

# Observații generale

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

Paráuligone de Programación

Titular

Miguel Zapatero

# 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 asistenții 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 bilete 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 motive de 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

# 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

# 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 SSI ID 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)

# Cum este cu înregistrările????

- No way!!!
- Why?

# 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...*

# Definirea unui limbaj

- **Sintaxa:** se definește gramatica unui limbaj
  - Ce înseamnă o propoziție corectă? dar un program correct?
  - 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

# Criterii generale de proiectare a unui limbaj

- |  |   |
|--|---|
| <ul style="list-style-type: none"><li>• Putere (excelează în rezolvarea...)</li><li>• Flexibilitate</li><li>• Expresivitate</li><li>• Ușor de scris (vezi C vs Pascal)</li><li>• Implementare eficientă</li><li>• Support pentru abstractizări</li></ul> | <ul style="list-style-type: none"><li>□ Simplitate</li><li>□ Claritate</li><li>□ Consistență (ortogonalitate) (puține structuri de control cu puține posibile combinații)</li><li>□ Usurință în urmărirea cod</li><li>□ Aplicabilitatea în domeniul problemei (<math>\geq \text{gen4}</math>)</li><li>□ Portabilitate</li></ul> |
|--|---|

# Alegerea Sistemului de Operare

# Ergonomie

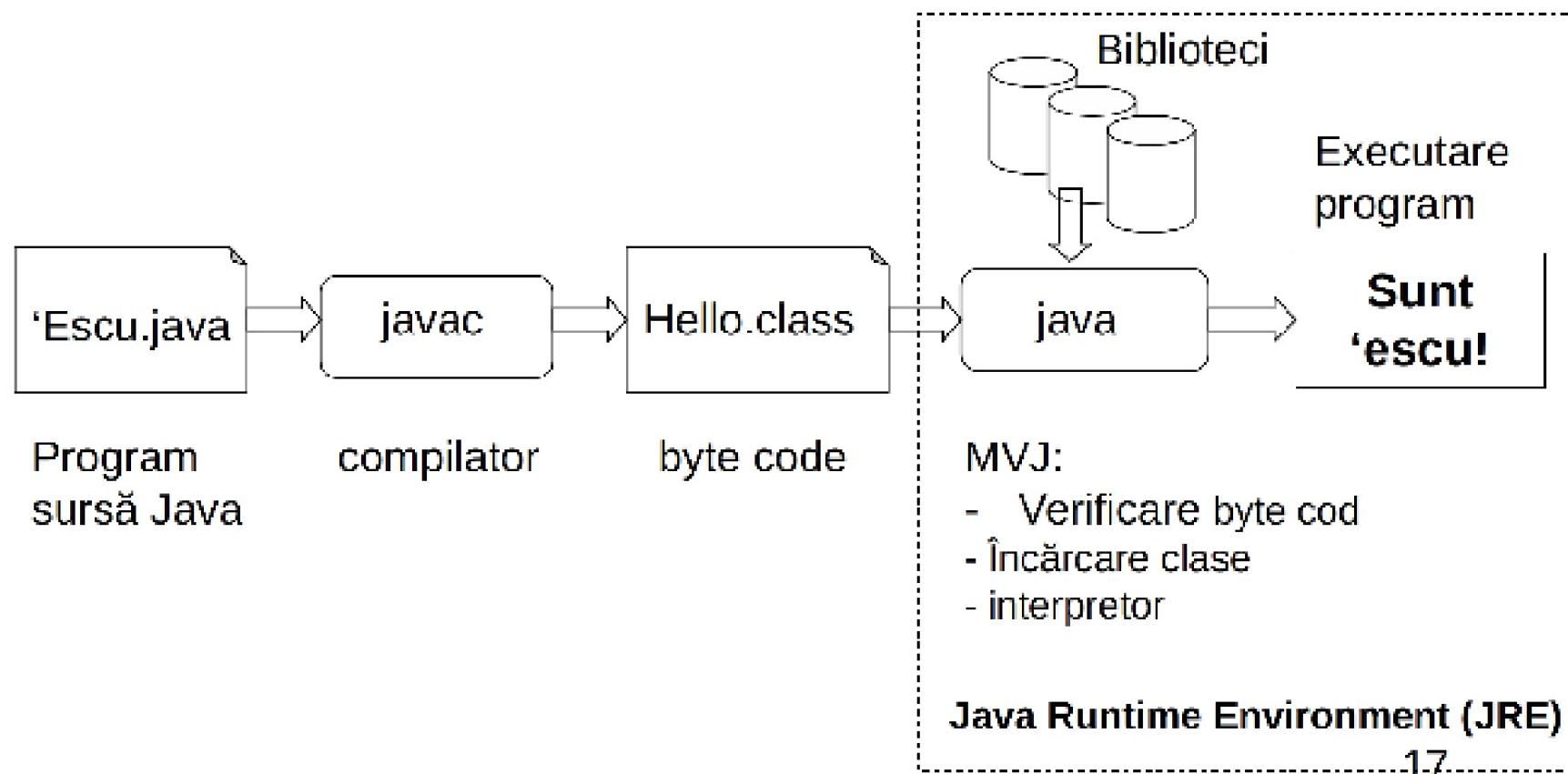
1. Alegere monitor – mat
2. Dimensiune minima ecran
3. Alegere densitate / rezoluție ecran
4. Reglare luminozitate și culoarea albastră
5. Poziție corectă de lucru – desktop/laptop
6. Sporturi recomandate

