihai Zaharia

**Cum a început.... (au fost o dată ca niciodată ...)**  
**ipoteza Church**

**și conducătorii lor  
de doctorat ...**

**ipoteza Turing**

# Java

# vs

# Kotlin

```
public class JavraLambda //ex 1
{ interface Test
 { void test(); }
private static void apel_test(Test test)
 { test.test(); }
public static void main(String[] args)
 { apel_test(()->{ System.out.println("apel metoda
apel_test()"); }); }
}
fun apel_test(test:()->Unit) //ex 2
{ test() }
fun main(args: Array<String>
{ apel_test{ println("apel functie apel_test()");}} }
```

## **Utilizarea unei funcții ca o proprietate**

```
fun main(args: Array<String>)
{
 val dif_numere = { x: Int, y: Int -> x - y }
 println("Diferenta 1 este ${dif_numere(33,66)}")
 println("Diferenta 2 este ${dif_numere(77,11)}")
}
```

# Sintaxa specifică funcțiilor Lambda în Kotlin

```
fun main(args: Array<String>) {
 val invers:(Int)->Int
 invers = {numar ->
 var n = numar
 var numarInvers = 0
 while (n>0) {
 val digit = n%10
 numarInvers=numarInvers*10+digit
 n/=10
 }
 numarInvers
 }
 println("inversul lui 123 ${invers(123)}")
 println("inversul lui 456 ${invers(456)}")
 println("inversul lui 789 ${invers(789)}")
}
```

**si exemplul de executie**  
inversul lui 123 321  
inversul lui 456 654  
inversul lui 789 987

Process finished with exit code 0

# Funcții de nivel superior

```
fun procesareNrPare(numar:Int,procesare:(Int)->Int):Int {
 if(numar%2==0) {
 return procesare(numar)
 } else {
 return numar
 }
}
fun main(args: Array<String>) {
 var nr1=4
 var nr2 = 5
 println("Apel cu ${nr1} si operatia (it*2): ${procesareNrPare(nr1,{it*2})}")
 println("Apel cu ${nr2} si operatia (it*2): ${procesareNrPare(nr2, {it*2})}")
}
```

## Funcții de nivel superior

```
fun apelCulntoarcere(n:Int):(String)->Char {
 return { it[n] }
}
fun main(args: Array<String>) {
 var pos = 4
 print("${apelCulntoarcere(1)("abc")}\n")
 print("${apelCulntoarcere(0)("def")}\n")
 try {
 print(apelCulntoarcere(pos)("ghi"))
 }
 catch (e: StringIndexOutOfBoundsException) {
 print("Indexul ${pos} este in afara sirului")
 }
}
```

## Efecte laterale

```
class operatiiAritmetice {
 var nr1:Int=0
 var nr2:Int=0
 fun suma(nr1:Int = this.nr1,nr2:Int =
this.nr2):Int {
 this.nr1 = nr1
 this.nr2 = nr2
 return nr1+nr2
 }
}
fun main(args: Array<String>) {
 var nr1 = 10
 var nr2 = 15
 val obCalcul = operatiiAritmetice()
 println("Suma dintre ${nr1} si ${nr2} este ${obCalcul.suma(nr1,nr2)}")
}
```

```
class functionalOperatiiAritmetice {
 val sumaf: (Int, Int) -> Int = { x, y -> x + y }
 fun executaOperatia(x: Int, y: Int, op: (Int,
Int) -> Int): Int = op(x, y)
}
fun main(args: Array<String>) {
 var nr1 = 10
 var nr2 = 15
 val obFCalcul =
functionalOperatiiAritmetice()
 println("Suma dintre ${nr1} si ${nr2} este
${obFCalcul.executaOperatia(nr1,nr2,obC
alcul.sumaf)})")
}
```

## Efecte laterale

```
fun main(args: Array<String>) {
 var nr1 = 10
 var nr2 = 15
 val sumaf: (Int, Int) -> Int = { x, y -> x + y } // stil functional
 fun sumak(x: Int, y: Int) = x + y // stil kotlin
 fun executaOperatia(x: Int, y: Int, op: (Int, Int) -> Int): Int = op(x, y)
 println("Suma dintre ${nr1} si ${nr2} este ${executaOperatia(nr1,nr2,sumaf)}")
 println("Suma dintre ${nr1} si ${nr2} este ${executaOperatia(nr1,nr2,:sumak)}")
}
```



## **Funcții pure**

```
fun suma(a:Int = 0,b:Int = 0):Int {
 return a+b
}
```

# vararg

```
fun calculMedie(listaNumere: List<Int>): Float {
 var suma = 0.0f
 for (element in listaNumere) {
 suma += element
 }
 return (suma / listaNumere.size)
}

fun calculMedieParametri(vararg lista_parametri: Int):
Float {
 var suma = 0.0f
 for (element in lista_parametri) {
 suma += element
 }
 return (suma / lista_parametri.size)
}

fun <T> enumerareCaOLista(vararg parametri_intrare:
T): List<T> {
 val rezultat = ArrayList<T>()
 for (element in parametri_intrare)
 rezultat.add(element)
 return rezultat
}
```

```
fun main(args: Array<String>) {
 val tablou = arrayListOf(1, 2, 3, 4)
 val rezultat = calculMedie(tablou)
 print("\nMedia este ${rezultat}")
 val rezultat1 = calculMedieParametri(1, 2, 3)
 print("\nMedia1 este ${rezultat1}")
 val rezultat3 = enumerareCaOLista(1, 2, 3, 4, 5, 6, 7,
8, 9)
 print("\nTransformare enumerare in lista
${rezultat3}")
 print("\nMedia1 este
${calculMedieParametri(*rezultat3.toIntArray())}")
 val tablou1 = intArrayOf(1, 2, 3, 4)
 val rezultat4 = calculMedieParametri(5, 6, 7, 8, 9,
*tablou1)
 print("\nMedia1 este ${rezultat4}")
}
```

# Parametri alias - Named parameters

```
typealias Kg = Double
typealias cm = Int
data class ClientBanca (val numeFamilie: String,
 val numeMijlociu: String,
 val numeMic: String,
 val seriePasaport: String,
 val greutate: Kg,
 val inaltime: cm,
 val semneParticulare: String)
fun main(args: Array<String>) {
 val client1 = ClientBanca("Mike", "Mouse", "Rabbit", "XX234837447", 82.3, 180, "nu are")
 val client2 = ClientBanca(
 numeMic = "Rabbit",
 numeMijlociu = "Mouse",
 numeFamilie = "Mike",
 seriePasaport = "xe4244rf33333",
 greutate = 100.0,
 inaltime = 180,
 semneParticulare = "nu are"
)
 print("\n${client1}")
 print("\n${client2}")
}
```

# **alias cu vararg sau funcții de nivel superior**

```
fun paramDupaVararg(nrCurs: Int, vararg lista_studenti: String, tempCamera: Double) {//ex1
 //corful functiei
}

paramDupaVararg(688, "Gica", "Bula", "Andreea", "Veorica", tempCamera = 15.0)//apel

fun test(f: (Int, String) -> Unit) {//ex2
 f(1, "Bula")
} //cu apelul

test { q, w ->
 //procesare
} //poate fi rescrisa ca

fun test(f: (nume:String, varsta:Int) -> Unit) {
 f("Strula", 10)
} //daca incerc insa ca mai jos

fun test(f: (nume:String, varsta:Int) -> Unit) {
 f(nume = "kati", virsta = 3,) //eroare de compilare}
```

# Funcții de extensie

```
fun String.trimitLaConsola() = println(this) // la tip
class Om(val nume: String)
fun Om.spune(): String = "${this.nume} spune Vai" //la clasa
fun main(args: Array<String>) {
 "IA examen".trimitLaConsola()
 val x=Om("Bula")
 x.spune().trimitLaConsola()
}
```

**și rezultatul executiei**

```
IA examen
Bula spune Vai
```

Process finished with exit code 0

# Funcții de extensie și funcții membre

```
open class Canina {
 open fun vorbeste() = "Un animal din clasa Caninelor face: ham ham!" }
fun mesajScris(canina: Canina) {
 println(canina.vorbeste())
}
class Caine : Canina() {
 override fun vorbeste() = "Un caine face: vauf vauf!" //modificare comportament in clasa derivata
}
fun mesajScris1(canina: Canina) {
 println(canina.vorbeste1())
}
fun Caine.vorbeste()="Din functia extensie Un caine face haf haf" //fiind functie de extensie nu supraincarca!!!
deci este ignorata!!!
fun Canina.vorbeste1() = "functie extensie specifica cainelui"
fun main(args: Array<String>) {
 mesajScris(Canina())
 mesajScris(Caine()) //baza polimorfismului
 mesajScris1(Caine()) //desi este definita la nivelul clasei canina prin mosternire poate fi apelata si in caine
}
```

# Funcții de extensie și funcții membre

```
open class Primata(val name: String)
fun Primata.vorbeste() = "$name: uhaha uhaha"
open class MaimutaMare(name: String) : Primata(name)
fun MaimutaMare.vorbeste() = "${this.name} : urlet"
fun mesajScris(primata: Primata) {
 println(primata.vorbeste())
}
fun mesajScris1(aimutza: MaimutaMare) {
 println(aimutza.vorbeste())
}
fun main(args: Array<String>) {
 mesajScris(Primata("alex")) // apeleaza vorbeste din primata
 mesajScris(MaimutaMare("crrr")) // apeleaza vorbeste din primata
 mesajScris1(MaimutaMare("ciii")) // apeleaza vorbeste din maimuta
 mesajScris1(Primata("jiii") as MaimutaMare) // apeleaza vorbeste din maimutaeroare nu pot
converti primata la maimuta
}
```

