PROGRAMARE ORIENTATĂ PE OBIECTE

GENGE BÉLA

Capitolul 1
Introducere

Administrativ

- Conf. dr.ing. Béla GENGE (titular curs + laborator).
- Pagina Web: www.ibs.ro/~bela.
- Adresa email: bela.genge@ing.upm.ro.
- Examen: scris (50% nota finală) – fără surse.
- Laborator: evaluări pe parcurs + proiect final.
- Condiție prezentare la examen: min. nota 5 la laborator.

Resurse de studiu

- Pagina Web a disciplinei: http://www.ibs.ro/~bela/Teachings/OOP/oop.html
- Oracle Java JDK documentation: https://docs.oracle.com/javase/9/
- Bibliotecă:
  - Lefkovits Szidonia, Lefkovits Laszlo: Bazele programării orientate pe obiecte în limbajul Java, 2017.
  - Gabriela Varvara, Limbajul Java și principiile de programare orientată obiect, 2002.
  - ... altele.

Programarea procedurală

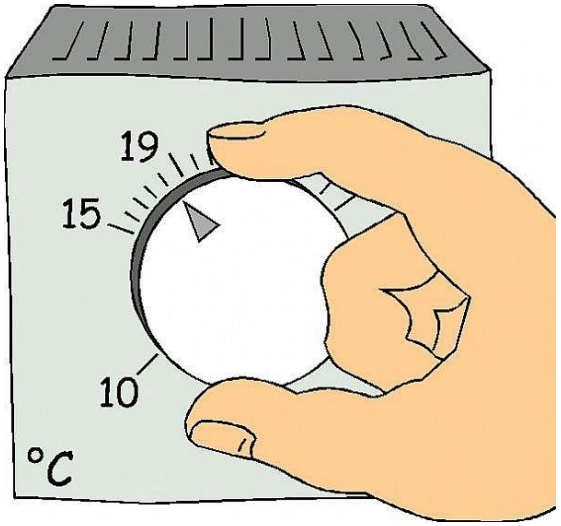
Caracteristici:

- Organizarea codului pe funcții/proceduri.
- Organizarea datelor pe tipuri de bază și structuri de date.
- Vizibilitatea datelor: variabile locale/globale.
- Protejarea datelor:
  - Declarare variabile locale.
  - Transmiterea variabilelor ca parametrii – pointeri.
- Protejarea funcțiilor/procedurilor:
  - Fișiere diferite: .h și .cpp în C/C++.
  - Module diferite: .dll / .so (funcții **statice**)

Problema principală

Exemplu:

- Termostat.
- Obiect “black-box”.
- “Interfața”:
  - Setare temperatura dorită.
  - Citire temperatura camerei.



Problema:

- Cum implementăm în C?

Programarea orientată pe obiecte

POO este despre **obiecte**

Caracteristici preliminare:

- Stare: variabile denumite **atribute** sau **câmpuri**.
- Funcții: denumite **metode**.
- Elementul de bază: **clasa**.
- Instanțierea.
- Obiecte.

Noțiuni fundamentale

Clasa:

- O implementare a unui concept sub forma unei colecții de câmpuri și metode.
- O colecție de obiecte cu aceeași structură și același comportament.

Obiectul:

- O instanță a clasei care definește o anume stare dată de valorile câmpurilor sale.

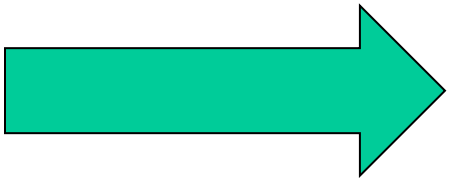
Instanțierea:

- Mecanismul de creare a unui obiect pornind de la implementarea unei clase.

