POO si Java

De la POO

Salt la: Navigare, căutare

Cuprins

[ascunde] [ascunde]

Programarea Orientată pe Obiecte

Programarea Orientata Obiect este o paradigma de programare care utilizeaza **obiecte** si **interactiuni** intre acestea pentru a modela arhitectura unui program.

Pana in anii '60, paradigma cea mai utilizata era cea a **programarii structurate**. Programarea structurata este tipul de programare pe care l-ati folosit pana acum, la cursul de Programare si la cel de SD. Aceasta paradigma consta in utilizarea **functiilor si procedurilor** pentru a realiza un program (cu eliminarea apelurilor GOTO).

Totusi, in anii '60, deja pe masura ce programale deveneau din ce in ce mai mari, randamentul programatorilor scadea si in consecinta TTM-ul (*time-to-market*) crestea. Stilul de programare structurata nu mai facea fata unor programe de dimensiuni mereu in crestere.

Fiecare paradigma de programare propune un nivel de impartire a taskului de realizat (adica a programului) in taskuri mai mici pentru a micsora complexitatea. Astfel, intr-un program **monoprocedural**, unitatea de abstractizare este instructiunea. In programarea **structurata** este functia/procedura. **Programarea Orientata Obiect** (POO) propune pentru acest lucru obiectul. Obiectele in POO modeleaza (sau ar trebui sa modeleze daca arhitectura aplicatiei este corecta) obiecte din lumea reala.

Obiectele din POO sunt **instante** ale unui tip de date numit **clasa**. Relatia dintre clasa si obiect-instanta a acelei clase este exemplificata de relatia intre conceptul de masa si masa din sufragerie. Adica:

- conceptul de masa implica existenta anumitor caracteristici (un numar de picioare si un blat, totul de o anumita culoare)
- conceptul de masa implica realizarea potentiala a unor actiuni (se poate manca pe masa)
- obiectul masa are caracteristicile respective (4 picioare, culoare neagra)
- obiectul masa permite realizarea practica a actiunilor respective (se poate manca doar pe masa fizica, nu pe ideea de masa)

Ce remarcam este ca exista niste tipare in lumea reala, care grupeaza in mintea noastra niste **atribute** ale obiectelor cu **actiunile** lor, pentru a forma un tot ce defineste obiectul respectiv. Pe acest concept, numit **incapsulare**, se sprijina programarea orientata obiect.

Folosirea POO permite realizarea de sisteme informatice de dimensiuni marite, cu timpi de dezvoltare, testare si mentenanta redusi fata de paradigmele anterioare. Totusi, pentru a crea un sistem functional este necesara intelegerea corecta a conceptelor care stau in spatele POO. Cu aceste concepte se ocupa cursul si laboratoarele de POO.

Platforma Java

Programarea Orientata pe Obiecte se poate aplica in orice limbaj care permite acest lucru. Cele mai cunoscute asemenea limbaje astazi sunt C++, Java, C#, chiar si PHP. In acest semestru vom ilustra conceptele de POO folosind limbajul Java.

Java a pornit ca o platforma de programare pentru sisteme embedded. Telurile principale ale proiectului erau:

- **independenta** de sistem
- utilizarea **POO**.

Astazi, Java este folosita mai mult ca o platforma pentru Internet si a atins o utilizare impresionanta.

Java este un mediu (platforma) de programare care consta in:

- un **limbaj** de programare (Java) care descrie programatorului ce instructiuni sunt valide si ce face fiecare
- un **compilator** (`javac.exe` (Windows) / `javac` (Linux)) care transforma fisierul sursa intr-un limbaj intermediar numit **bytecode**

- o **masina virtuala**, **Java Virtual Machine** (`JVM`), care permite transformarea codului intermediar in instructiuni executabile pe procesorul curent.
- o **biblioteca** puternica ce raspunde foarte bine nevoilor aparute in practica (*class library*)

Workflowul este urmatorul. Dezvoltatorul instaleaza **Java Development Kit** (`JDK`) care consta in principal din:

- **Java Runtime Environment** (`JRE`), ce contine `JVM`
- **compilator**.

Compilatorul este aplicat codului scris si se obtin fisiere continand bytecode. Aceste fisiere au in Java extensia `.class`.

Diagrama, pana acum, arata astfel:

`Clasamea.java` ----compilare---> `Clasamea.class` [pe masina de dezvoltare]

Acest pas corespunde cu invocarea compilatorului astfel:

```
javac Clasamea.java
```

Apoi codul bytecode este distribuit (nu analizam acum cum se face acest lucru) utilizatorului. El are instalat `JRE`, care este masina care interpreteaza bytecode-ul si il transforma intr-un flow de instructiuni pentru procesorul utilizatorului (exista un `JRE` pentru fiecare procesor si sistem de operare folosit).

