ANALIZĂ COMPARATIVĂ A PROGRAMĂRII ORIENTATE PE OBIECTE ÎN LIMBAJELE DE PROGRAMARE - ABAP ŞI JAVA

Mihaela Ivan

Academia de Studii Economice, Bucureşti

ivanmihaela88@gmail.com

Rezumat: Articolul are ca obiectiv formarea unei imagini generale prin prezentarea conceptelor modelului de programare orientat obiect a două limbaje de programare de uz industrial, ABAP şi Java. Această elaborare comparativă evidențiază asemănările şi deosebirile acestui model de programare cât şi elementele destinate să îmbunătățească eficiența programării. Sunt prezentate şi principalele metode de organizare a activității de programare.

Cuvinte cheie: programare orientată obiect, Java, ABAP, clasă, obiect.

Abstract: The objective of the article is to create a general idea by describing the concepts of the object-oriented programming through two languages of industrial use, ABAP and Java. This comparative elaboration highlights the resemblances and distinctions of this programming model and also the elements to enhance programming efficiency. Furthermore, it displays with detail the most essential methods of organizing programming activities.

Key words: Object-oriented programming, Java, ABAP, class, object.

1. Noțiuni generale ale programării orientate pe obiecte

Programarea orientată pe obiecte

- **Programul** [1] este o colecție de clase care modelează entitățile din problemă precum şi relațiile şi comportamentul acestora;
- **Clasele** sunt organizate în ierarhii, iar obiectele comunică prin mesaje;
- **Starea** unui obiect este definită de valorile atributelor sale;
- **Comportamentul** unui obiect este reprezentat de metodele sale.

Concepte fundamentale:

- **Abstractizarea** permite stăpânirea complexității. Desparte comportamentul de implementare;
- **Încapsularea** permite ascunderea implementării. Permite mai multe implementări pentru o interfață dată;
- **Modularitatea** permite divizarea unor proiecte complexe şi sprijină reutilizarea codului;
- **Ierarhizarea** permite exprimarea dependențelor şi reutilizarea codului.
- **Moştenirea** permite crearea unei clase noi prin extinderea unor clase existente. Poate fi simplă sau multiplă. Crează relații de tip subclasă-superclasă;
- **Polimorfismul** permite folosirea unui obiect de un tip în locul altuia de alt tip dar având aceiaşi interfață.

Noțiuni fundamentale

Clasa

- matriță pentru creat obiecte;
- conține membrii: atribute şi metode;
- membrii publici formează interfața clasei.

Obiect

Obiectele sunt de obicei reprezentări ale obiectelor din viața reală (domeniul problemei), astfel încât programele realizate prin această tehnică sunt mai uşor de înțeles, de depanat şi de extins decât programele procedurale. Aşadar, în cazul programării obiectuale, abstractizarea presupune definirea unor structuri de date complexe dotate cu un comportament exterior şi cu o stare internă, care se suprapun peste obiectele din lumea reală, figura 1.