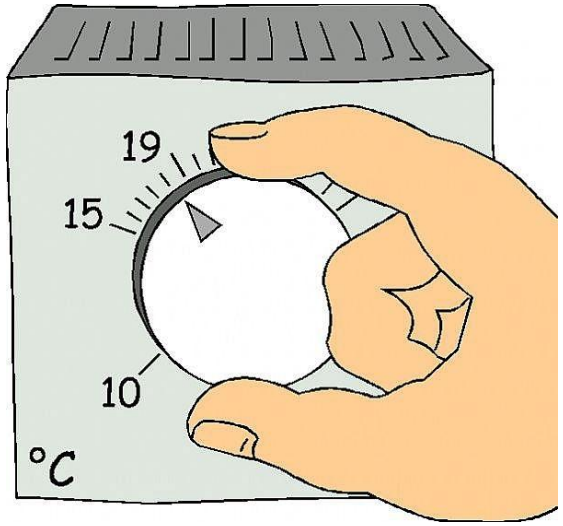
De la clasă la instanță

Clasa

Instanțiere



Obiectul



Principalele proprietăți ale POO

Încapsularea:

- Asigură gruparea datelor și funcțiilor (metodelor) într-o singură structură de date, definind modul de ascundere a implementării și modalitatea prin care obiectul poate interacționa cu restul programului.

Polimorfism:

- Comportament multiplu. Decizia asupra operației executate se ia în timpul rulării.

Moștenirea:

- Crearea de tipuri noi de date pornind de la clasele de bază.

Primul limbaj orientat obiect

C++ (anii ‘80):

- Este un limbaj hibrid, înglobează programare procedurală și programare orientată obiect.

```
1   #include "ExpatWrapper.h"
2   #include <stdio.h>
3   #include <string>
4   using namespace std;
5   int main()
6   {
7       ExpatWrapper ew;
8
9       if ( !ew.initialise() )
10          printf("Error");
11      else
12          printf("Not an error");
13
14
15      string s = "<element1 attr=\"test\">value</element1>";
16
17      if ( !ew.parse( s.c_str(), true ) ) {
18          printf("ERROR parsing!!!\n");
19      }
20  }
```

Primul limbaj orientat obiect

```
5  #if !defined(__EXPAT_WRAPPER_H__)
6  #define __EXPAT_WRAPPER_H__
7
8  #include "expat.h"
9
10 class ExpatWrapper
11 {
12 public:
13     ExpatWrapper();
14     virtual ~ExpatWrapper();
15
16     bool initialise ( void );
17     void destroy ( void );
18
19 public:
20
21     bool parse ( const char *pszBuffer, bool bIsFinal = true );
22     void enableHandlers( void );
23
```

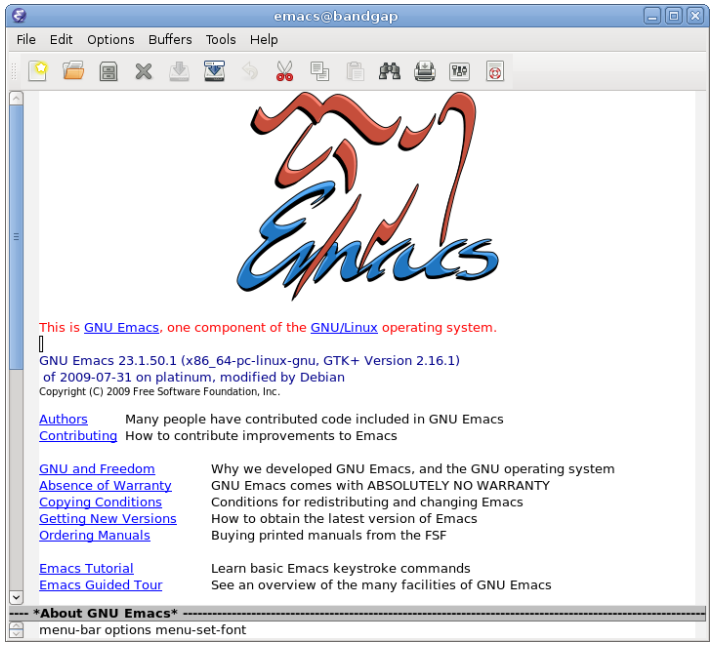
Primul limbaj orientat obiect

```
5   #include "ExpatWrapper.h"
6   #include <string.h>
7   #include <stdio.h>
8
9   ////////////////////////////////////////////////////////////////////////
10  // Construction/Destruction
11  ////////////////////////////////////////////////////////////////////////
12
13  ExpatWrapper::ExpatWrapper():
14      m_pParser( NULL )
15  {
16  }
17
18  ExpatWrapper::~ExpatWrapper()
19  {
20      if ( NULL != m_pParser ) {
21          XML_ParserFree( m_pParser );
22      }
23
24      m_pParser = NULL;
25  }
26
27  bool ExpatWrapper::initialise( void )
28  {
29      // Destroy old parser
30      destroy();
31
32      // Create a new parser
33      m_pParser = XML_ParserCreate_MM ( NULL, NULL, NULL );
34      if ( NULL == m_pParser ) {
35          return false;
```

Limbajul Java

Java și-a început viața în jurul ‘91:

- Sun a început dezvoltarea de software pentru electrocasnice.
- Leaderul proiectului: James Gosling (emacs).