# 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.

# Java: o strategie hibridă

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



# Program Interpretat vs Compilat

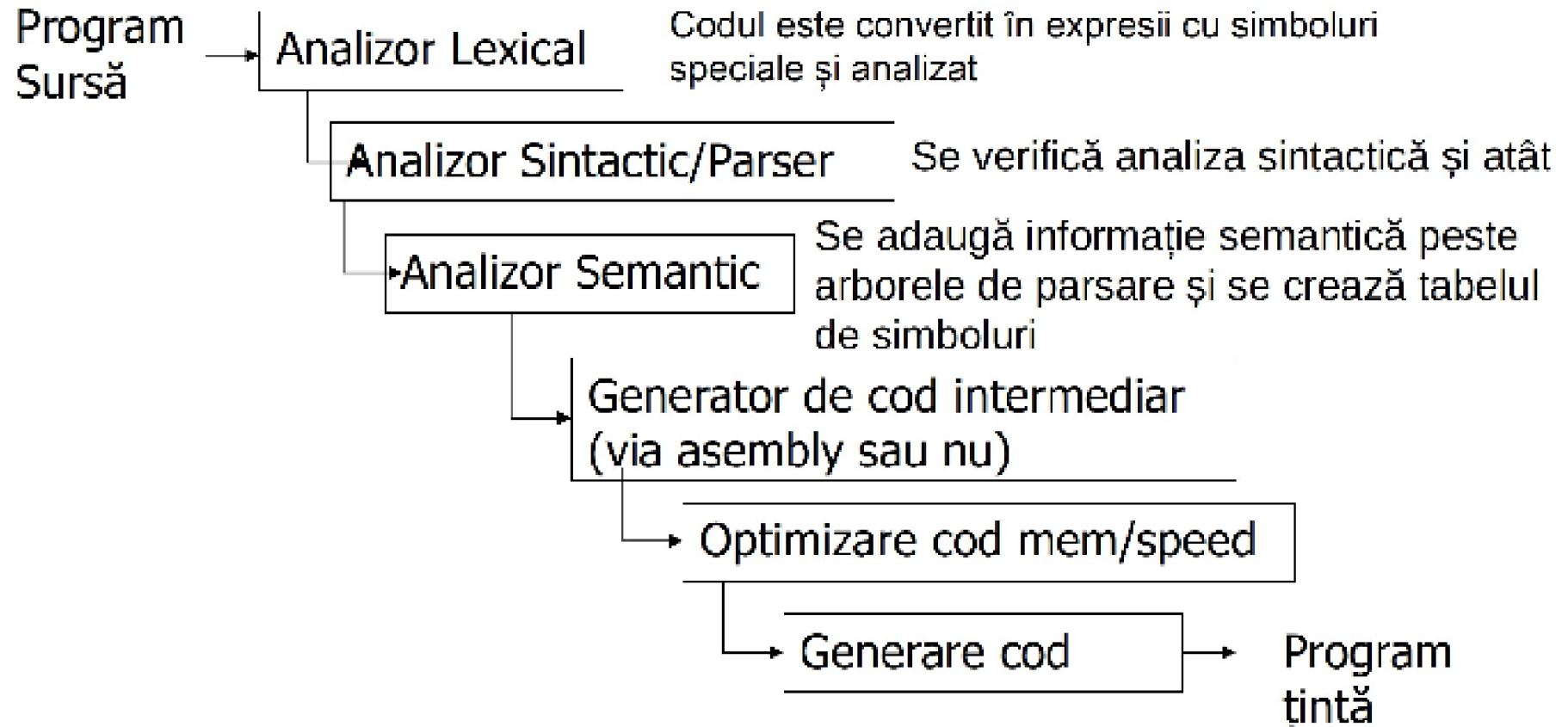
## Interpretat

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

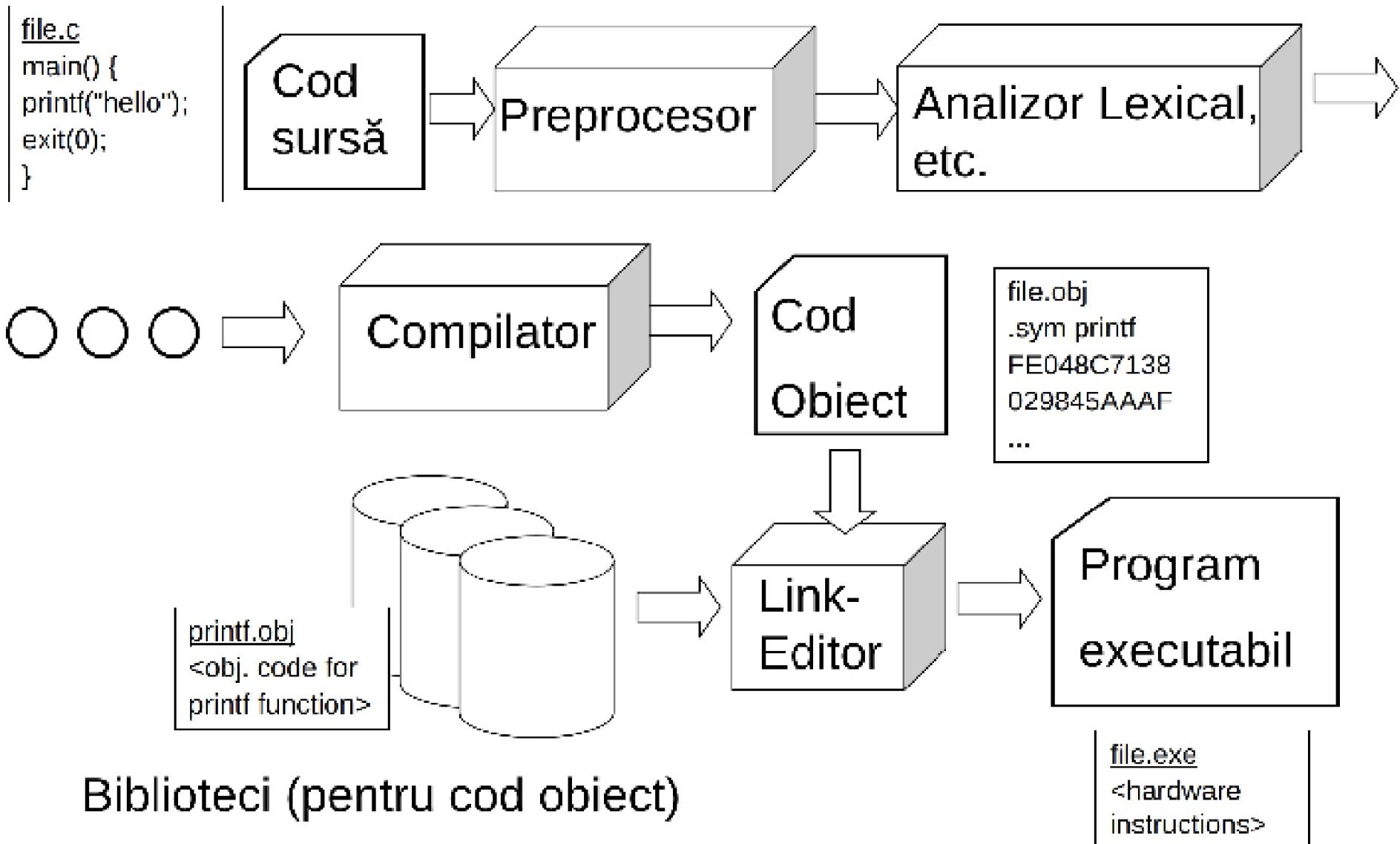
## Compilat

- 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 săi

# Etape ale compilării

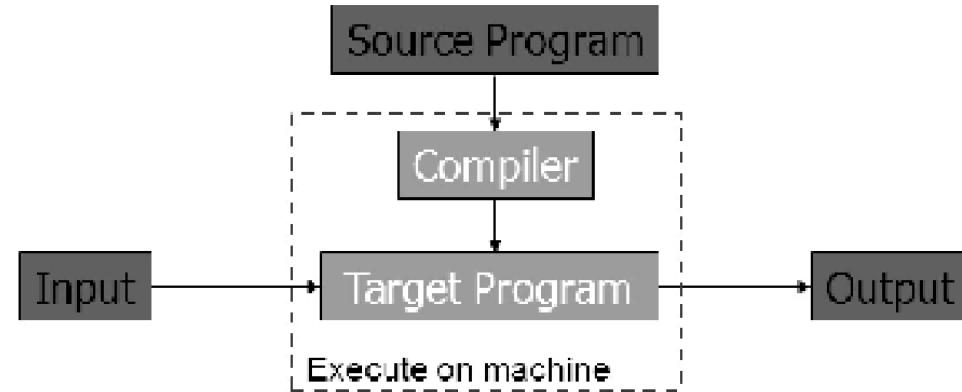


# Compilarea unui program

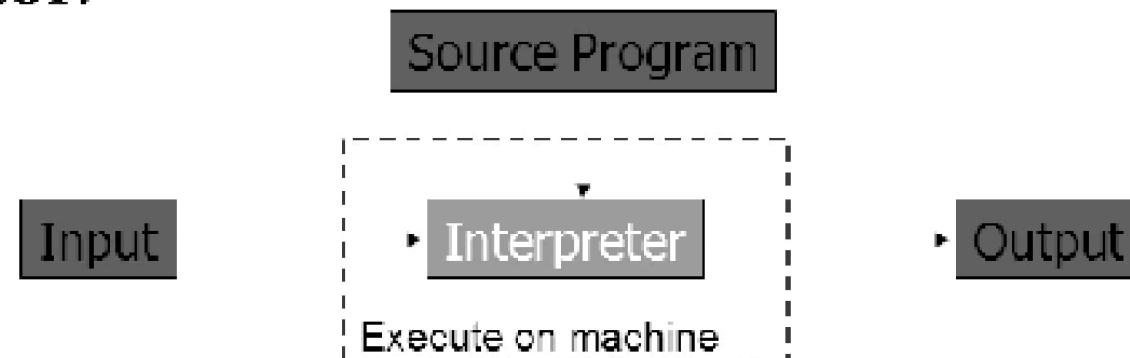


# Strategii de implementare

- **Compilator:**



- **Interpreter:**



- **Hibrid:**

## ~~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

# 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
```

# Depanare pas cu pas???

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

## 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.

# 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**

## ~~Cele mai cunoscute limbaje din generația a treia~~

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

## 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

## 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

$\text{total} = \text{quantity} * \text{price} * (1 + \text{taxrate});$

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 );
```