# Dispatch recv

```
open class Felina
open class Pisica():Felina()
open class Primata(val name: String)
fun Primata.vorbeste() = "$name: face uhaha uhaha"//extensie primate
open class MaimutaMare(name: String) : Primata(name)
fun MaimutaMare.vorbeste() = "${this.name} : urlet" //ignorata deoarece amm deja un vorbeste
din primata
fun mesajScris(primata: Primata) { println(primata.vorbeste()) }
open class Ingrijitor(val name: String) {
 open fun Felina.react() = "HRRMR!!!"
 fun Primata.react() = "$name se joaca cu ${this@Ingrijitor.name}" //distribuitor intern bagat aici
pentru ca altfel nu am acces la .name
 fun areGrija(felina: Felina) { println("Felina face: ${felina.react()}") }
 fun areGrija(primata: Primata){ println("Primata spune: ${primata.react()}") }
 fun Ingrijitor.react() = "$name din afara clasei se joaca cu ${this@Ingrijitor.name}" //neglijat }
open class Vet(name: String): Ingrijitor(name) { override fun Felina.react() = "fuge de $name" }
```

# Dispatch recv

```
fun main() {
 val om = Ingrijitor("ingrijitorul")
 val pisica = Pisica()
 val maimutaMare = Primata("maimuta")
 val femeie = Vet("corin")
 mesajScris(Primata("alex"))//apel functie din primata
 mesajScris(MaimutaMare("gibon"))//apel functie din primata
 om.areGrija(pisica)//apel functie din felina
 om.areGrija(maimutaMare)//apel dispatcher intern
 listOf(om, femeie).forEach { ingrijitor ->
 println("${ingrijitor.javaClass.getSimpleName} ${ingrijitor.name}")//Vet corin
 ingrijitor.areGrija(pisica)//Felina face: fuge de corin
 ingrijitor.areGrija(maimutaMare)//Primata spune: maimuta se joaca cu
 corin }
}
```

# Funcții de extensie - conflict posibil de nume

```
class Sclav {
 fun munca() = "*munca la birt*"
 private fun odihna() = "*odihna la vecina*"
}

fun Sclav.munca() = "munca cu mila" //neglijata
fun <T> Sclav.munca(t:T) = "* muncesc azi? nuuuu $t*"
fun Sclav.odihna() = "odihna in sant"

fun main()
{
 val sclav = Sclav()
 println(sclav.munca()) //apel f membru
 println(sclav.munca("de maine ma apuc"))
 println(sclav.odihna()) //apel f extensie
}
```

# Funcții de extensie pentru obiecte

```
object Constructor {
}

fun Constructor.casaNouaCaramida() = "casa pe pamant"

class Proiectant {
 companion object {
 }
 object Birou {
 }
}
fun Proiectant.Companion.prototipNou() = "montaj test"
fun Proiectant.Birou.mapaDeLucrari() = listOf("Proiect casa", "Proiect bloc")
fun main()
{
 println(Constructor.casaNouaCaramida())
 println(Proiectant.prototipNou())
 Proiectant.Birou.mapaDeLucrari().forEach(::println)
}
```

# Funcții de extensie pentru liste

```
fun <T> List<T>.tail(): List<T> = this.drop(1)
```

```
infix fun <T> T.prependTo(list: List<T>): List<T> = listOf(this) + list
```

```
fun <T> List<T>.destructured(): Pair<T, List<T>> = first() to tail()
```

```
fun main()
{
 val intregi = listOf(11, 5, 3, 8, 1, 9, 6, 2)
 println(intregi.tail())
 println(intregi.prependTo(intregi))
 println(intregi.destructured())
}
```

# Funcții infix

```
infix fun Int.sumaSmecheraCu(i: Int) = this + i //exemplul 1
fun main() {
 println(4 sumaSmecheraCu 7)
 println(2.sumaSmecheraCu(11))
}

object Toate { // exemplul 2
 infix fun aleTale(base: Pair<Masini, Noua>) {}
}
object Masini {
 infix fun ne(apartin: Apartin) = this
}
object Apartin
object Noua
fun main() {
 println(Toate aleTale (Masini ne Apartin to Noua))
}
```

# Funcția map

```
fun main(args: Array<String>) {
 val lista1 = listOf<Int>(7,15,24,19,8,45,65,55)
 val lista2 = List(10) {
 (1..12).shuffled().first()
 }
 val lista3 = (1..15).map { it }
 val transformareLista = lista1.map { it*2 }
 println("lista 1 =${lista1}")
 println("Transformare lista1 cu map -> $transformareLista")
 println("lista 2 =${lista2}")
 println("lista 3 =${lista3}")
}
```

și rezultatul executiei  
ista 1 =[7, 15, 24, 19, 8, 45, 65, 55]  
Transformare lista1 cu map -> [14, 30, 48, 38, 16, 90, 130, 110]  
lista 2 =[11, 4, 1, 9, 7, 12, 10, 7, 2, 2]  
lista 3 =[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]  
Process finished with exit code 0

# Funcția Filtru - filter

```
import kotlin.math.*
```

```
fun main(args: Array<String>) {
 val lista = 1.until(15).toList()
 val lista1 = (1..15).map { it }
 val sublistaImpare = lista.filter { it%2==0 }
 println("Sublista care contine numai numerele pare este -> $sublistaImpare")
 val listaPatrate = lista1.filter {
 val radacinaPatrata = sqrt(it.toDouble()).roundToInt()
 radacinaPatrata*radacinaPatrata==it //conditia de filtrare
 }
 println("filteredListPSquare -> $listaPatrate")
}
```

și rezultatul executiei

Sublista care contine numai numerele pare este -> [2, 4, 6, 8, 10, 12, 14]  
filteredListPSquare -> [1, 4, 9]

Process finished with exit code 0

# Funcția FlatMap

```
fun main(args: Array<String>) {
 val lista1 = List(10) {
 (26..120).shuffled().first()
 }
 val listaBatutaCuCiocanul = lista1.flatMap {
 it.rangeTo(it+2*it).toList()
 }
 println("Lista dupa aplicarea flat Map este $listaBatutaCuCiocanul")
}
```

și rezultatul executiei

lista de intrare este [80, 35, 76, 85, 58, 42, 43, 98, 97, 110]

Lista dupa aplicarea flat Map este [80, 81, 82, 83, 84, 85, 86, 87, 88, ... mai sunt destule elemente

Process finished with exit code 0

## Funcții pentru tăiere submulțimi - drop

```
fun main(args: Array<String>) {
 val dimLista = 20
 val lista1 = listOf(2, 1, 1, 7, 6, -8, 9, -12)
 val lista = 1.until(dimLista).toList()
 val catTaiStanga = 8
 val catTaiDreapta = 9
 val taiDinStanga = lista.drop(catTaiStanga)
 val taiDinDreapta = lista.dropLast(catTaiDreapta)
 var extragSubinterval = lista.drop(catTaiStanga).dropLast(catTaiDreapta)
 var extragereSelectiva = lista1.dropWhile { e->e>0}
 println("lista originala $lista")
 println("elimin subinterval stang [1..${catTaiStanga}]) -> ${taiDinStanga}")
 println("elimin subinterval drept [dim lista - ${catTaiDreapta}...dim lista] -> ${taiDinDreapta}")
 println("Extragere submultime de la ${catTaiStanga} la ${catTaiDreapta} -> ${extragSubinterval}")
 println("Extragere selectiva pe baza predicat - ${extragereSelectiva}")
}
```

## Funcții pentru extragere din submulțimi - take

- fun main(args: Array<String>) {  
    val lista = 1.until(25).toList()  
    val catlau = 12  
    println("Iau primele \${catlau} din lista -> \${lista.take(catlau)}")  
    println("Iau ultimile \${catlau} din lista -> \${lista.takeLast(catlau)}")  
    println("Iau primele \${catlau} din lista -> \${lista.takeWhile { it<=catlau }}")  
    println("Iau toate elem incepand de la indexul \${catlau} ->  
          \${lista.takeLastWhile { it>=catlau }}")  
}

# Funcția Fermoar - Zip

```
fun main(args: Array<String>) {
 val lista1 = listOf(1,2,3,4,5)
 val lista2 = listOf(
 "element 1",
 "element 2",
 "element 3",
 "element 4",
 "element 5"
)

 val listaRezultat = lista1.zip(lista2)
 println("Utilizare zipWithNext -> ${lista1.zipWithNext()}")
 println(listaRezultat)
}

si rezultatul executiei
Utilizare zipWithNext -> [(1, 2), (2, 3), (3, 4), (4, 5)]
[(1, element 1), (2, element 2), (3, element 3), (4, element
4), (5, element 5)]
```

Process finished with exit code 0

# Gruparea colecțiilor

```
fun main(args: Array<String>) {
 val lista = 1.rangeTo(20).toList()
 println(lista.groupBy { it%3 })
 println(lista.groupingBy { it }.eachCount())
}
```

**și rezultatul executiei**

```
{1=[1, 4, 7, 10, 13, 16, 19], 2=[2, 5, 8, 11, 14, 17, 20], 0=[3, 6, 9, 12, 15, 18]}
{1=1, 2=1, 3=1, 4=1, 5=1, 6=1, 7=1, 8=1, 9=1, 10=1, 11=1, 12=1, 13=1, 14=1, 15=1, 16=1, 17=1, 18=1, 19=1, 20=1}
```

Process finished with exit code 0

# Functori - Functors

```
fun main(args: Array<String>) {
 listOf(1, 2, 3)
 .map { i -> i * 2 }
 .map(Int::toString)
 .forEach(::println)
}
```

# Functori

```
sealed class TipSimplu<out T> {
 object None : TipSimplu<Nothing>() {
 override fun toString() = "cu None"
 }
 data class Some<out T>(val value: T) : TipSimplu<T>()
 companion object
}
fun <T, R> TipSimplu<T>.map(transform: (T) -> R): TipSimplu<R> = when (this) {
 TipSimplu.None -> TipSimplu.None
 is TipSimplu.Some -> TipSimplu.Some(transform(value))
}
fun main(args: Array<String>) {
 println(TipSimplu.Some("Simulare cu Some")
 .map(String::toUpperCase))
```

si rezultatul executiei  
Some(value=SIMULARE CU SOME)

Process finished with exit code 0

# Functori

```
sealed class TipSimplu<out T> {
 object None : TipSimplu<Nothing>() {
 override fun toString() = "None"
 }
 data class Some<out T>(val value: T) : TipSimplu<T>()
 companion object
}
fun <T, R> TipSimplu<T>.map(transform: (T) -> R): TipSimplu<R> = when (this) {
 TipSimplu.None -> TipSimplu.None
 is TipSimplu.Some -> TipSimplu.Some(transform(value))
}
fun main(args: Array<String>) {
 println(TipSimplu.Some("Simulare cu Some").map(String::toUpperCase))
 println(TipSimplu.None.map(String::toUpperCase))
}
```

si rezultatul executiei  
Some(value=SIMULARE CU SOME)  
None  
Process finished with exit code 0