Limbajul Java

Java și-a început viața în jurul ‘91:

- La început s-a folosit C++.
- Probleme cu tratarea erorilor, s-a pornit de la ideea modificării C++.
- S-a ajuns la un nou limbaj: inițial OAK.

Limbajul Java

Java și-a început viața în jurul ‘91:

- Problema cu numele OAK: marcă înregistrată de firmă producătoare de semiconductoare.
- S-a ajuns la *Java* dintr-o serie de discuții care doreau să încapsuleze esența limbajului:
  - Dynamic
  - Alive
  - Jolt
  - Impact
  - Revolutionary
  - ... mai multe detalii aici:

http://www.javaworld.com/article/2077265/core-java/so-why-did-they-decide-to-call-it-java-.html

De ce studiem Java?

Despre Java

Simplitate:

- Lipsa pointerilor.
- Adresa metodei apelate se stabilește în timpul rulării.
- Eliberarea memoriei se realizează automat prin *Java Garbage Collector.*

Orientat Obiect:

- Cu precădere construit după principii OO.

Mecanisme pentru tratarea erorilor:

- Fiind dezvoltat pentru echipamente hardware există metode avansate pentru tratarea erorilor.

Despre Java

Mecanisme pentru tratare erori (cont.):

- Indexarea tablourilor este verificată.

Securitate sporită:

- Lipsa pointerilor elimină breșe serioase de securitate.
- Vectorii sunt obiecte, indexarea verificată => depășirile de memorie pot fi verificate.
- Șirurile de caractere sunt modificate doar prin apeluri de metode.
- Transformările de tip (*cast*) trebuie forțate, menționate explicit.

Despre Java

Securitate sporită (cont.):

- 4 nivele de protejare a metodelor și câmpurilor.
- Obiectele au definite valori inițiale.

Portabilitate:

- Codul compilat este multi-platformă (în limite rezonabile!).

Mașina Virtuală Java

Problematică:

- Înainte de execuție, un program trebuie *transformat* într-o formă ce poate fi executată de platforma destinație.
- Transformarea din cod sursă în executabil: **compilare – compilator**.
- Execuția programului sursă: **translator.**
- **Compilarea duce de regulă la o execuție mai rapidă**.

Mașina Virtuală Java

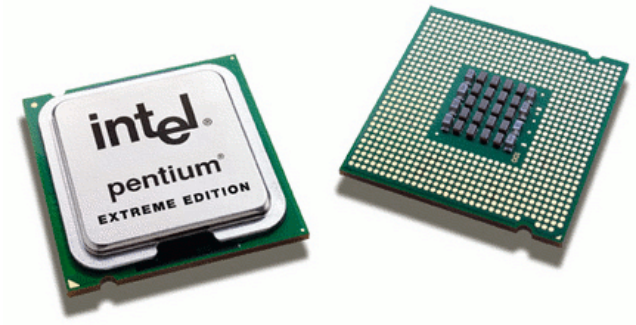
Soluția adoptată de Java:

- Compilare + interpretare.
- Codul sursă este **compilat** într-o formă intermediară: **bytecode**.
- Bytecode e după aceea **interpretat+compilat** de o mașină virtuală**.**

Mașina Virtuală Java

Despre Java Virtual Machine (JVM):

- JVM este un “procesor” cu propriul set de instrucțiuni.
- La primul apel al metodei compilează codul pentru target.
- Java Runtime Environment (JRE) instalează JVM.
- Java Development Kit (JDK) include JRE.