# 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
```

## ~~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 - Bel Lab, Fortran de IBM, Java de Sun )
  - Pentru necesități guvernamentale (Ada/Clips de către DoD al US).
  - Preferate de experti:

## ~~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

# Exemplu - 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**

# Limbaje din generația a patra

Acstea sunt deja orientate pe anumite tipuri de aplicatii

SQL pentru baze de date

Limbajul Postscript pentru descrierea unei pagini folosit de imprimante

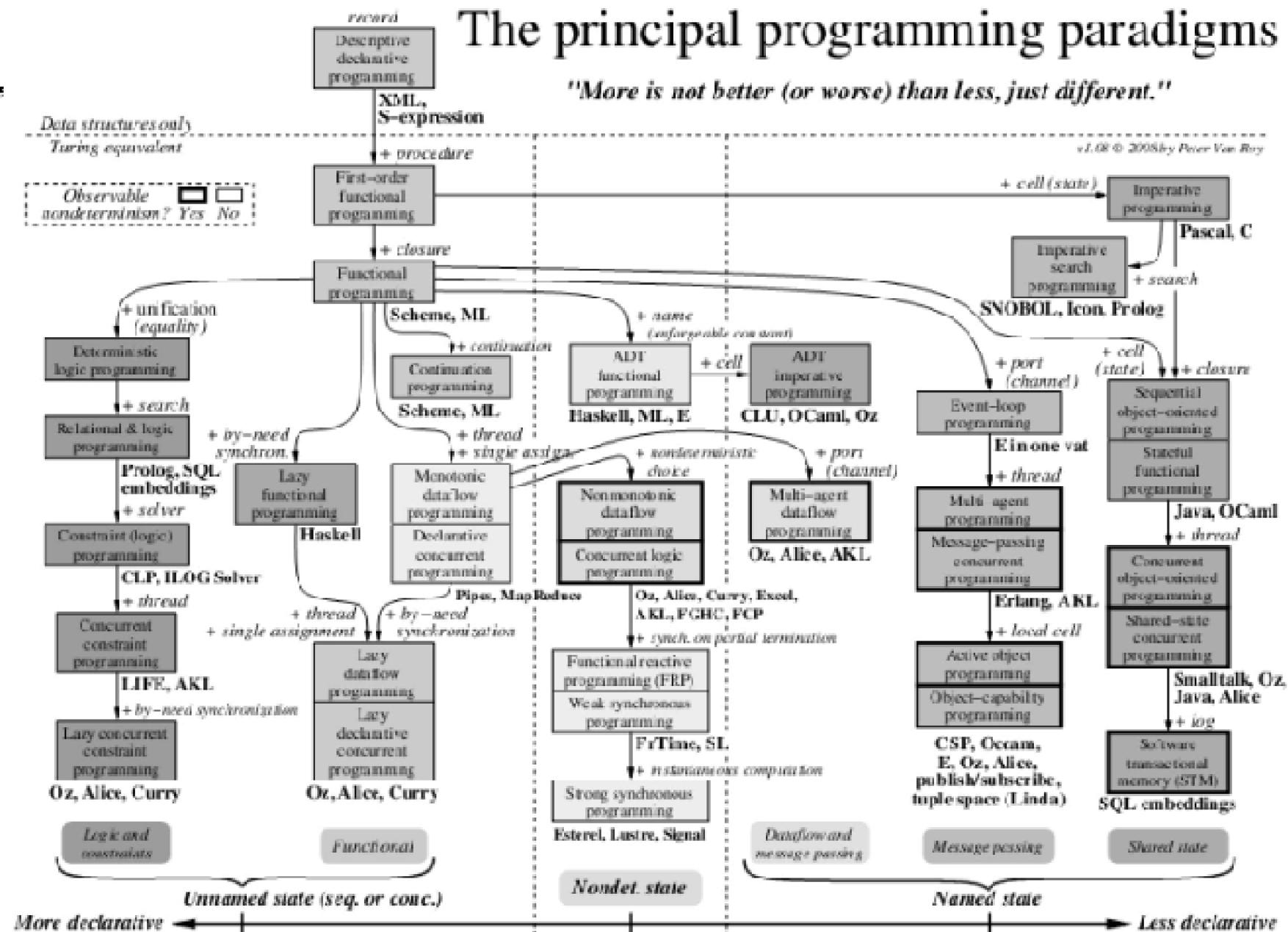
PDF pentru documente on-line

HTML and PHP pentru continut World Wide Web

Mathematica

# The principal programming paradigms

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



## Ce este o paradigmă de programare?

- D<sub>1</sub> (generală) Caz exemplar, **model**, **prototip**, situație ideală, structură tip, **arhetip** standard și.a.
- D<sub>2</sub>. (în filozofie, la L.Wittgenstein) Modelele filosofice, acele "tipare" care orientează gândirea noastră în direcții predeterminate.,
- D<sub>3</sub> (în filozofia limbajului) Listă de cazuri tipice de jocuri lingvistice prin care putem înțelege conceptul general..."

Daca doar muncim...

# 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 cateva 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 proiectează și se validează aplicația stabilindu-se timpul necesar dezvoltării precum și resursele necesare.

Se trece la implementare

Se face verificarea și depanare

Se trimite versiunea beta pentru testare reală