# Functori

Pornim de la:

```
fun <A, B, C> ((A) -> B).map(transformare: (B) -> C): (A) -> C = { t -> transformare(this(t)) }
```

- Și avem două posibile implementări

//versiunea 1

```
typealias FunctieCuInt = (Int) -> Int
infix fun FunctieCuInt.map(g: FunctieCuInt): FunctieCuInt {
 return { x -> this(g(x)) }
}
```

```
val aduna3SilmultesteCu3 = { a: Int -> a + 2 } map { a: Int -> a * 3 }
```

//versiunea 2

```
val aduna5SilmultesteCu5: (Int) -> Int = { i: Int -> i + 3 }.map { j -> j * 2 }
```

```
fun main(args: Array<String>) {
 println(aduna3SilmultesteCu3(33))
 println(aduna5SilmultesteCu5(66))
}
```

# Functori cu liste

```
fun List<Int>.convertLaStr(): List<String> =
 if (size > 0) {
 val ListaNoua = ArrayList<String>(size)
 for (element in this) {
 ListaNoua.add(procesare(procesareLambda(element,5)).toString())
 }
 ListaNoua
 } else {
 emptyList()
 }
val procesareLambda = { x: Int, y: Int -> x - y }
fun procesare(x:Int):Int
{
 return x+5
}
fun main(args: Array<String>) {
 val Listalntregi = listOf(271, 3, 17, 23, 51)
 println(Listalntregi.convertLaStr())
}
```

# Functor cu arbore

```
class ArboreBinar<T>(var value: T) {
 var stanga : ArboreBinar<T>? = null
 var dreapta: ArboreBinar<T>? = null
 fun <U> map(f: (T) -> U): ArboreBinar<U> {
 val arbore = ArboreBinar<U>(f(value))
 if (stanga != null) arbore.stanga = stanga?.map(f)
 if (dreapta != null) arbore.dreapta = dreapta?.map(f)
 return arbore
 }
 fun AfisezParteaSuperioara() = "(${stanga?.value}, $value, ${dreapta?.value})"
}
fun main(args: Array<String>) {
 val barb = ArboreBinar(6)
 barb.stanga = ArboreBinar(20)
 barb.dreapta = ArboreBinar(9)
 println(barb.AfisezParteaSuperioara())
 val barb1 = barb.map { it * 50.0 }
 println(barb1.AfisezParteaSuperioara())
}
```

**si exemplu de executie**

(20, 6, 9)

(1000.0, 300.0, 450.0)

Process finished with exit code 0

# Monade

```
fun main(args: Array<String>) {
 val rezultat = listOf(1, 2, 3)
 .flatMap { i ->
 listOf(i * 2, i + 3)
 }
 .joinToString()

 println("La procesarea monadica a rezultat${rezultat}")
}
si rezultatul executiei
La procesarea monadica a rezultat2, 4, 4, 5, 6, 6
```

Process finished with exit code 0

# Monade

```
sealed class TipSimplu<out T> {
 object None : TipSimplu<Nothing>() {
 override fun toString() = "None"
 }
 data class Some<out T>(val value: T) : TipSimplu<T>()
 companion object
}
fun <T, R> TipSimplu<T>.flatMap(fm: (T) -> TipSimplu<R>): TipSimplu<R> = when (this) {
 TipSimplu.None -> TipSimplu.None
 is TipSimplu.Some -> fm(value)
}
fun calculReducere(pret: TipSimplu<Double>): TipSimplu<Double> {
 return pret.flatMap { p ->
 if (p > 50.0) {
 TipSimplu.Some(5.0)
 } else {
 TipSimplu.None
 }
 }
}
fun main(args: Array<String>) {
 println(calculReducere(TipSimplu.Some(95.0)))
 println(calculReducere(TipSimplu.Some(25.0)))
 println(calculReducere(TipSimplu.None))
}
```

si rezultatul executiei  
Some(value=5.0)  
None  
None  
  
Process finished with exit code 0

# Monade

```
sealed class TipSimplu<out T> {
 object None : TipSimplu<Nothing>() {
 override fun toString() = "None"
 }
 data class Some<out T>(val value: T) : TipSimplu<T>()
 companion object
}
fun <T, R> TipSimplu<T>.flatMap(fm: (T) -> TipSimplu<R>): TipSimplu<R> = when (this) {
 TipSimplu.None -> TipSimplu.None
 is TipSimplu.Some -> fm(value)
}
fun main(args: Array<String>) {
 val poatePatru = TipSimplu.Some(4)
 val poateSapte = TipSimplu.Some(7)
 println(poatePatru.flatMap { f ->
 poateSapte.flatMap { t ->
 TipSimplu.Some(f + t)
 }
 })
}
```

Si rezultatul executiei  
Some(value=11)

Process finished with exit code 0

# Monade

```
sealed class TipSimplu<out T> {
 object None : TipSimplu<Nothing>() {
 override fun toString() = "None" }
 data class Some<out T>(val value: T) : TipSimplu<T>()
 companion object }
fun <T, R> TipSimplu<T>.flatMap(fm: (T) -> TipSimplu<R>): TipSimplu<R> = when (this) {
 TipSimplu.None -> TipSimplu.None
 is TipSimplu.Some -> fm(value) }
fun main(args: Array<String>) {
 println(TipSimplu.Some(13).flatMap(::laJumatate))
 println(TipSimplu.Some(22).flatMap(::laJumatate))
 println(TipSimplu.None.flatMap(::laJumatate))
 TipSimplu.Some(34).flatMap(::laJumatate).flatMap(::laJumatate).flatMap(::impOriDoi) }
fun laJumatate(a: Int) = when {
 a % 2 == 0 -> TipSimplu.Some(a / 2)
 else -> TipSimplu.None }
fun impOriDoi(a: Int) = when {
 a % 2 == 1 -> TipSimplu.Some(a * 2)
 else -> TipSimplu.None }
```

si ejecutia

None

Some(value=11)

None

Process finished with exit code 0

# Applicatives

```
fun <T, R> List<T>.ap(fab: List<(T) -> R>): List<R> = fab.flatMap { f ->
 this.map(f) }
```

```
fun main(args: Array<String>) {
 val numere = listOf(8, 13, 21, 34)
 val functii = listOf<(Int) -> Int>({ i -> i * 2 }, { i -> i + 3 })
 val rezultat = numere
 .ap(functii)
 .joinToString()
 println(rezultat)
}
```

## Cum se poate face o funcție să se comporte ca un aplicative?

```
object functie {
 fun <A, B> pura(b: B) = { _: A -> b } }
 fun <A, B, C> ((A) -> B).map(transform: (B) -> C): (A) -> C = { t -> transform(this(t)) }
 fun <A, B, C> ((A) -> B).flatMap(fm: (B) -> (A) -> C): (A) -> C = { t -> fm(this(t))(t) }
 fun <A, B, C> ((A) -> B).ap(fab: (A) -> (B) -> C): (A) -> C = fab.flatMap { f -> map(f) }
 fun main(args: Array<String>) {
 val sumCu7SiMulCu11: (Int) -> Int = { i: Int -> i + 7 }.ap { { j: Int -> j * 11 } }
 println(sumCu7SiMulCu11(7))
 println(sumCu7SiMulCu11(8))
 println(sumCu7SiMulCu11(9))
 val sumCu7SiMulCu11Deb: (Int) -> Pair<Int, Int> = { i:Int -> i + 7 }.ap { original ->
 { j:Int -> original to (j * 11) } }
 println(sumCu7SiMulCu11Deb(10)) 154
 println(sumCu7SiMulCu11Deb(11)) 165
 println(sumCu7SiMulCu11Deb(12)) 176
 (10, 187)
 (11, 198)
 (12, 209)
 }
}
```

Process finished with exit code 0

# Aplicative

```
sealed class TipSimplu<out T> {
 object None : TipSimplu<Nothing>() {
 override fun toString() = "None" }
 data class Some<out T>(val value: T) : TipSimplu<T>()
 companion object { }
 fun <T, R> TipSimplu<T>.flatMap(fm: (T) -> TipSimplu<R>): TipSimplu<R> = when (this) {
 TipSimplu.None -> TipSimplu.None
 is TipSimplu.Some -> fm(value) }
 fun <T, R> TipSimplu<T>.map(transform: (T) -> R): TipSimplu<R> = when (this) {
 TipSimplu.None -> TipSimplu.None
 is TipSimplu.Some -> TipSimplu.Some(transform(value)) }
 fun <T> TipSimplu.Companion.pur(t: T): TipSimplu<T> = TipSimplu.Some(t)
 fun <T, R> List<T>.ap(fab: List<(T) -> R>): List<R> = fab.flatMap { f -> this.map(f) }
 infix fun <T, R> TipSimplu<(T) -> R>.aplic(o: TipSimplu<T>): TipSimplu<R> = flatMap { f: (T) -> R -> o.map(f) }
 fun main(args: Array<String>) {
 val poatePatru = TipSimplu.Some(4)
 val poateSapte = TipSimplu.Some(7)
 println(TipSimplu.pur { f: Int -> { t: Int -> f + t } } .aplic poatePatru .aplic poateSapte) }
```

si executia  
Some(value=11)

Process finished with exit code 0

# Aplicative

```
infix inline fun <A, reified B> Array<(A) -> B>.aplicatAsupraLui(a: Array<A>) =
 Array(this.size * a.size) {
 this[it / a.size](a[it % a.size])
 }
```

```
fun main(args: Array<String>) {
 val tablou = arrayOf<(Int) -> Int>({ it + 3 }, { it * 2 }) aplicatAsupraLui arrayOf(1,
 2, 3)
 println("[${tablou.joinToString()}]")
}
```