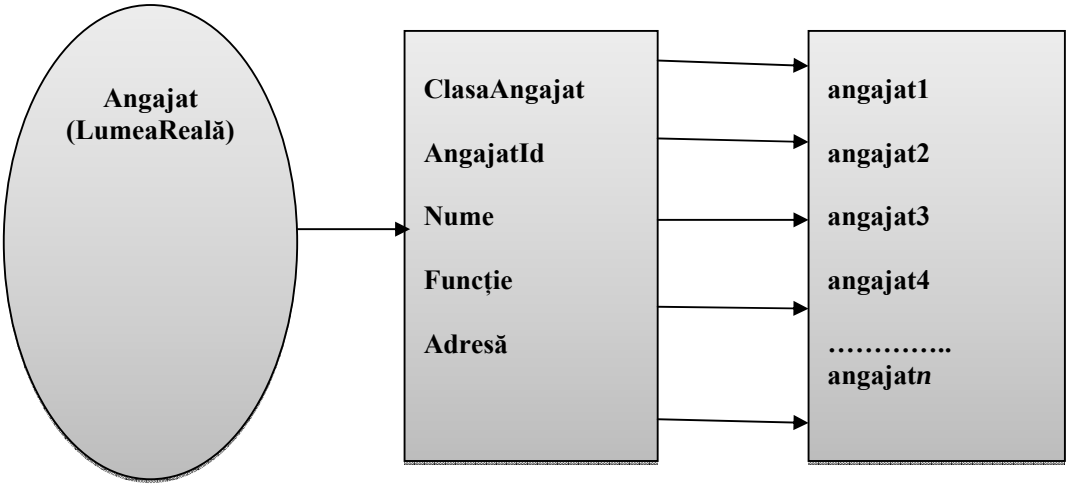

Figura 1. Modelul de abstractizare a lumii înconjurătoare

Ovalul reprezintă lumea înconjurătoare a unui angajat, iar în pătratul din mijloc clasa Angajat este o reprezentare din lumea reală prin atributele si metodele corespunzătoare acesteia. În ultimul pătrat angajat1, angajat2, angajat3, angajat4… angajat*n* sunt obiectele clasei Angajat.

Metoda obiectuală de dezvoltare conduce la o mai bună respectare a unor principii ale ingineriei software ca:

- localizarea şi modularizarea: codul sursă corespunzător unui obiect poate fi scris şi actualizat independent de alte obiecte;
- ascunderea informației: un obiect are o interfață publică pe care celelalte obiecte o pot utiliza în vederea comunicării. Dar, pe lângă interfața publică, un obiect poate include date şi operații private, "ascunse" față de alte obiecte care pot fi modificate oricând fără a afecta restul obiectelor;
- reutilizarea codului: multe clase de obiecte pot fi definite pe baza altor clase deja existente, preluânduse automat (prin moştenire) conținutul acestora din urmă.

2. Declararea claselor

Clasele reprezintă o modalitate de a introduce noi tipuri de date într-o aplicație, figura 2.

În Java conținutul clasei este format din:

- declararea şi inițializarea variabilelor de instanță şi de clasă;
- declararea şi implementarea constructorilor;
- declararea şi implementarea metodelor de instanță şi de clasă;
- declararea claselor imbricate (interne).

```
public class Cont {
        private int cantitate;
        public Cont(String id) {
        ...
        }
        public void depozit(int
        cantitate) {
        ...
        }
        ...
}
```

Java

```
CLASS cont DEFINITION.
PUBLIC SECTION.
METHODS:
        constructor IMPORTING id
TYPE string,
        depozit IMPORTING cantitate
TYPE i,
PRIVATE SECTION.
DATA:
      cantitate TYPE i.
ENDCLASS.

CLASS cont IMPLEMENTATION.
METHOD constructor.
...
ENDMETHOD.
  METHOD depozit.
...
ENDMETHOD.
...
ENDCLASS.
```

ABAP

Figura 2. Declararea claselor

În contrast cu limbajul de programare Java, în ABAP nu este permisă implementarea unei metode imediat după declararea acesteia.

Declararea şi implementarea [2], [5] metodelor unei clase sunt scrise separat în secțiunile de DEFINITION şi IMPLEMENTATION.

DEFINITION: secțiune utilizată pentru a declara componente ale claselor ca atribute, metode sau evenimente.

IMPLEMENTATION: secțiune utilizată pentru implementarea tuturor metodelor definite în secțiunea DEFINITION.

2.1 Atribute şi metode

Aşa cum este menționat mai sus o clasă conține atribute şi metode. Însă, o diferența majoră între cele două limbaje de programare este că în ABAP nu există conceptul de supraîncărcare a metodelor (overloading).

Cât despre Java, în cadrul unei clase pot exista metode cu acelaşi nume, cu condiția ca semnăturile lor să fie diferite (lista de argumente primite să difere fie prin numărul argumentelor, fie prin tipul lor) astfel încât la apelul funcției cu acel nume să se poată stabili în mod unic care dintre ele se execută.

În Java regulile de vizibilitate se aplică atât atributelor cât şi operațiilor din clase şi se referă la domeniul de acces permis la un membru al unei clase.

Există trei modificatori de acces :

- PUBLIC: accesibilitate la nivelul întregului sistem;
- PROTECTED: accesibilitate în arborele de moştenire;

- PRIVATE: accesibilitate numai din interiorul clasei.