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

**Resurse folosite (linear cu depășirea daca este cazul)**

# **Programare structurată**

- Una din paradigmile fundamentale de programare și poate fi văzută ca un subset al programării procedurale.
- Programarea structurată respectă teorema lui Boehm și Jacopini: Orice program poate fi realizat prin combinarea structurilor de control: secvența, selecția și iterația, iar fiecare structură de control are o singură intrare și o singură ieșire (deja contestată)
- Modelul este suportat de următoarele limbaje C, C++, C#, Pascal, Delphi, JavaScript, JScript, Java, Pascal, Perl, VBScript, Visual Basic

# Programarea procedurală

- Una din paradigmile fundamentale de programare sinonim pentru *programarea imperativă* dar se mai referă și la paradigma de programare bazată pe conceptul de *apel procedural*
- Procedurile numite alteori rutine/subrutine sau funcții care sunt similare cu cele folosite în *programarea funcțională* conțin o serie de pași de executat. Orice procedură poate fi chemată din orice punct al programului inclusiv din ea însăși (recursivitatea)
- Schimbul de date între funcții se realizează prin transferul de parametri (argumente de intrare și valori rezultate în urma execuției procedurii)
- Un nivel superior de modularizare este obținut prin intermediul programării orientate obiect. Modelul este suportat de următoarele limbaje
- BASIC C C++, C# , ColdFusion, COBOL, Component Pascal, D, Delphi, JavaScript, JScript, Fortran, Java, Maple, Mathematica MATLAB, Pascal, VBScript Visual Basic

# Programarea nestructurată

- Bloc continuu
- Salt direct

## ASM