## Currying vs aplicare parțială

|                            | <b>Currying</b>                                                       | <b>Aplicare parțială</b>                                                                        |
|----------------------------|-----------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| <b>Valoare întoarsă</b>    | când se primesc N parametri în funcție vor rezulta un lanț de funcții | când se primesc n funcții rezultă o funcție primară și una care se aplică asupra celorlalte n-1 |
| <b>Utilizare parametri</b> | după aplicare numai un parametru din lanț poate fi aplicat            | orice parametru poate fi aplicat în orice ordine                                                |
| <b>inversare proces</b>    | poate reconveritită la o funcție cu N parametri                       | nu este posibil                                                                                 |

## Exemplu de curry/uncurry

```
fun <P1, P2, P3, R> Function3<P1, P2, P3, R>.curried(): (P1) -> (P2) -> (P3) -> R =
 { p1 -> { p2 -> { p3 -> this(p1, p2, p3) } } }
fun <P1, P2, P3, R> ((P1) -> (P2) -> (P3) -> R).uncurried(): (P1, P2, P3) -> R {
 return { f1: P1, f2: P2, f3: P3 -> this(f1)(f2)(f3) }
}
var add = { x: Int, y: Int, z: Int-> x + y + z}.curried()
val add1 = { x: Int-> {y: Int -> {z: Int -> x+y+z}}}.uncurried()
fun main()
{
 val x = add(3)(4)(5)
 val y = add1(3,4,5)
 println(x)
 println(y)
}
```

## **Delegați pentru proprietăți (delegați standard)**

- funcția Delegates.notNull și lateinit
- funcția lazy
- funcția Delegates.Observable
- funcția Delegates.vetoable

## Delegates.notNull

```
import kotlin.properties.Delegates
class Loser {
 var nume: String by Delegates.notNull()
 fun initializare(nume: String) {
 this.nume = nume
 }
}
fun main() {
 val utilizator = Loser()
 utilizator.initializare("Bibistrocel")
 println(utilizator.nume)
}
```

# Delegat construit de programator

```
import kotlin.reflect.KProperty
class ExempluDelegatPropriu {
 var valoareDelegata: String by DelegatPropriu()
 override fun toString() = "Exemplu delegat propriu"
}
class DelegatPropriu() {
 operator fun getValue(thisRef: Any?, prop: KProperty<*>): String {
 return "$thisRef am primit urmatoarea delegare '${prop.name}'" }
 operator fun setValue(thisRef: Any?, prop: KProperty<*>, value: String) {
 println("$value a fost trimisa catre ${prop.name} in $thisRef") }
}
fun main() {
 val exemplu = ExempluDelegatPropriu()
 println(exemplu.valoareDelegata)
 exemplu.valoareDelegata = "Unul nou"
}
```

## lateinit

```
class SubiectTestare
{ fun zice()
 { println("Iar ma chinuie astia") }
}
public class PrimulTest {
 lateinit var subiect: SubiectTestare
 fun setup() {
 subiect = SubiectTestare() }
 fun test() {
 if (::subiect.isInitialized) subiect.zice() }
}
fun main()
{ val x=PrimulTest()
 x.setup() //comentati linia si vedeti ce se intampla
 x.test()
}
```

# **Lazy**

```
fun main() {
 val i by lazy {
 println("Evaluare la cerere")
 1
 }
 println("inainte de utilizarea lui i")
 println(i)
}
```

# Observable

```
import kotlin.properties.Delegates
class Looser {
 var nume: String by Delegates.observable("inca nu am nume") {
 proprietate, valoareVeche, valoareNoua ->
 println("$valoareVeche - $valoareNoua")
 }
}
fun main() {
 val utilizator = Looser()
 utilizator.nume = "Cel mai prost din curtea scolii"
 utilizator.nume = "Inca si mai prost"
 utilizator.nume = "bun de avansare"
}
```

# Vetoable

```
import kotlin.properties.Delegates
class Looser {
 var valoareCrescatoare: Int by Delegates.vetoable(0) {
 proprietate, valoareVeche, valoareNoua ->
 if(valoareVeche > valoareNoua) true
 else throw IllegalArgumentException("Noua valoare trebuie sa fie mai mare decat cea veche")
 }
}
fun main()
{
 var x= Looser()
 x.valoareCrescatoare =10
 x.valoareCrescatoare =12
 try{
 x.valoareCrescatoare =6
 } catch (e: java.lang.IllegalArgumentException) {
 println("ai un prietene in vizita prin program?")
 }
 x.valoareCrescatoare =15
 println(x.valoareCrescatoare)
}
```

# Delegated map

```
data class Carte (val delegate:Map<String,Any?>) {
 val nume:String by delegate
 val autori:String by delegate
 val numarPagini:Int by delegate
 val dataPublicarii:String by delegate
 val editura:String by delegate }
fun main() {
 val mapCarte1 = mapOf(
 Pair("nume","Povesti corecte politic de adormit copiii"),
 Pair("autori","James Finn Garner"),
 Pair("pageCount",200),
 Pair("dataPublicarii","01/06/2006"),
 Pair("editura","Humanitas"))
 val mapCarte2 = mapOf(
 "nume" to "Critica ratiunii pure",
 "autori" to "Immanuel Kant",
 "numarPagini" to 250,
 "dataPublicarii" to "12/05/1998",
 "editura" to "IRI")
 val cartea1 = Carte(mapCarte1)
 val cartea2 = Carte(mapCarte2)
 println("Prima Carte \n$cartea1 \nA doua Carte \n$cartea2")
}
```

# Delegați pentru clase

```
interface Persoana {
 fun afiseazaNume()
}

class ImplementarePersoana(val nume:String):Persoana {
 override fun afiseazaNume(){
 println(nume)
 }
}

class Utilizator(val persoana:Persoana):Persoana by persoana {
 override fun afiseazaNume(){
 println("Numele este:")
 persoana.afiseazaNume()
 }
}

fun main() {
 val persoana = ImplementarePersoana("Bugs Bunny")
 persoana.afiseazaNume()
 val utilizator = Utilizator(persoana)
 utilizator.afiseazaNume()
}
```

# *Paradigma Calcului Funcțional*

Cursul nr. 13  
Mihai Zaharia

# Lambda

- redefinim true, false și iff astfel:

    true = lambda x, y: x

    false = lambda x, y: y

    iff = lambda p, x, y: p(x, y)

- Cum ar arata o structură elementară de date și anume o pereche

    pair = lambda x, y: lambda f: f(x, y)

- Se introduc următoarele două funcții:

    first = lambda p: p(true)

    second = lambda p: p(false)

“P este o pereche de două obiecte x și y dacă există două funcții first și second astfel încât  $\text{first}(P)=x$  și  $\text{second}(P)=y$ ”

# **argumente și keyword arguments**

```
def func(a, *args, **kw):
 print(a)
 print(args)
 print(kw)
func('valoare A', 'valoare B', 'valoare C', argumentA = 'valoare D',
argumentB = 'valoare D')
```

**și rezultatul executiei**

valoare A

('valoare B', 'valoare C')

{'argumentA': 'valoare D', 'argumentB': 'valoare D'}

Process finished with exit code 0

# Funcții de prim nivel

- def exemplu(a, b, \*\*kw): return a \* b
- print(type(exemplu))
- print(exemplu.\_\_code\_\_.co\_varnames)
- print(exemplu.\_\_code\_\_.co\_argcount)

**și rezultatul executiei**

```
<class 'function'>
```

```
('a', 'b', 'kw')
```

```
2
```

```
Process finished with exit code 0
```

# Funcții pure

```
from functools import reduce
x=int(input("numar="))
calculNumereMersenne = lambda x: 2**x-1
print("Tipul variabilei calculNUmereMersene este:
"+str(type(calculNumereMersenne)))
print(str(calculNumereMersenne(x)))
print("mersene(%i)=%i"%(x, calculNumereMersenne(x)))
lista = [50, 71, 11, 97, 54, 62, 77]
rezultat = list(filter(lambda x: (x%2 == 0) , lista))
print(rezultat)
rezultat = list(map(lambda x: (x*2) , lista))
print(rezultat)
suma = reduce((lambda x, y: x + y), lista)
print("Suma elementelor din lista este %d" %suma)
```

si rezultatul executiei  
numar=10  
Tipul variabilei calculNUmereMersene este:  
<class 'function'>  
1023  
mersene(10)=1023  
[50, 54, 62]  
[100, 142, 22, 194, 108, 124, 154]  
Suma elementelor din lista este 422

Process finished with exit code 0

## Funcții de nivel superior

```
from functools import reduce

lista = [50, 71, 11, 97, 54, 62, 77]

rezultat=min(max(list(filter(lambda x: (x%2 == 0),
lista))),min(list(map(lambda x: (x*2) , lista))),reduce((lambda x, y: x + y),
lista))

print("rezultatul unei functii de nivel superior este %d"%rezultat)
```

si rezultatul executiei

rezultatul unei functii de nivel superior este 22

Process finished with exit code 0

## **împachetează ->procesează->despachetează**

```
preturiCarne = [(2000,30), (2001, 35), (2002, 40),(2003, 45),
 (2004, 48), (2005, 52), (2006, 57),(2007, 65),
 (2008, 70), (2009, 75), (2010, 80)]
```

```
snd= lambda x: x[1]
```

```
print(snd(max(map(lambda yc: (yc[1], yc), preturiCarne))))
```

**și rezultatul executiei**

(2010, 80)

Process finished with exit code 0

## Evaluare la cerere

True and print("and cu true")

False and print("and cu false")

1 and print("and cu unu")

0 and print("and cu zero")

True or print("or cu true")

False or print("or cu false")

**și rezultatul executiei**

and cu true

and cu unu

or cu false

Process finished with exit code 0

# Evaluare la cerere

```
from typing import Iterator
def numere(n: int) -> Iterator[int]:
 for i in range(n):
 print(f"= {i}")
 yield i
def sumaPrimelorN(n: int):
 suma: int = 0
 for i in numere(n):
 print(f"= {i}")
 if i == n: break
 suma += i
 return suma
def sumaPrimelorN1(n: int):
 suma: int = 0
 for i in range(n):
 suma += i
 return suma
```

```
def sumaPrimelorN2(n: int):
 suma: int = 0
 i:int =0
 while True:
 print(f"= {i}")
 suma += i
 i+=1
 if(i>=n):break
 return suma
x = 6
print("Sumare cu generator %d" %sumaPrimelorN(x))
print("Sumare cu range %d" %sumaPrimelorN1(x))
print("Sumare clasica %d" %sumaPrimelorN2(x))
```

|                                   |  |
|-----------------------------------|--|
| <b>si rezultatul executiei</b>    |  |
| = 0                               |  |
| = 0                               |  |
| = 1                               |  |
| = 1                               |  |
| = 2                               |  |
| = 2                               |  |
| = 3                               |  |
| = 3                               |  |
| = 4                               |  |
| = 4                               |  |
| = 5                               |  |
| = 5                               |  |
| Sumare cu generator 15            |  |
| Sumare cu range 15                |  |
| = 0                               |  |
| = 1                               |  |
| = 2                               |  |
| = 3                               |  |
| = 4                               |  |
| = 5                               |  |
| Sumare clasica 15                 |  |
| Process finished with exit code 0 |  |