În ABAP [3], [6] o clasă are trei secțiuni de vizibilitate: public, protected şi private; figura 3.

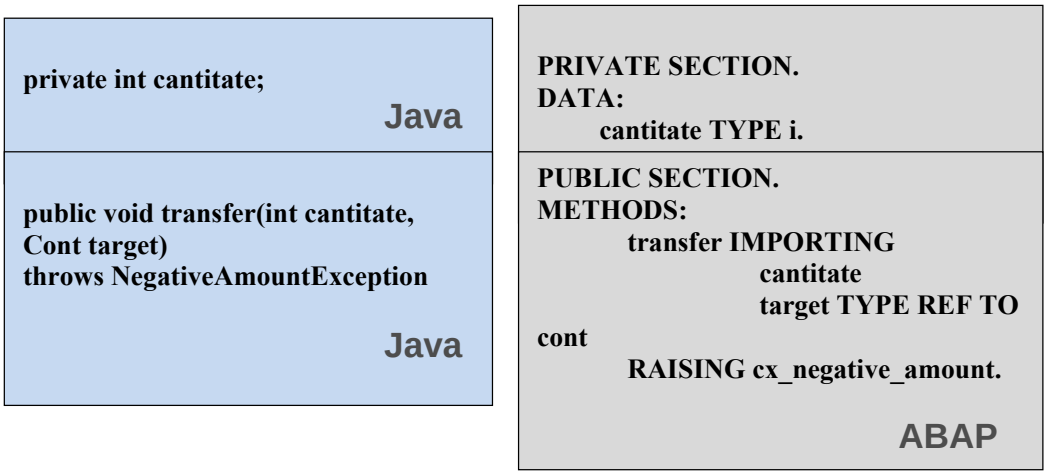

Figura 3. Atribute şi metode

- PUBLIC: Datele declarate în secțiunea publică pot fi accesate de clasă în sine, din subclase, precum şi din afara clasei;
- PROTECTED: Datele declarate în secțiunea protejată pot fi accesate de către clasă în sine, de subclasele ei, dar nu şi din afara clasei;
- PRIVATE: Datele declarate în secțiunea privată pot fi accesate doar de clasă în sine.

2.2 Accesul la atribute şi apelul comportamentelor

Pentru operațiunile de bază, sintaxa din ABAP are înlocuitor în sintaxa din Java după cum urmează, figura 4:

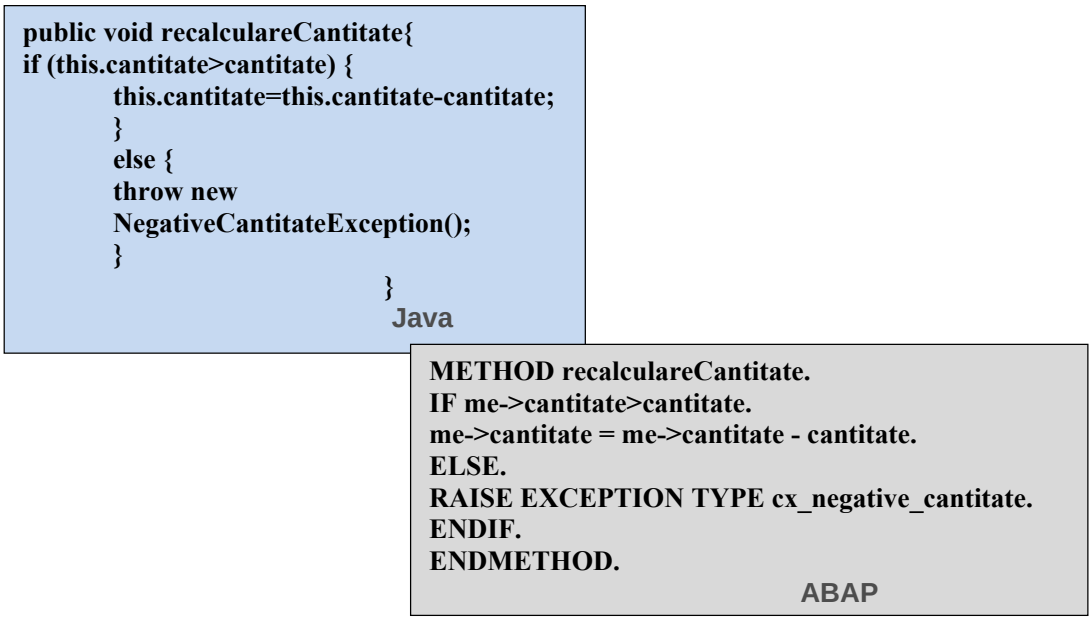

Figura 4. Accesul la atribute şi apelul comportamentelor

Aşadar similarităţile sintaxelor celor două limbaje de programare sunt:

- " -> "(Sau" - ") înlocuieşte " . "

  Prin " -> ", " - " şi " . " se face accesul la atribute şi metode.

- " . " înlocuieşte " ; "

  Prin " . " şi " ; " finalizăm o instrucţiune.

- "me" înlocuieşte "this"

  Prin "me" şi "this" putem accesa atributele şi metodele clasei curente.

Încă o diferenţă este aceea că în ABAP blocurile sunt construite cu ajutorul cuvintelor-cheie, în comparaţie cu acoladele {} din Java care de obicei încadrează blocul de cod.

2.3 Excepţii şi moştenire

În cele două limbaje de programare analizate, tratarea excepţiilor urmează principii similare [4], [9], [10], figura 5. O excepţie este un eveniment ce se produce în timpul execuţiei unui program şi care provoacă întreruperea cursului normal al execuţiei acestuia.

În momentul în care o eroare se produce în timpul execuţiei va fi generat un obiect de tip excepţie ce conţine:

- informaţii despre excepţia respectivă;
- starea programului în momentul producerii aceleaşi excepţii.

Crearea unui obiect de tip excepţie se numeşte aruncarea unei excepţii throwing an exception. În momentul în care o metodă generează (aruncă) o excepţie, sistemul de execuţie este responsabil cu găsirea unei secvenţe de cod dintr-o metodă care să o trateze.

La apariţia unei erori este ”aruncată” o excepţie, iar cineva trebuie să o ”prindă” pentru a o trata. Dacă sistemul nu găseşte nici un analizator pentru o anumită excepţie, atunci programul se opreşte cu un mesaj de eroare.

Adeseori poate apărea necesitatea creării unor excepţii proprii pentru a pune în evidenţăcazuri speciale de erori provocate de metodele claselor unei librării, cazuri care nu au fost prevăzute în ierarhia excepţiilor standard.

O excepţie proprie trebuie să se încadreze însă în ierarhia excepţiilor Java, cu alte cuvinte clasa care o implementează trebuie să fie subclasă a uneia deja existente în această ierarhie, preferabil una apropiată ca semnificaţie, sau superclasa Exception.

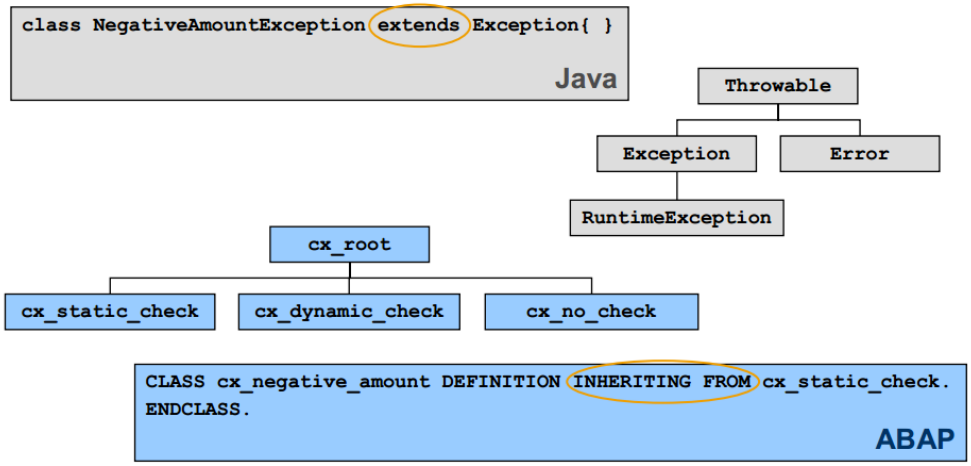

Figura 5. Excepţii şi moştenire

Gestionarea erorilor în Java se bazează pe o ierarhie de clase de excepție. Simpla alegere a clasei de excepție corespunzătoare reprezentării unei probleme furnizează informații despre condiția care a provocat-o.