```
.model small
.stack 1024
.data
z db ? ; n
MOV AX, @data
MOV DS, AX
MOV al, 3 ; m=3
ADD al, 5 ; m=m+5
Jmp etl
etl: MOV z,al ; n=5
MOV al,z
MOV ah, 4ch
INT 21h
end start
```

## Basic

```
10 rem program exemplu
20 m=3
30 m=m+5
40 goto 50
50 n=5
60 m=n
60 end
```

# Paradigme de programare

Action	Declarative (vs:Imperative)	Expression-oriented	Non-structured (vs:Structured)
Agent-oriented	Constraint	Feature-oriented	Array
Aspect-oriented	Dataflow	Function-level (vs:Value-level)	Nondeterministic
Automata-based	Cell-oriented (spreadsheets)	Generic	Parallel computing
Component-based	Reactive	Imperative (vs:Declarative)	Process-oriented
Flow-based	Intensional	Procedural	Programming in the large and small
Pipelined	Functional	Language-oriented	Semantic
Concatenative	Logic	Discipline-specific	Structured (vs:Non-structured)
Concurrent computing	Abductive logic	Domain-specific	Modular (vs:Monolithic)
Relativistic programming	Answer set	Grammar-oriented	Object-oriented (OOP)
Data-driven	Constraint logic	Dialecting	By separation of concerns:
	Functional logic	Intentional	Aspect-oriented
	Inductive logic	Metaprogramming	Role-oriented
	End-user programming	Automatic	Subject-oriented
	Event-driven	Reflective	35

# **Programare bazată pe componente**

- Este similară oop dar bottom-up
- Se lucrărează la nivel de cutie neagră, descrieri UML, specifică ingineriei software Folosește componenta care poate fi definită ca un obiect/service descris în urma unei specificații
- De multe ori particularizată în OOP
- Există diferențe față de OOP deoarece în loc să se proiecteze conform unui model descris de proiectant se folosesc componentele disponibile pentru a asambla o soluție viabilă
- Limbaje care suportă acesta paradigmă
- Visual Basic Extensions, OCX/ActiveX/COM and DCOM from Microsoft
- Enterprise Java Beans from Sun Microsystems

# Programarea orientată obiect

- Obiectele POO sunt de obicei reprezentări ale obiectelor din viața reală (*domeniul problemei*), astfel încât programele realizate prin tehnica POO sunt mai ușor de înțeles, de depanat și de extins decât programele procedurale.
- Limbaje pure oo: Eiffel, Emerald, JADE, Obix, Ruby, Scala, Smalltalk, Self
- Limbaje dominant oo: Delphi/Object Pascal, C++, Java, C#, VB.NET, Python.
- Limbaje cu extensii oo: Pascal, Visual Basic (derived from BASIC), Fortran, Perl, COBOL 2002, PHP, ABAP.
- Limbaje bazate pe principiile oo dar nerespectând standardul: Oberon (Oberon-1 or Oberon-2) and Common Lisp.

# **Programarea modulară**

- Trebuie ținut cont de următoarele reguli de modularizare:
- Structura fiecărui modul trebuie să fie suficient de simplă pentru a putea fi complet înțeleasă.
- Detaliile sistemului care se presupune că se vor modifica independent vor fi plasate în module diferite.
- Singurele legături între module vor fi acelea a căror modificare este improbabilă.
- Orice structură de date este încapsulată într-un modul; ea poate fi accesată direct din interiorul modulului, dar nu poate fi accesată din afara modulului decât prin intermediul variabilelor globale conținute în acel modul.
- Limbaje Modula și Ada

# Programarea modulară

- Este aceea care a deschis calea abstractizării datelor.
- Modulele furnizau o sintactică pentru descompunerea programelor în componente mai mult sau mai puțin independente.
- Modulele sunt independente, atomice Principul de bază: **încapsularea**
- Modulul are două componente:
  - interfața și implementarea
  - Tehnica este folosita în C cand se realizează proiectele din mai multe module.