# Generatoare recursive

```
from typing import Iterator
def pfactorsr(x: int) -> Iterator[int]:
 def factor_n(x: int, n: int) ->
 Iterator[int]:
 if n*n > x:
 yield x
 return
 if x % n == 0:
 yield n
 if x//n > 1:
 yield from factor_n(x // n, n)
 else:
 yield from factor_n(x, n+2)
 if x//2 > 1:
 yield from pfactorsr(x//2)
 return
print(list(pfactorsr(1455560)))
```

si rezultatul executiei  
[2, 2, 2, 5, 36389]  
[  
Process finished with exit code 0

## Un alt exemplu combinare liste de caractere

```
def combinare(ADTiterabil, index=0, lungime=1):
 it = iter(ADTiterabil)
 for contor in range(index):
 yield next(it)
 combinata = next(it)
 for count in range(lungime-1):
 combinata += next(it)
 yield combinata
 for element in it:
 yield element

l1 = ['11', '22', '3', '4']
l2 = []
l2 = list(combinare(l1,0,len(l1)))
print(l2)
```

si rezultatul executiei  
['112234']  
Process finished with exit code 0

## Afișare fără conversie la listă

```
def printIterator(it):
 s=""
 for x in it:
 s=s+' '+str(x)
 print("%s\n%s")
 print(l2)
l1 = [1, 2, 3, 4]
t1 = (5, 6, 7, 8, 9, 10)
m = map(lambda x, y: x * y, l1, t1)
printIterator(m)
print(list(m))#pentru ca iteratorul a fost consumat deja
```

si rezultatul executiei  
5 12 21 32  
[]  
Process finished with exit code 0

# Funcții din biblioteca `itertools`

- **accumulate(iterable[, func])** furnizează o serie de serii ale elementelor dintr-o structură iterabilă
- **chain(\*iterables)** parcurge mai multe structuri iterabile una după alta fără a crea o listă intermediară a tuturor elementelor
- **combinations(iterable, r)** generează toate combinațiile de lungime **r** pornind de la o structură iterabilă
- **compress(data, selectors)** va aplica o mască logică (booleană) furnizată de selectori asupra elementelor și ne furnizează numai acele valori care corespund criteriilor de selecție din selectori.
- **count(start, step)** generează o secvență infinită de valori începând cu cea de start și incrementând-o cu step la fiecare apel
- **cycle(iterable)** parcurge în mod repetat (într-o buclă) valorile dintr-o structură iterabilă
- **repeat(elem[, n])** repetă un element de **n** ori
- **dropwhile(predicate, iterable)** extrage o submulțime de elemente pornind dela primul și continuând până ce predicatul devine Fals
- **groupby(iterable, keyfunc)** crează un iterator care grupează elemente în funcție de rezultatul furnizat de funcția `keyfunc()`.
- **permutations(iterable[, r])** furnizează permutări succesive de dimensiune **r** ale elementelor dintr-o structură iterabilă.

## **itertools - iteratori infiniti - count**

```
import itertools as it
contorReal = it.count(start=0.5, step=0.75)
print(list(next(contorReal) for _ in range(5)))
contorRealPrecizie= (0.5+x*.75 for x in it.count())
print(list(next(contorRealPrecizie) for _ in range(5)))
contorDescrescator=it.count(start=-1, step=-0.5)
print(list(next(contorDescrescator) for _ in range(5)))
#simulare functie enumerate
print(list(zip(it.count(), ['a', 'b', 'c'])))
numaraDinTreInTrei=it.count(step=3)
print(list(next(numaraDinTreInTrei) for _ in range(5)))
```

si rezultatul executiei

[0.5, 1.25, 2.0, 2.75, 3.5]

[0.5, 1.25, 2.0, 2.75, 3.5]

[-1, -1.5, -2.0, -2.5, -3.0]

[(0, 'a'), (1, 'b'), (2, 'c')]

[0, 3, 6, 9, 12]

Process finished with exit code 0

## calculul erorii prin acumulare

```
import itertools as it
#functie utila
def until(terminate, iterator):
 i = next(iterator)
 if terminate(*i):
 return i
 return until(terminate, iterator)
source = zip(it.count(0, .1), (.1*c for c in it.count()))
neq = lambda x, y: abs(x-y) > 1.0E-12
print(until(neq, source))
#daca cati pasi apare deja diferenta
print(until(lambda x, y: x != y, source))
```

si rezultatul executiei  
(92.799999999999, 92.8000000000001)  
(92.899999999999, 92.9)

Process finished with exit code 0

## **itertools - iteratori infiniți - cycle**

```
sinus=it.cycle([1, -1])
print(list(next(sinus) for _ in range(6)))
ceva=it.cycle([3,1,0,-1,-3])
print(list(next(ceva) for _ in range(9)))
sir=it.cycle(['a', 'b', 'c'])
print(list(next(sir) for _ in range(6)))
```

**și rezultatul executiei**

```
[1, -1, 1, -1, 1, -1]
[3, 1, 0, -1, -3, 3, 1, 0, -1]
['a', 'b', 'c', 'a', 'b', 'c']
```

Process finished with exit code 0

# itertools - accumulate

```
import operator as op
import itertools as it
rezultat = list(it.accumulate([1, 2, 3, 4, 5], op.add))
print(rezultat)
rezultat = list(it.accumulate([1, 2, 3, 4, 5]))
print(rezultat)
rezultat = list(it.accumulate([1, 2, 3, 4, 5], lambda x, y: (x + y) / 2))
print(rezultat)
#ordinea argumentelor
ordine1Arg = list(it.accumulate([1, 2, 3, 4, 5], lambda x, y: x - y))
print(ordine1Arg)
ordine2Arg = list(it.accumulate([1, 2, 3, 4, 5], lambda x, y: y - x))
print(ordine2Arg)
#pentru $s_i = P * s_{i-1} + Q$, $\forall i \in \mathbb{N}$ vom avea:
def generareSecventa(p, q, valoareInitiala):
 return it.accumulate(it.repeat(valoareInitiala), lambda s, _: p * s + q)
pare = generareSecventa(1, 2, 0)
primeleOpt = list(next(pare) for _ in range(8))
print(primeleOpt)
```

si rezultatul executiei

[1, 3, 6, 10, 15]

[1, 3, 6, 10, 15]

[1, 1.5, 2.25, 3.125, 4.0625]

[1, -1, -4, -8, -13]

[1, 1, 2, 2, 3]

recursivitate

[0, 2, 4, 6, 8, 10, 12, 14]

Process finished with exit code 0

# itertools - iteratori infiniti - repeat

```
import itertools as it
print(list(tuple(it.repeat(i, times=i)) for i in range(5)))
print(list(sum(it.repeat(i, times=i)) for i in range(10)))
def generareSecventa(p, q, valoareInitiala):
 return it.accumulate(it.repeat(valoareInitiala), lambda s, _: p * s + q)
print("serii")
numerePare = generareSecventa(1,2,0)
secvNumerePare = list(next(numerePare) for _ in range(8))
print(secvNumerePare)
numerelImpare = generareSecventa(1,2,1)
secvNumerelImpare = list(next(numerelImpare) for _ in range(8))
print(secvNumerelImpare)
numereDinTreilnTrei = generareSecventa(1,3,0)
secvNumereDinTreilnTrei = list(next(numereDinTreilnTrei) for _ in range(8))
print(secvNumereDinTreilnTrei)
numaiUnu = generareSecventa(1,0,1)
secvNumaiUnu = list(next(numaiUnu) for _ in range(8))
print(secvNumaiUnu)
numereAlternative = generareSecventa(-1,1,1)
secvNumereAlternative = list(next(numereAlternative) for _ in range(8))
print(secvNumereAlternative)
```

**și rezultatul executiei**

[(), (1,), (2, 2), (3, 3, 3), (4, 4, 4, 4)]  
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]  
serii  
[0, 2, 4, 6, 8, 10, 12, 14]  
[1, 3, 5, 7, 9, 11, 13, 15]  
[0, 3, 6, 9, 12, 15, 18, 21]  
[1, 1, 1, 1, 1, 1, 1, 1]  
[1, 0, 1, 0, 1, 0, 1, 0]

Process finished with exit code 0

# itertools - creare funcții de ordin doi

- pentru reamintire:  $s(n) = p * s(n-1) + q * s(n-2) + r$

```
import itertools as it
```

```
def secentaOrdinDoi(p, q, r, initial_values):
```

```
 intermediate = it.accumulate(
```

```
 it.repeat(initial_values),
```

```
 lambda s, _: (s[1], p*s[1] + q*s[0] + r))
```

```
 return map(lambda x: x[0], intermediate)
```

```
numereFibonacci = secentaOrdinDoi(1, 1, 0, (0, 1))
```

```
secvNumereFibonacci = list(next(numereFibonacci) for _ in range(8))
```

```
print(secvNumereFibonacci)
```

```
numerePell = secentaOrdinDoi(2, 1, 0, (0, 1))
```

```
secvNumerePell = list(next(numerePell) for _ in range(6))
```

```
print(secvNumerePell)
```

```
numereLucas = secentaOrdinDoi(1, 1, 0, (2, 1))
```

```
secvNumereLucas = list(next(numereLucas) for _ in range(6))
```

```
print(secvNumereLucas)
```

```
numereFibonacciAlternative = secentaOrdinDoi(-1, 1, 0, (-1, 1))
```

```
secvNumereFibonacciAlternative = list(next(numereFibonacciAlternative) for _ in range(6))
```

```
print(secvNumereFibonacciAlternative)
```