Clasa Error şi subclasele sale sunt, de cele mai multe ori, utilizate pentru a indica probleme serioase de infrastructură, cum ar fi de exemplu un defect în însăşi maşina virtuală Java.

Clasa RunTimeException şi subclasele sale sunt denumite excepții neverificate, deoarece compilatorul nu cere captura acestora.

Subclasele clasei Exception care nu extind clasa RunTimeException sunt cunoscute sub numele de excepții verificate.

Gestionarea erorilor în ABAP se bazează pe o altă ierarhie de clase de excepție. Clasele de excepție: CX_STATIC_CHECK, CX_DYNAMIC_CHECK şi CX_NO_CHECK sunt subclase ale clasei globale generale CX_ROOT.

2.4 Arhitectura J2EE şi Arhitectura ABAP

Atât arhitectura J2EE cât şi arhitectura ABAP [7], [8] reprezintă o arhitectură multi-start. Scopul utilizării straturilor este posibilitatea de distribuire a componentelor software şi a serviciilor pe mai multe calculatoare, pentru scalabilitate şi securitate.

Arhitectura pe trei nivele este soluția care a adoptat ideea reunirii mai multor funcții de afaceri într-o singură aplicație rulând pe o singură bază de date.

Componentele principale ale acestor arhitecturi sunt serverul pentru baza de date, serverele de aplicații şi serverele de prezentare, figura 6 şi figura 7.