# **Programare orientată pe subiect**

- Complementar OOP pentru dezvoltarea sistemelor de dimensiuni mari C++, Java, Smalltalk)
- Dezvoltat de IBM și își propune rezolvarea următoarelor probleme:
- crearea de extensii la programe și configurații ale acestora
  - fară a modifica sursa inițială
  - creșterea nivelului de încapsulare pentru platforme, versiuni și facilități multiple

# **Pattern-uri pentru proiectare**

- **Algoritmi pentru strategie**
- **Patternuri computationale**
- **Patternuri pentru executie**
- **Paternuri pentru implementare**
- **Pattern-uri structurale**

# **Programare "post object"**

- Bazată pe design patterns:
  - Proiectarea sistemelor oo este o activitate dificilă, iar proiectarea sistemelor oo reutilizabile este încă și mai grea.
  - Soluția trebuie să fie specifică problemei, dar totodată suficient de generală pentru a putea fi aplicată și pe viitor, pentru a evita în ultima instanță "reinventarea roții" de fiecare dată (sau cel puțin pentru a minimiza acest lucru).
  - Un proiectant familiar cu asemenea modele le va putea aplica repede, fără a trebui să le redescopere.

## **Programare bazată pe aspect (din oop)**

- AOP reprezintă o noua metodologie care vine ca o completare la OOP ( sau la alte paradigmă inclusiv cea procedurală), aducând câteva principii noi care imbogățesc această paradigmă de bază.
- Nu este o nouă paradigmă (a se vedea termenul metodologie pe care îl folosim) ci o simplă completare la OOP.

# **Programare bazată pe tabele**

- Scopul modelării datelor este descrierea formală dar intuitivă, o schemă conceptuală a specificațiilor de utilizare a informației în cadrul unei aplicații.
- Entitățile sunt descrise cu ajutorul atributelor și pot fi organizate în ierarhii care exprimă derivarea unui concept specific din unul general.
- Relațiile sunt caracterizate de condiții de cardinalitate care impun restricții asupra numărului de relații pe care le poate avea un obiect.
- Foxpro, oracle, mysql, db2

# **Programare bazată pe constrângeri**

- În acest model relațiile intre variabile pot fi descrise sub forma de constrângeri.
- În general sunt asociate la diverse limbaje cum ar fi programarea logică, funcțională și imperativă
- În general sunt bazate pe Prolog
  - B-Prolog (Prolog based, proprietary)
  - CHIP V5 (Prolog based, also includes C++ and C libraries, proprietary)
  - Ciao Prolog (Prolog based, Free software: GPL/LGPL)
  - ECLiPSe (Prolog based, proprietary)
  - SICStus (Prolog based, proprietary)
  - GNU Prolog
  - YAP Prolog

# **Programare orientată pe concept**

- Pentru a descrie un sistem complex nouă abordare folosește conceptele ca elemente principale în construcția programului.
- Un concept este alcătuit dintr-o clasă de obiecte și o clasă de referințe.
- Un astfel de program va folosi relația de incluziune a conceptelor în loc de moștenire .
- Groovy, Python, Ruby, AspectJ, Meta-Programming System ( MPS )

# Programare orientată pe concept

- Principalul element constructiv este conceptul. Se referă la metode de afaceri și la reprezentări și accesul acestora.
- Una din principalele presupuneri de bază este că orice sistem poate fi văzut ca fiind alcătuit din două componente majore
  - *Metode de afaceri Business methods (BMs)* care sunt utilizate explicit în codul sursă și
  - *Reprezentare și acces Representation and access (RA)* metode care sunt apelate implicit atunci cand sunt folosite metodele de afaceri

# **Programare bazată pe eveniment**

- În loc să se aștepte completarea execuției unei comenzi pentru a procesa informația, în acest caz sistemul este preprogramat cu o buclă pe eveniment