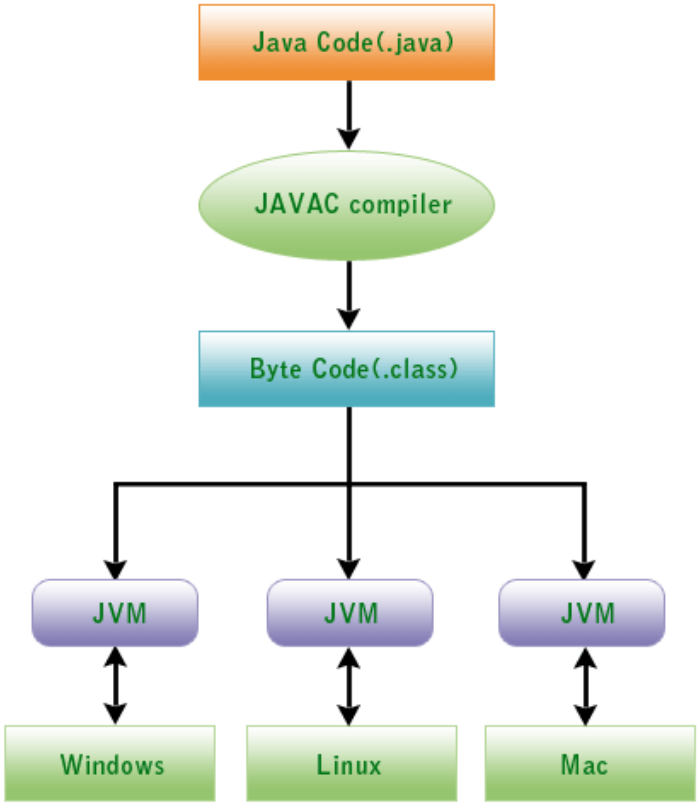


Mașina Virtuală Java

Sursa: www.devmanuals.com

Despre Java Virtual Machine (JVM):

- Prin JVM se asigură compilarea o singură dată și rularea pe diverse platforme.



Primul Program Java

Scrierea codului:

- Orice editor de text.
- Salvare cu extensia: .java
- Denumirea clasei *publice* aceeași cu denumirea fișierului.
- Pot fi declarate mai multe clase în același fișier. O singură clasă va fi publică.

Compilarea:

- javac HelloWorld.java => HelloWorld.class

Rularea:

- java HelloWorld

Configurare cale: variabila PATH.

Primul Program Java

```
public class HelloWorld {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        System.out.println("Hello World");
    }

}
```

Tipuri Principale de Programe Java

De sine stătătoare:

- Aplicațiile includ un punct de intrare – metoda *main*.
- Nu necesită integrarea într-o altă aplicație.
- **Necesită instalarea JRE**.

Tipuri Principale de Programe Java

Integrate:

- Ex.: browsere Web.
- **Appleturi**: mediu de execuție Browsere.
- Astăzi nu mai sunt “la modă”:
  - API-ul inițial prezenta numeroase dezavantaje, greoi de utilizat (GUI).
  - “Concurența” – Flash, JavaScript – au făcut operațiile mult mai simple.
  - Appleturile necesită mai mult timp pentru încărcare, au codul mai mare.
  - Microsoft nu a dorit includerea în Windows XP a JVM, a eliminat suportul Java din Internet Explorer în 2004 => nu este sigur că aplicațiile rulează pe sisteme noi
  - Multe probleme de securitate.
  - Applet un mediu general de programare, Flash un mediu grafic mult mai puternic.

Tipuri Principale de Programe Java

Exemplu Applet:

```
3  import java.applet.Applet;
4  import java.awt.*;
5
6  public class HelloWorld extends Applet {
7      public void paint (Graphics g) {
8          g.drawString("Hello World", 20, 20);
9      }
10 }
```

Integrarea în browser:

```
1  <html>
2      <body>
3          <applet code="HelloWorld.class" width=300 height=300>
4          </applet>
5      </body>
6  </html>
7
8
```

Elemente de Bază ale Limbajului

Caracteristici:

- Fiind construit pe C, există multe asemănări.
- Folosește caractere Unicode pe 16 biți.
- Comentariile se scriu la fel ca în C.
- Principii de definire a denumirilor:
  - Denumirea claselor începe cu literă mare, fiecare cuvânt cu literă mare: **Animal**, **StudentAudient**.
  - Denumirea metodelor începe cu literă mică, fiecare cuvânt cu literă mare: **getCount()**, **runApplication().**
  - Constantele cu majuscule: **ALB**, **VALMAX**.
  - Atributele: încep cu ‘_’ (sau ‘m’) urmate de minuscule.