**si rezultatul executiei**

[0, 1, 1, 2, 3, 5, 8, 13]

[0, 1, 2, 5, 12, 29]

[2, 1, 3, 4, 7, 11]

[-1, 1, -2, 3, -5, 8]

Process finished with exit code 0

## alte functii din itertools

```
import itertools as it
numere = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
def grupareOptimizata(lista, numarTuple):
 iteratori = [iter(lista)] * numarTuple
 return zip(*iteratori)
grupate1 = list(grupareOptimizata(numere, 2))
print(grupate1)
grupate2 = list(grupareOptimizata(numere, 4))
print(grupate2)
def grupareOptimizataCuPadding(lista, numarTuple, valoareCompletare=None):
 iteratori = [iter(lista)] * numarTuple
 return it.zip_longest(*iteratori, fillvalue=valoareCompletare)
grupate3 = list(grupareOptimizataCuPadding(numere, 4))
print(grupate3)
```

## alte functii din itertools

```
combinare = list(it.combinations(numere, 3))
print(combinare)
seturiCuSuma12 = []
for i in range(1, len(numere) + 1):
 for combinatie in it.combinations(numere, i):
 if sum(combinatie) == 12:
 seturiCuSuma12.append(combinatie)
print(seturiCuSuma12)
seturiUniceCuSuma12 = set(seturiCuSuma12)
print(seturiUniceCuSuma12)
seturiCuSuma12Silnlocuire = []
for i in range(1, len(numere) + 1):
 for combinatie in it.combinations_with_replacement(numere, i):
 if sum(combinatie) == 12:
 seturiCuSuma12Silnlocuire.append(combinatie)
print(seturiCuSuma12Silnlocuire)
```

## Clonare iteratori -tee()

```
import itertools as it
numere = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
def calcul(lista):
 it0, it1 = it.tee(lista,2)
 s0= sum(1 for x in it0)
 s1= sum(x for x in it1)
 return s1/s0
print(calcul(numere))
```

# list comprehension

```
• lista = [expresie for element in lista_intrare if conditie]
x=[]#cod clasic
for i in (1,2,3):#la intrare o tupla
 x.append(i)
print(x)
#utilizand list comprehension
x = [i for i in [1, 2, 3]]
print(x)#incepe exemplu 3
oLista = (1, "4", 9, "a", 0, 4)
for i in oLista:
 print(type(i))
listalntregi = [i**2 for i in oLista if str(i.__class__)==<class \'int\'>]
print("Lista filtrata%s" %''.join(map(str, listalntregi)))
```

si rezultat executie

[1, 2, 3]  
[1, 2, 3]  
<class 'int'>  
<class 'str'>  
<class 'int'>  
<class 'str'>  
<class 'int'>  
<class 'int'>  
Lista filtrata1 81 0 16

Process finished with exit code 0

## **procesare avansată liste**

```
oLista = (1, "4", 9, "a", 0, 4)
print(type(oLista))
rezultat =map(lambda el:el**2, filter(lambda el: str(type(el))=='<class
\'int\'>',oLista))
print("Lista filtrata %s" %str(list((rezultat))[0:4]))
```

```
si rezultatul
<class 'tuple'>
<class 'tuple'>
Lista filtrata [1, 81, 0, 16]
```

Process finished with exit code 0

## Alt exemplu

```
#aplicare partiala calcul functional
numere = [1, 2, 3, 4, 5]
def doiX(x):
 return x + x
def laPatrat(x):
 return x * x
listaProcesata = []
for i in numere:
 listaProcesata = listaProcesata+list(map(lambda x: x(i), (doiX, laPatrat)))
print(listaProcesata)
#daca nu am nevoie de reutilizarea celor doua functii
listaProcesata1 = []
for i in numere:
 listaProcesata1 = listaProcesata1+list(map(lambda x: x(i), (lambda x:x+x, lambda x:x*x)))
print(listaProcesata1)
```

## set & dictionary comprehensions

```
nume = ['Ion', 'BULA', 'sile', 'cucu', 'GODAC', 'J&k', 'CORO']
```

```
submultime = {element[0].upper() + element[1:].lower() for element in
nume if len(element) > 2 and len(element) < 4}
```

```
print(list(submultime))
```

```
dictionar = {'a': 10, 'b': 34, 'A': 7, 'Z': 3}
```

```
frecv_dictionar = {k.lower(): dictionar.get(k.lower(), 0) +
dictionar.get(k.upper(), 0) for k in dictionar.keys()}
```

```
print(frecv_dictionar)
```

si rezultatul executiei  
['Ion', 'J&k']  
{'a': 17, 'b': 34, 'z': 3}

Process finished with exit code 0

# comprehensie Dictionar

```
#intersectie
x = {'a': 1, 'b': 2}
y = {'b': 3, 'c': 4}
z = {**x, **y}
print(z)

#comprehensie
valoriGradeF = {'t1': -32,'t2': -24,'t3': -13,'t4': 0}
valoriGradeC = {k:round((float(5)/9)*(v-32)) for (k,v) in valoriGradeF.items()}
print(valoriGradeC)

dict1 = {'e': 1, 'f': 2, 'k': 3, 'p': 4, 'l': 5}
aplicare conditie (valoare element > 2)
filtrareDictCuCond = {k:v for (k,v) in dict1.items() if v>2}
print(filtrareDictCuCond)

aplicare conditii multiple
filtrareDictCuCond = {k:v for (k,v) in dict1.items() if v>2 if v%2 == 0}
print(filtrareDictCuCond)

#aplicare conditii una intr-alta
nestedDict = {'primulEl':{'a':1}, 'alDoileaEl':{'b':2}}
conversieFloatDict = {exterior_k: {float(interior_v) for (interior_k, interior_v) in exterior_v.items()}
 for (exterior_k, exterior_v) in nestedDict.items()}
print(conversieFloatDict)
```

si rezultatul executiei

{'a': 1, 'b': 3, 'c': 4}

{'t1': -36, 't2': -31, 't3': -25, 't4': -18}

{'k': 3, 'p': 4, 'l': 5}

{'p': 4}

{'primulEl': {1.0}, 'alDoileaEl': {2.0}}

Process finished with exit code 0

# Nested comprehensions

```
numarLinii=5
```

```
numarColoane=5
```

```
matrice = [[1 if elementMatrice == linie else 0 for elementMatrice in
range(0, numarColoane)] for linie in range(0, numarLinii)]
```

```
for i in range(0,numarLinii):
```

```
 print(matrice[i])
```

si rezultatul executiei

```
[1, 0, 0, 0, 0]
```

```
[0, 1, 0, 0, 0]
```

```
[0, 0, 1, 0, 0]
```

```
[0, 0, 0, 1, 0]
```

```
[0, 0, 0, 0, 1]
```

```
for i in range(0,numarLinii):
```

```
 for j in range(0, numarColoane):pass
```

```
#print(matrice[i][j])
```

Process finished with exit code 0

# Operatorii any si all

```
listaNumere=(2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103)
listaNumereNormale = [i for i in range(25)]
listaNumerePare = [2*i for i in range(15)]
def isprime(n):
 n = abs(int(n))
 if n < 2:
 return False
 if n == 2:
 return True
 if not n & 1:
 return False
 for x in range(3, int(n**0.5)+1, 2):
 if n % x == 0:
 return False
 return True
if all(isprime(x) for x in listaNumere):print("Lista de numere prime")
if not all(isprime(x) for x in listaNumereNormale):print("Lista de numere")
if any(not isprime(x) for x in listaNumerePare):print("Lista de numere")
```

si rezultatul executiei  
Lista de numere prime  
Lista de numere  
Lista de numere  
Process finished with exit code 0

## Zip si unzip

```
listaNumere=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
 89, 97, 101, 103]
```

```
listaNumereNormale = [i for i in range(28)]
```

```
fermoar=zip(listaNumereNormale,listaNumere)
```

```
print(list(fermoar))
```

```
print(list(fermoar))
```

```
stanga=(x[0] for x in fermoar)
```

```
dreapta=(x[1] for x in fermoar)
```

```
print(list(stanga))
```

```
print(list(dreapta))
```

si rezultatul executiei

```
[(0, 2), (1, 3), (2, 5), (3, 7), (4, 11),
(5, 13), (6, 17), (7, 19), (8, 23), (9,
29), (10, 31), (11, 37), (12, 41), (13,
43), (14, 47), (15, 53), (16, 59), (17,
61), (18, 67), (19, 71), (20, 73), (21,
79), (22, 83), (23, 89), (24, 97), (25,
101), (26, 103)]
```

[]

[]

[]

Process finished with exit code 0

## **zip si unzip**

```
listaNumere=[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37,
 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
 89, 97, 101, 103]
listaNumereNormale = [i for i in range(28)]
fermoar=zip(listaNumereNormale,listaNumere)
listaDinObjZip=list(fermoar)
listaFinala = listaDinObjZip[:]
listaFinala1 = list(listaDinObjZip)
print(list(listaFinala))
print(list(listaFinala1))
stanga=(x[0] for x in listaFinala)
dreapta=(x[1] for x in listaFinala1)
print(list(stanga))
print(list(dreapta))
print(list(stanga))
print(list(dreapta))
```

## Data Flatten - ex listă cu tuple

```
listaValori=[('5', '3', '5', '7', '11'), ('13', '17', '19', '23', '29'), ('31', '137', '41', '143',
'47'), ('53', '59', '61', '67', '71'), ('173', '9', '83', '89', '97'), ('101', '103', '107', '109',
'113'), ('12', '131', '17', '139', '19'), ('151', '157', '163', '167', '173'), ('179', '18',
'191', '199', '197'), ('194', '21', '123', '225', '229')]
def flatten(aList):
 t = []
 for i in aList:
 if not isinstance(i, tuple):
 t.append(i)
 else:
 t.extend(flatten(i))
 return t
listaCurata=flatten(listaValori)
print(list(listaCurata))
```

**si rezultatul executiei**  
['5', '3', '5', '7', '11', '13', '17', '19', '23', '29', '31', '137', '41',  
'143', '47', '53', '59', '61', '67', '71', '173', '9', '83', '89', '97',  
'101', '103', '107', '109', '113', '12', '131', '17', '139', '19', '151',  
'157', '163', '167', '173', '179', '18', '191', '199', '197', '194', '21',  
'123', '225', '229']