- Practic programarea se rezumă la descrierea funcțiilor declanșator pentru a obține comportamentul dorit Exemple tipice, OS, GUI,
- Limbaje, C++, Visual Basic, Java, C#, Visual Age, Lab Windows CVI,

# Programare declarativă

- Termenul are două înțelesuri bine definite
  1. un program este declarativ dacă descrie „ceva” și mai puțin accent despre cum este creat acel ceva  
(ex pag web, și limbajele asociate)
  2. un program este declarativ dacă este scris într-un limbaj care suportă modelul programării funcționale, logice sau cu constringeri
- Modelul este specific următoarelor limbaje Ruby on Rails, JUnit TK Solver, Haskell, Lisp, Oz, ABSET, Lustre, MetaPost, OpenLaszlo, Prolog, SQL, XSL Transformations

# **Programarea imperativă**

- În general este un termen definit prin opoziție cu programarea declarativa care descrie realizarea operațiilor de calcul folosind termenii (concepțele) de stare și declarație/instrucțiune care va conduce la modificarea stării
- Modelul este suportat de următoarele limbaje FORTRAN, ALGOL, PASCAL, C și ADA, JAVA, PHP, BASIC

# **Programare funcțională**

- realizează sinteza automată a tipurilor => depistarea erorilor subtile generate de declararea și utilizarea eronată a tipurilor datelor prelucrate.
  - pot fi implementate paralel
- Probleme:
  - este greu de estimat necesarul de resurse (timp, spațiu);
  - este greu de incorporat input/output.
- Limbaje care suportă total sau parțial modelul:  
Lambda Calculus, IPL, clips, LISP, Common LISP, Haskel, ML, SML, OCaml, ERlang, Mapple, Matematica, TCL, PERL, Python, Ruby

# Programare funcțională

- Tratează orice calcul ca o evaluare a funcțiilor matematice
- Rezolvarea unei probleme înseamnă descrierea dependenței
  - $rezultate = f(date)$
- Instrumentul prelucrărilor este funcția (d.p.d.v matematic).
- Mecanismul de control principal: apelul (aplicarea) funcțiilor => programare aplicativă.
- Caracteristicile limbajelor funcționale:
  - au un suport matematic solid;
  - au un grad ridicat de abstractizare;

# TIOBE Programming Community Index for 2020

Jan 2021	Feb 2020	Change	Programming Language	Ratings	Change
1	2	▲	C	18.34%	+0.43%
2	1	▼	Java	11.29%	-6.07%
3	3		Python	10.86%	+1.62%
4	4		C++	6.88%	+0.71%
5	5		C#	4.44%	-1.46%
6	6		Visual Basic	4.33%	-1.53%
7	7		JavaScript	2.27%	+0.21%
8	8		PHP	1.75%	-0.27%
9	9		SQL	1.72%	+0.20%
10	12	▲	Assembly language	1.65%	+0.54%
11	13	▲	R	1.58%	+0.55%
12	26	▲	Groovy	1.50%	+1.08%
13	1	▼	Go	1.28%	+0.15%
14	15	▲	Ruby	1.23%	+0.39%
15	10	▼	Swift	1.13%	-0.53%
16	16		MATLAB	1.06%	+0.27%
17	18	▲	Delphi/Object Pascal	1.02%	+0.27%
18	22	▲	Classic Visual Basic	1.01%	+0.40%
19	19		Perl	0.93%	-0.23%
20	20		Objective-C	0.80%	+0.20%

# 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%

# 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		

## **Programare "value level" & "function level"**

- **Programare "value level"**

- Este legată de analiza, algebra tipurilor de date precum și de Lambda Calcul Limbaje (LISP, ISWIM, Scheme)

- **Programare "function level"**

- Un program este creat direct din programe care sunt date prin combinarea lor cu operații speciale (programm forming) în scopul obținerii programului final care are caracteristicile dorite

- Exemple de limbaje FP, FL și J

# Programare reflexivă

- **reflective care poate însemna și a medita, a reflecta, reciprocitate**
- O extensie a oop care permite adăugarea unor facilități de autooptimizare applicațiilor
- Proces?
- Ca avantaj se poate menționa accesul la variabilele proprii în cazul execuției.
- Java, C#, Ruby, PHP