Diagrama arata asa:

| flow de bytecode | ------> | JRE | ----> | flow instructiuni native |

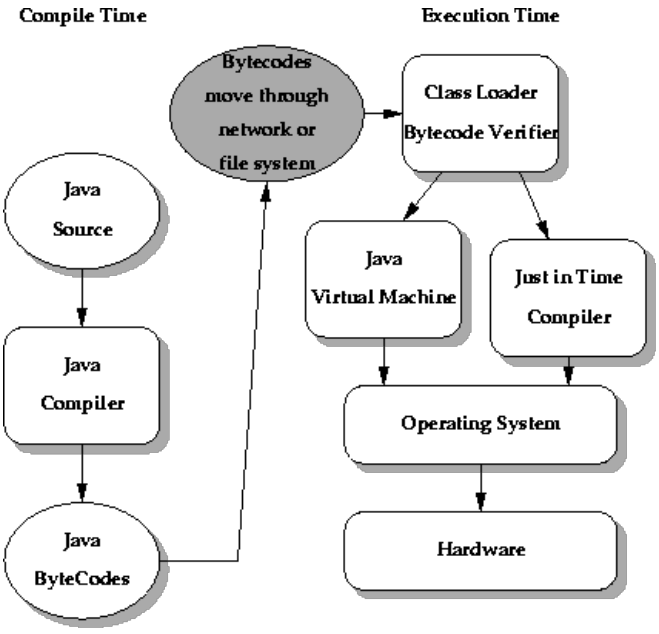
Pasul corespunde cu invocarea masinii virtuale astfel:

```
java Clasamea
```

Rezultatul instructiunilor native afecteaza flowul de instructiuni bytecode, astfel incat rolul `JRE` nu este doar o etapa de preprocesare. Nu se aplica o simpla transformare de instructiuni ca sa se obtina o imagine, dupa care sa se trimite imaginea de executabil nativ la procesor. Masina virtuala "interpreteaza" tot timpul. Codul bytecode este numit interpretat din aceasta cauza.

Cel mai important avantaj in acest workflow este ca permite obtinerea **independentei** de sistem. Dezvoltatorul are nevoie de un compilator functional pentru platforma pe care face dezvoltarea, iar utilizatorii, pe orice patforma ar fi (Sistem de Operare, Arhitectura hardware), pot utiliza programul cat timp au o masina virtuala Java instalata pentru acea platforma.

Un dezavantaj este viteza scazuta a codului Java. Exista overheadul implicat de actiunile aditionale realizate de `JRE` tot timpul rularii programului. Pentru a combate acest dezavantaj au aparut compilatoare *just-in-time* (`JIT`) care permit transformarea bytecodeului in cod executabil la prima rulare a unei secvente de instructiuni bytecode, apoi stocarea acestuia pentru refolosire. Aici masina virtuala nu este folosita decat o data. Modelul clasic este C#, care foloseste acest artificiu inca de la aparitia sa. (Intarzierea cauzata de prima pornire a aplicatiei .NET respective este vizibila in multe cazuri). Retineti ca modelul clasic Java este unul cu compilator si interpretor (masina virtuala).



Hello World

Pentru a incepe dezvoltarea avem nevoie de `JDK` pe care il gasim pe site-ul Sun. `JDK` contine si `JRE` pentru procesorul curent, asa ca putem testa aplicatiile pe masina locala.

Instructiuni de instalare

Pentru a seta variabilele de mediu in Linux adaugati urmatoarea linie in `/etc/bash.bashrc` (pentru a fi disponibil tuturor utilizatorilor):

```
export JAVA_HOME=/usr/lib/jvm/java-(versiunea ta)
```

unde valoarea variabilei `JAVA_HOME` este calea catre directorul unde aveti Java instalat.

Conform paradigmei POO, programul este compus din clase. Pentru a avea un entry point al programului (punct de inceput, cum este functia `main` in C), trebuie sa scriem o clasa:

```
class HelloWorld  {
    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

Observam ca:

- o clasa se defineste prin listarea metodelor, cuprinse intre acolade, dupa declararea ei prin cuvantul cheie class urmat de numele clasei.
- semnatura functei de entry in program este: `public static void main(String[] args)`
- intuim ca linia `System.out.println("Hello world")` va afisa mesajul de intampinare

Salvam textul ca `HelloWorld.java`. Compilam programul cu:

```
javac HelloWorld.java
```

Observam aparitia in directorul curent a unui fisier `HelloWorld.class` cu:

```
dir (ls pentru Linux)
```

Pentru rulare:

```
java HelloWorld
```

Instalare Eclipse IDE

Pentru Java exista mai multe medii de dezvoltare dintre care noi recomandam Eclipse datorita plugin-urilor disponibile. Eclipse ofera plugin-uri si pentru PHP, C/C++, Python etc.

Eclipse poate fi download-at de aici. Pentru instalare va recomandam acest tutorial.

Download

- JDK
- Eclipse

Adus de la "http://cursuri.cs.pub.ro/~poo/wiki/index.php/POO_si_Java"

- Schimbări recente
- Pagină aleatorie
- Ajutor

Caută

Du-te Caută

Trusa de unelte

- Ce se leagă aici
- Modificări corelate
- Trimite fişier
- Pagini speciale
- Versiune de tipărit
- Legătură permanentă
- Print as PDF