Process finished with exit code 0

# Data Flatten - ex cap tabel extras din json

```
inregistrare = {'Nume':'Bula', 'Locatia':{'Oras':'Pocreaca','Tara':'ROM'}, 'hobi':['Manea', 'Bautura', 'Femei']}
def flatten_tabel(y):
 iesire = {}
 def flatten(x, nume=''):
 if type(x) is dict:
 for a in x:
 flatten(x[a], nume + a + '_')
 elif type(x) is list:
 i = 0
 for a in x:
 flatten(a, nume + str(i) + '_')
 i += 1
 else:
 iesire[nume[:-1]] = x
 flatten(y)
 return iesire
listaCurata=flatten_tabel(inregistrare)
print(listaCurata)
```

**si rezultatul executiei**

```
{'Nume': 'Bula', 'Locatia_Oras': 'Pocreaca', 'Locatia_Tara': 'ROM', 'hobi_0': 'Manea', 'hobi_1': 'Bautura', 'hobi_2': 'Femei'}
```

Process finished with exit code 0

## enumerate, slice & reverse

```
listaSimpla=['5', '3', '5', '7', '11', '13', '17', '19', '23', '29', '31', '137', '41', '143',
'47', '53', '59', '61', '67', '71', '173', '9', '83', '89', '97', '101', '103']
student = ('manaca','doarme','femei')
#slice
n=len(listaSimpla)
listaSliced=zip((listaSimpla[2*i::n] for i in
range(round(n/2))), (listaSimpla[2*i+1::n] for i in range(round(n/2))))
ptListat=list(listaSliced)
print(ptListat)
listaReversed = reversed(listaSimpla)
print(list(listaReversed))
print(list(enumerate(student)))
```

[([['5'], ['3']], ([['5'], ['7']]), ([['11'], ['13']]), ([['17'], ['19']]), ([['23'],
['29']]), ([['31'], ['137']]), ([['41'], ['143']]), ([['47'], ['53']]), ([['59'],
['61']]), ([['67'], ['71']]), ([['173'], ['9']]), ([['83'], ['89']]), ([['97'],
['101']]), ([['103'], []])]  
['103', '101', '97', '89', '83', '9', '173', '71', '67', '61', '59', '53',
'47', '143', '41', '137', '31', '29', '23', '19', '17', '13', '11', '7', '5',
'3', '5']  
[(0, 'manaca'), (1, 'doarme'), (2, 'femei')]

Process finished with exit code 0

## **map() echivalent pentru for**

for e in it:

    func(e)

map(func, it)

- **Versiunea 2**

compunereFunctii = lambda f, \*args: f(\*args)

map(compunereFunctii, [f1, f2, f3])

- **Versiunea 3**

compunereFunctii = lambda fns, \*args: [list(map(fn, \*args)) for fn in fns]

## și exemplul de aplicare

```
ista1=list(range(11, 17))
f1=lambda x:x*2
f2=lambda x:x+2
f3=lambda x:round(x/2)
tabelFunctii=[f1, f2, f3]
compunereFunctii = lambda tabelFunctii, *args: [list(map(functie, *args)) for
functie in tabelFunctii]
lista=compunereFunctii([f1,f2,f3],lista1)
print(lista)

atenție la următoarele diferențe:
map(lambda for v: in map(lambda for w: v + w, in y), in x) #este "pseudo"
lambda
iar codul echivalent
map(lambda v : map(lambda w : v + w, y), x) #este lambda real
```

**și rezultatul executiei**

[[22, 24, 26, 28, 30, 32], [13, 14, 15, 16, 17, 18], [6, 6, 6, 7, 8, 8]]

Process finished with exit code 0

## **echivalare if elif în calcul funcțional sau scurtcircuit**

- # structură standard de control a fluxului de date

```
if <cond1>: func1()
elif <cond2>: func2()
else: func3()
```

- # Își echivalentul ei funcțional numit câteodată și expresie scurtcircuit ( $<\text{cond1}> \text{ and } \text{func1}()$ ) or ( $<\text{cond2}> \text{ and } \text{func2}()$ ) or ( $\text{func3}()$ )

- În sfârșit vine și lambda cu scurtcircuit pentru aceeași structură:  
 $\text{lambdascurtcircut} = \lambda x: (\text{cond1} \text{ and } \text{func1}(\text{parlist})) \text{ or } (\text{cond2} \text{ and } \text{func2}(\text{parlist})) \text{ or } (\text{func3}(\text{parlist}))$

# Expresii scurtcircuit

```
lista2 = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83,
89, 97, 101, 103]
```

```
val1 = lambda x: x in range(0, 3)
```

**si rezultatul executiei**

```
val2 = lambda x: x in range(3, 100)
```

```
[4, 5, 7, 9, 13, 15, 19, 21, 25, 31, 33, 39, 43, 45, 49, 55, 61,
63, 69, 73, 75, 81, 85, 91, 99, 50, 52]
```

```
val3 = lambda x: x > 100
```

```
f1 = lambda x: x * 2
```

Process finished with exit code 0

```
f2 = lambda x: x + 2
```

```
f3 = lambda x: round(x / 2)
```

```
eval = (lambda x, y: y if x == True else False)
```

```
scurtcircuit = lambda x: eval(val1(x), f1(x)) or eval(val2(x), f2(x)) or eval(val3(x), f3(x))
```

```
map1 = map(scurtcircuit, lista2)
```

```
print(list(map1))
```

# Closure

```
def fabricaDeFunctii(x):
 def contine(lst):
 return x in lst
 return contine
```

```
si exemplu de executie
<function fabricaDeFunctii.<locals>.contine at 0x7fd15d496d90>
False
True
True
Process finished with exit code 0
```

```
amProcent100 = fabricaDeFunctii(100)
print(amProcent100)
print(amProcent100([1,2,30,40,50]))
print(amProcent100([1,2,30,40,50,100]))
print(amProcent100(range(1, 200)))
```

## contor ca o closure - ver 1

```
def fabricaDeContoare():
 contorvar = 0
 def contor():
 nonlocal contorvar
 contorvar += 1
 return contorvar
 return contor
numarator1 =fabricaDeContoare()
print(numarator1())
print(numarator1())
print(numarator1())
numarator2 =fabricaDeContoare()
print(numarator2())
print(numarator2())
print(numarator2())
```

## contor ca o closure - ver 2

```
def fabricaDeContoare(): si rezultatul executiei
 contorvar = 0
 def contor():
 nonlocal contorvar
 contorvar += 1
 return contorvar
 def valoareCurentaContor():
 nonlocal contorvar
 return contorvar
 return contor, valoareCurentaContor
contor,valoareContor = fabricaDeContoare()
print(contor())
print(contor())
print(contor())
print(valoareContor())
```

1  
2  
3  
3  
Process finished with exit code 0

# Analiză de performanțe

```
import time
nrIteratii = 10000000
def f(k):
 return 2*k
def benchmark(functie, numeFunctie):
 start = time.time()
 functie()
 stop = time.time()
 print("calculul a durat {0} seconde pentru {1}".format((stop - start), numeFunctie))
def listaA():
 listaA = []
 for i in range(nrIteratii):
 listaA.append(f(i))
def listaB():
 listaB = [f(i) for i in range(nrIteratii)]
def listaC():
 listaC = map(f, range(nrIteratii))
benchmark(listaA, "structurata")
benchmark(listaB, "caclul functional")
benchmark(listaC, "operator dedicat")
```

## și rezultatul executiei

```
calculul a durat 1.2460236549377441 seconde
pentru structurata
calculul a durat 0.9806721210479736 seconde
pentru caclul functional
calculul a durat 6.198883056640625e-06 seconde
pentru operator dedicat
```

```
Process finished with exit code 0
```

## pentru de zece ori mai multe operatii

```
calculul a durat 11.70402979850769 seconde
pentru structurata
calculul a durat 9.303850650787354 seconde
pentru caclul functional
calculul a durat 5.7220458984375e-06 seconde
pentru operator dedicat
```

```
Process finished with exit code 0
```

# Excepții personalizate

```
#versiune simplista
def cereVarsta(varsta):
 try:
 assert int(varsta) > 18
 except ValueError:
 return 'Interzis minorilor'
 else:
 return 'batran dar nebunatic'

print(cereVarsta('ciaia bai'))
print(cereVarsta(60))
print(cereVarsta(6))
```

**și rezultatul executiei**

Interzis minorilor

Traceback (most recent call last):

batran dar nebunatic

File "/home/bugs/PycharmProjects/exceptii  
personalizate/excepții personalizate.py", line 12, in  
<module>

print(cereVarsta(6))

File "/home/bugs/PycharmProjects/exceptii  
personalizate/excepții personalizate.py", line 4, in  
cereVarsta

assert int(varsta) > 18

AssertionError

Process finished with exit code 1

# Excepții personalizate

```
def cereVarsta(varsta):
 try:
 if(int(varsta) > 18):
 raise ZeroDivisionError
 except ValueError:
 return 'Interzis minorilor'
 else:
 return 'batran dar nebunatic'
```

```
print(cereVarsta('ciaia bai'))
print(cereVarsta(60))
print(cereVarsta(6))
```

si rezultatul executiei

Interzis minorilor

Traceback (most recent call last):

File "/home/bugs/PycharmProjects/excepții personalizate/excepții personalizate.py", line 12, in <module>

    print(cereVarsta(60))

File "/home/bugs/PycharmProjects/excepții personalizate/excepții personalizate.py", line 5, in cereVarsta

    raise ZeroDivisionError

ZeroDivisionError

# Excepții personalizate

```
class EsteMinor(Exception):
 pass

def areVarsta(varsta):
 if int(varsta) < 18:
 raise EsteMinor

try:
 areVarsta(23)
 areVarsta(17)
except EsteMinor:
 print("Va rugam sa reveniti cand imbatraniti")
```

si rezultatul executiei  
Va rugam sa reveniti cand imbatraniti  
Process finished with exit code 0