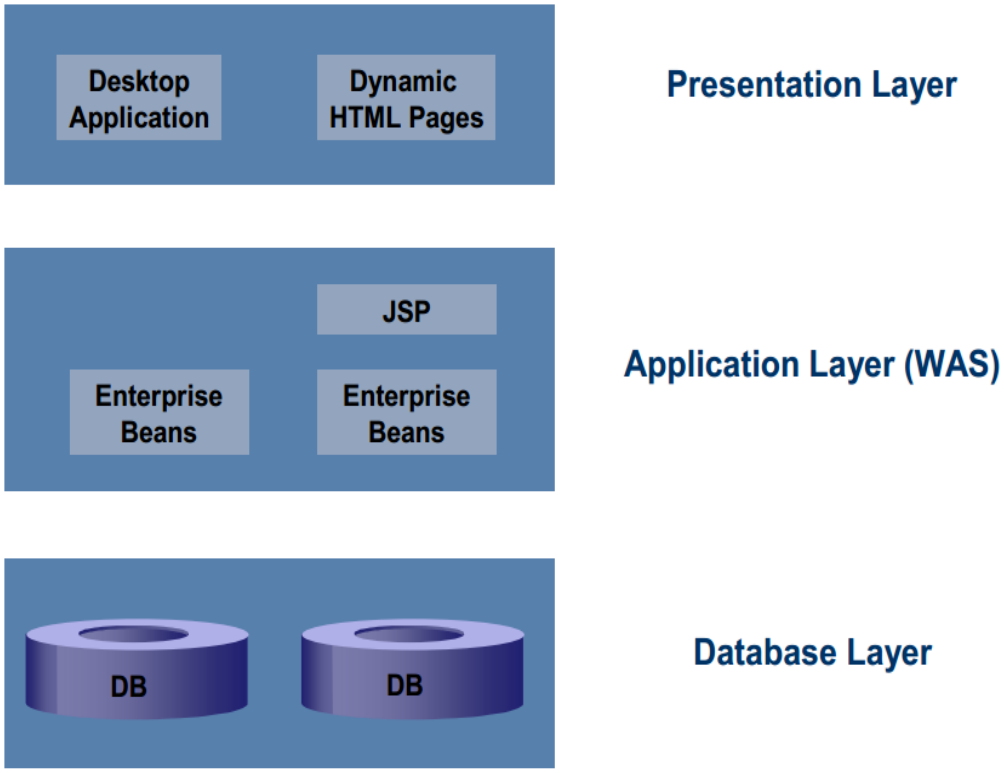

Figura 6. Arhitectura J2EE

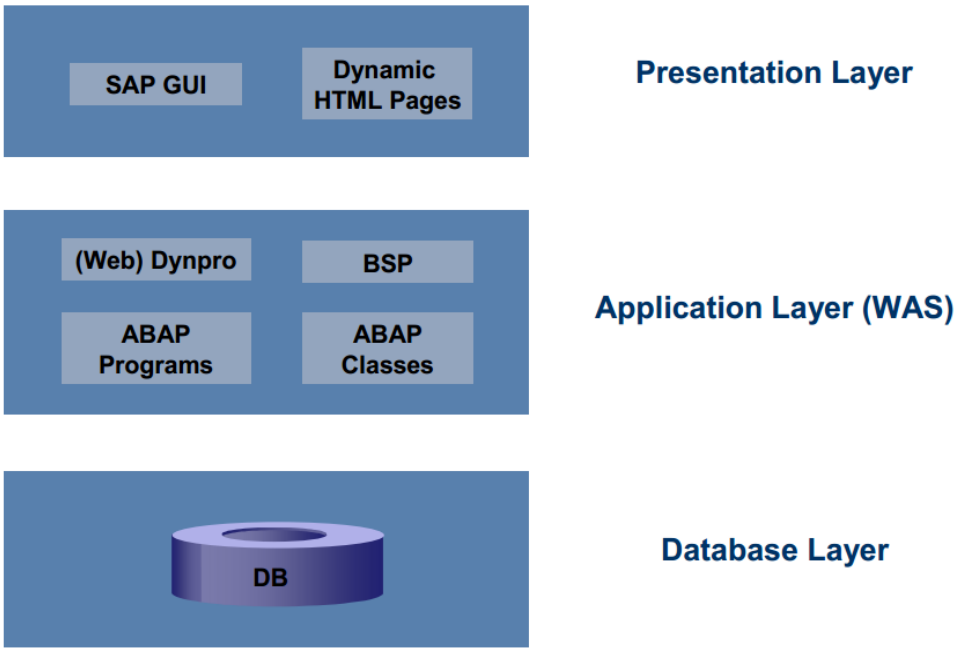

Figura 7. Arhitectura ABAP

3. Concluzii

Din mulțimea limbajelor care susțin remarcabil programarea orientată pe obiecte, în aceast articol am prezentat limbajele Java şi ABAP, ca limbaje suport pentru abordarea principiilor şi conceptelor a acestui model de programare.

Java este un limbaj de programare orientat pe obiect, puternic tipizat. Limbajul împrumută o mare parte din sintaxă de la C şi C++, dar are un model al obiectelor mai simplu.

Ca şi Java, ABAP este un limbaj de programare orientat pe obiect, care rulează pe o maşină virtuală şi este baza unei întregi tehnologii. Programarea orientată pe obiecte din ABAP este foarte similară cu JAVA prin prisma aspectelor prezentate în acest articol şi anume că ABAP Objects are la bază clase şi interfețe. La rândul lor clasele şi interfețele au atribute şi metode. Iar obiectele sunt create cu ajutorul claselor şi sunt accesate prin intermediul variabilelor de referință.

BIBLIOGRAFIE

1. **Lesson:** Object-Oriented Programming Concepts in Tutorial Java Sun.
2. **WOODS, D.; WORD, J.:** SAP NetWeaver for dummies.
3. **WOOD, J.:** Object-Oriented Programming with ABAP Objects.
4. Materiale de curs Telecom Academy http://www.telacad.ro
5. http://docs.oracle.com/javase/tutorial/java/concepts
6. http://www.webopedia.com/
7. http://pavelgk.pbworks.com/f/abap-object-oriented-programming-tutorials.pdf
8. http://help.sap.com/abapdocu_70/en
9. ABAP Objects and Application Areas; ABAP certification coursework.
10. BC - ABAP Programming; ABAP certification coursework.