Elemente de Bază ale Limbajului

Constante literale:

- Întregi:
  - +150, -500.
  - 052, 0127 (baza 8).
  - 0x321, 0xABC (baza 16).
- Reale:
  - 5.171 (double) – 8 octeți.
  - 4.157F (float) – 4 octeți.
  - 3.14E10 (double) – 8 octeți.
- Caractere:
  - \uhhhh (format Unicode).
  - ‘\u0042’, ‘\u0062’ (coduri caractere Unicode).
  - ‘A’, ‘b’

Elemente de Bază ale Limbajului

Constante literale:

- Șiruri de caractere:
  - “abc”.
  - “A\u0042\u0043”.

Tipuri de date:

- Primitive și referință:

Elemente de Bază ale Limbajului

Tipuri primitive:

Java Primitive Data Types

Type	Values	Default	Size	Range
byte	signed integers	0	8 bits	-128 to 127
short	signed integers	0	16 bits	-32768 to 32767
int	signed integers	0	32 bits	-2147483648 to 2147483647
long	signed integers	0	64 bits	-9223372036854775808 to 9223372036854775807
float	IEEE 754 floating point	0.0	32 bits	+/-1.4E-45 to +/-3.4028235E+38, +/-infinity, +/-0, NAN
double	IEEE 754 floating point	0.0	64 bits	+/-4.9E-324 to +/-1.7976931348623157E+308, +/-infinity, +/-0, NaN
char	Unicode character	\u0000	16 bits	\u0000 to \uFFFF
boolean	`true, false`	`false`	1 bit used in 32 bit integer	NA

Sursa: http://www.write-technical.com/

Observație! Nu există tipul unsigned – pentru evitarea confuziilor legate de utilizarea împreună a termenilor cu semn și a celora fără semn.

Elemente de Bază ale Limbajului

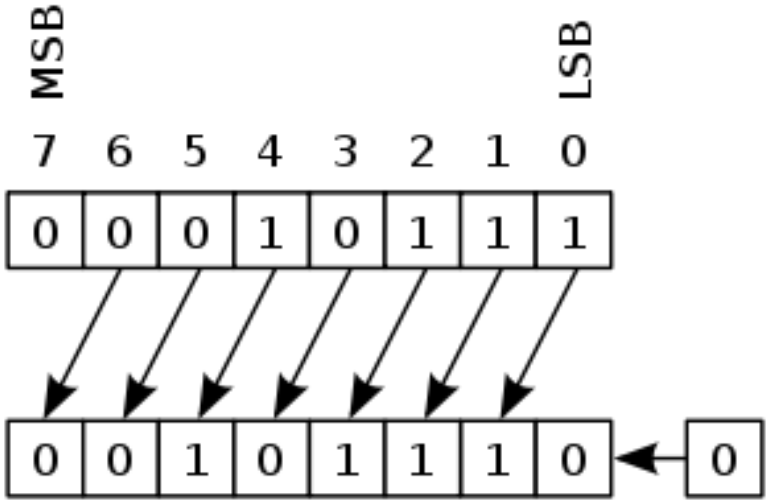
Tipuri referință:

- *Referința* este un “pointer” similar celor din C.
- Diferența este că referințele nu arată efectiv către o anumită zonă de memorie ci reprezintă un *identificator* al unui anume obiect.
- Referințe pentru: tablouri, obiecte.
- Rezultă implicit: nu cunoaștem adresa de memorie a variabilelor și a obiectelor în general.

Elemente de Bază ale Limbajului

Operatori:

- Majoritatea din C.
- ‘>>>’ și ‘<<<’ cu excepția că adaugă ‘0’ la stânga, respectiv la dreapta pentru a nu deplasa bitul de semn.
- ‘>>’ și ‘<<’ vor deplasa bitul de semn, la fel ca în C.



Elemente de Bază ale Limbajului

Operatori (cont.):

- instanceOf – returnează true/false dacă un obiect este sau nu o instanță a unei clase (utilizat la polimorfism).

- Conversii de tip: implicit sau explicit, la fel ca în C.