# Decorator - stil macro

```
decorator fara argumente
def ornament(oFunctie):
 def impachetezFunctia():
 print("blocul predecesor al apelului functiei originale")
 oFunctie()
 print("blocul successor al apelului functiei originale")
 return impachetezFunctia

aplicam decoratorul asupra unei functii si o definim
@ornament
def functie1():
 print("se executa functia originala din decorator")

#apelul decoratorului
functie1()
```

si rezultatul executiei  
blocul predecesor al apelului functiei originale  
se executa functia originala din decorator  
blocul successor al apelului functiei originale

Process finished with exit code 0

# Decorator - stil PeOO

```
class ornament(object):
```

```
 def __init__(self, f):
```

```
 print("sunt in constructorul decoratorului")
```

```
 f()
```

```
 def __call__(self):
```

```
 print("sunt in decorator")
```

si **rezultatul decorarii**

sunt in constructorul decoratorului

sunt in interiorul functiei decorate

sunt in decorator

```
@ornament
```

```
def functie1():
```

```
 print("sunt in interiorul functiei decorate")
```

```
functie1()
```

# Decorator POO versiunea mai serioasă

```
#versiune mai serioasa
class ornament(object):
 def __init__(self, f):
 self.f = f
 def __call__(self):
 print("Intru in corpul decorator", self.f.__name__)
 self.f()
 print("ies din corpul decoratorului", self.f.__name__)
@ornament
def functia1():
 print("sunt in functia 1")
@ornament
def functia2():
 print("sunt in functia 2")

functia1()
functia2()
```

**si rezultatul executiei**  
Intru in corpul decorator functia1  
sunt in functia 1  
ies din corpul decoratorului functia1  
Intru in corpul decorator functia2  
sunt in functia 2  
ies din corpul decoratorului functia2

Process finished with exit code 0

# Decorator cu parametri

```
class ornamentCuParametri(object):
 def __init__(self, par1, par2, par3):
 #print("nu mai pot apela functia decorata in constructorul decoratorului")
 self.par1 = par1
 self.par2 = par2
 self.par3 = par3
 def __call__(self, f):
 #print("in interiorul decoratorului")
 def functieImpachetata(*args):
 #print("in interiorul functiei de impachetare")
 print("Argumentele decoratorului sunt", self.par1, self.par2, self.par3)
 f(*args)
 #print("dupa apelul functiei decorate")
 return functieImpachetata
@ornamentCuParametri("vreau", "sa merg", 42)
def sayHello(a1, a2, a3, a4):
 print("%s %s %s %s" %(a1, a2, a3, a4))

sayHello("ma","duc","la","bere")
```

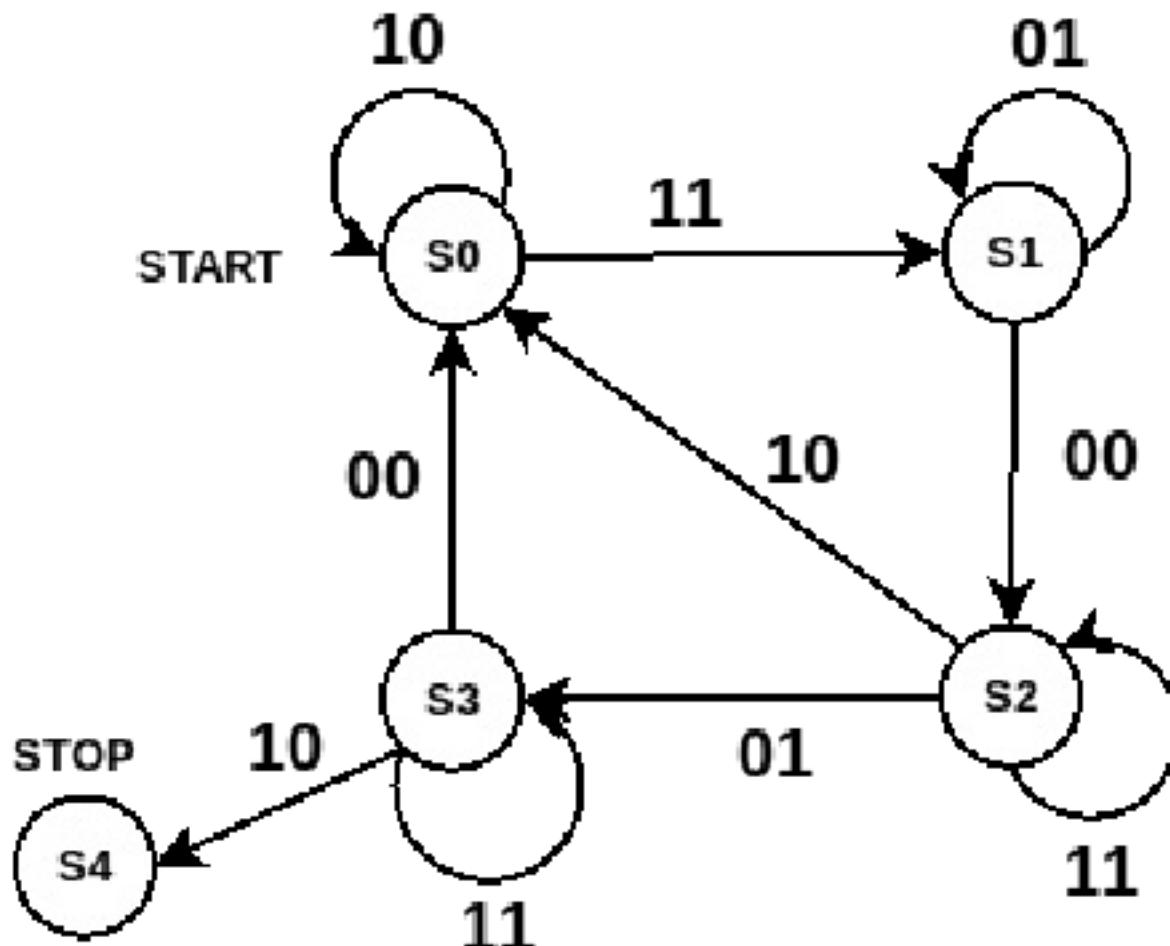
# Decorator utilizat ca adapter

```
class ornamentCuParametri(object):
 def __init__(self, par1, par2, par3):
 #print("nu mai pot apela functia decorata in constructorul decoratorului")
 self.par1 = par1
 self.par2 = par2
 self.par3 = par3
 def __call__(self, f):
 #print("in interiorul decoratorului")
 def functieImpachetata(*args):
 print("Argumentele decoratorului sunt", self.par1, self.par2, self.par3)
 #if(f.__code__.co_varnames[0]=='ma'):f(*args)
 if(not args[0]):
 f(self.par1,args[1],args[2],args[3])
 else:
 f(*args)
 return functieImpachetata
@ornamentCuParametri("vreau", "sa merg", 42)
def sayHello(a1, a2, a3, a4):
 print("%s %s %s %s" %(a1, a2, a3, a4))
sayHello("", "duc", "la", "bere")
sayHello("ma", "duc", "la", "bere")
```

Argumentele decoratorului sunt vreau sa merg 42  
vreau duc la bere

Argumentele decoratorului sunt vreau sa merg 42  
ma duc la bere

## Să implementăm un automat



# Implementare state machine

```
class StateMachine():
 def __init__(self):
 self.handlers = {}
 self.startState = None
 self.endStates = []
 self.cargo1 = -1
 self.cargo2 = -1
 self.handler=0
 def add_state(self, name, handler, start_state=0,
 end_state=0):
 name = name.upper()
 self.handlers[name] = handler
 if end_state:
 self.endStates.append(name)
 if start_state:
 self.startState = name.upper()
 def setCargo(self, cargo1, cargo2):
 self.handler = self.handlers[self.startState]
 self.cargo1 = cargo1
 self.cargo2 = cargo2
```

# Implementare state machine

```
def run(self):
 (newState, self.cargo1, self.cargo2) =
 self.handler(self.cargo1, self.cargo2)
 if newState.upper() in self.endStates:
 print("Am ajuns in stare terminala")
 else:
 self.handler = self.handlers[newState.upper()]
def stare0(x1, x2):
 print("S0")
 while True:
 if (x1 == 1) and (x2 == 1):
 newState = "S1"
 break
 elif (x1 == 1) and (x2 == 0):
 newState = "S0"
 break
 else:
 newState = "S4"
 print("inrare eronata - STOP")
 break
 print(">>") # pe post de actiune in starea curenta
 return (newState, x1, x2)
```

```
def stare1(x1, x2):
 print("S1")
 while True:
 if (x1 == 0) and (x2 == 1):
 newState = "S1"
 break
 elif (x1 == 0) and (x2 == 0):
 newState = "S2"
 break
 else:
 newState = "S4"
 print("inrare eronata - STOP")
 break
 print(">>") # pe post de actiune in starea curenta
 return (newState, x1, x2)

def stare2(x1, x2):
 print("S2")
 while True:
 if (x1 == 1) and (x2 == 1):
 newState = "S2"
 break
 elif (x1 == 1) and (x2 == 0):
 newState = "S0"
 break
 elif (x1 == 0) and (x2 == 1):
 newState = "S3"
 break
 else:
 newState = "S4"
 print("inrare eronata - STOP")
 break
 print(">>") # pe post de actiune in starea curenta
 return (newState, x1, x2)
```

# Implementare state machine

```
def stare3(x1, x2):
 print("S3")
 while True:
 if (x1 == 1) and (x2 == 1):
 newState = "S3"
 break
 elif (x1 == 0) and (x2 == 0):
 newState = "S0"
 break
 elif (x1 == 1) and (x2 == 0):
 newState = "S4"
 break
 else:
 newState = "S4"
 print("in rare eronata - STOP")
 break
 print(">>") # pe post de actiune in starea curenta
 return (newState, x1, x2)
```

```
def stare4(x1, x2):
 print("S4-STOP")
 newState = "S4"
 return (newState, x1, x2)
if __name__ == "__main__":
 m = StateMachine()
 m.add_state("S0", stare0, 1, 0)
 m.add_state("S1", stare1)
 m.add_state("S2", stare2)
 m.add_state("S3", stare3)
 m.add_state("S4", None, 0, 1)

x1x2=[(1,0),(1,1),(0,0),(1,1),(0,1),(1,1),(1,0)]
for i in x1x2:
 print(i)
 m.setCargo(*i)
 m.run()
```