# Java - un posibil succesor al Cobol-ului în informatica economică modernă?

Augustin MAN
Centrul de Informatică Aplicată Cluj

*Articolul prezintă mediul de dezvoltare multiplatformă **Java**, în contextul impactului pe care îl poate avea asupra realizării unor aplicații complexe destinate informaticii economice. Sunt prezentate caracteristicile care fac din informatica economică o categorie fundamentală a informaticii, se face o scurtă analiză care explică fostul succes de marcă al acestui domeniu, limbajul COBOL şi se prezintă evoluția informaticii de la momentul COBOL la cel actual. Este prezentat succint mediul **Java**, cu accentuarea calităților care îl deosebesc de alte medii de dezvoltare. Un paragraf este dedicat relației **JDBC**, interfața **Java** de acces la bazele de date relaționale. În încheiere se expun câteva concluzii.*
***Cuvinte cheie**: mediu de dezvoltare, limbaj de programare, prelucrare distribuită, rețea de calculatoare, client-server, sistem de operare, programare obiectuală, bază de date, driver, browser.*

## 1. Introducere

Informatica economică se constituie ca o parte deosebit de importantă în contextul informatic modern. Aspectele definitorii care fundamentează această prioritate sunt:
- frecvența utilizatorilor prin numărul imens de sisteme care rulează programe de informatică economică, de la contabilitatea activității micilor meseriaşi la gestiunile complexe ale unor bănci de importanță mondială;
- ordinul de mărime şi importanța valorilor gestionate, cu repercusiuni uriaşe în cazul unor eventuale erori de prelucrare; în acest context, singura paralelă posibilă fiind aceea a controlului proceselor nucleare sau chimice;
- caracteristicile de înaltă performanță impuse prelucrărilor, care tocmai datorită celor expuse mai sus, trebuie să fie:
  - *corecte*, nu numai în sensul elementar al corectitudinii calculelor propriu-zise, ci şi în privința acoperirii tuturor posibilităților de prelucrare şi/sau incident;
  - *rapide*, în contextul unor volume uriaşe de date şi al necesității frecvente a unui răspuns cu o întîrziere maximă de ordinul minutelor, iar în unele cazuri chiar al secundelor (de exemplu, la bursele de valori), viteza de prelucrare trebuie să se ridice la un nivel corespunzător;
  - *fiabile*, în primul rând în sensul recuperării cu pierderi minime în cazul unor incidente (prin algoritmul tranzacțiilor şi/sau prin redundanță) şi apoi prin persistența față de factorii perturbatori;
  - *sigure*, prin necesitatea protejării datelor în contextul creşterii presiunii asupra securității lor atât în ceea ce priveşte necesitatea transmiterii lor între diferite puncte, cât şi în privința măririi volumului de date confidențiale.

Se observă că există suficiente motive pentru ca tehnicile de prelucrare a datelor, atât hardware cât şi software, să acorde cea mai mare importanță domeniului informaticii economice, miza de piață fiind imensă, ca volum al vânzărilor şi pentru deschidere ulterioară în timp.
Evoluția categoriei hardware este mai complexă şi nu constituie subiectul materialului de față, care se va axa în cele ce urmează pe partea de software, cu accent asupra mediului de dezvoltare de programe. Importanța acestui element pentru sistemele de prelucrare din cadrul informaticii economice credem că nu mai trebuie justificată.
Strategia adoptată de diverse firme de software pentru cucerirea acestei piețe este în principal de două categorii, diametral opuse:
- *obstrucționistă*, de "legare" a utilizatorului de produsele firmei respective şi apoi taxarea lui pentru orice element de dezvol-

tare, fără de care acesta nu mai poate progresa. Exemplul cel mai concludent este acela al companiei Microsoft;

- *deschisă*, în sensul oferirii unor produse cu o largă compatibilitate, din care utilizatorul poate alege ceea ce i se pare mai potrivit cu problema pe care vrea să o rezolve şi cu dotarea deja existentă. Companii care s-au raliat acestei strategii sunt IBM şi Sun.

În contextul diversificării şi evoluţiei tot mai dinamice a platformelor hardware, corelate cu impunerea în tot mai mare măsură a reţelelor de transmisie a datelor şi a prelucrărilor distribuite, ideea de "închidere" software nu ni se pare prea potrivită. Poziţia este şi nu este nouă. Aparenta lipsă de logică a afirmaţiei este contrazisă de succesul fenomenal al mediului de dezvoltare multiplatformă **Java**. Ideea unui software portabil pe orice platformă nu este deloc nouă, ea datând încă din timpul main-frame-urilor, însă succesul punerii ei în practică este absolut nou - acum doi ani aproape nimeni nu auzise de Java.

## 2. COBOL - un fost remarcabil succes

Aplicaţiile economice realizate în perioada mainframe - mini (anii '70, '80) au fost concepute într-o proporţie covârşitoare (cca 90%) în limbajul de programare COBOL. El s-a impus în lumea economică pentru că a răspuns bine la cerinţele esenţiale ale programatorilor acelei perioade, printr-o serie de caracteristici de bază.

- *claritatea* limbajului, care l-a distanţat net de stilul "criptic" al majorităţii limbajelor de programare din acea vreme; COBOL-ul a fost criticat uneori tocmai sub acuzaţia că se iroseşte prea mult timp descriind în detaliu elementele programului).
- *structura clară* a programului, cu elemente bine determinate cuprinse în secţiuni, care se grupau la rândul lor în diviziuni; structura rigidă avea avantajul că programatorul nu putea uita afară o componentă esenţială a aplicaţiei.

COBOL-ul, deşi evident depăşit după standardele moderne ale unui mediu de dezvoltare de programe, a fost la timpul său un instrument puternic şi comod pentru realizarea aplicaţiilor informatice din domeniul economic.

Să vedem în continuare care au fost principalele linii de evoluţie informatică care l-au adus azi aproape în stadiul de exponat de muzeu: spunem aproape, fiindcă dorinţa de a economisi 2 bytes (!!) într-un timp când chiar unul singur în plus putea duce la irosirea unei pagini de memorie de 2KO, a generat azi "problema anului 2000", dezgropând surse demult uitate şi căutând în disperare programatori CO-BOL. Prin optica de azi, când o solicitare de +4MO pentru un anumit produs este ceva normal, acţiunea din trecut pare curată prostie, dar să nu uităm că atunci, într-un singur MO, rulau 3 utilizatori şi 4 taskuri de ex-ploatare pe mainframe şi 19-20 de utilizatori pe mini!

## 3. O evoluţie contradictorie

Dinamica informaticii este atât de puternică încât chiar şi menţionarea ei ne poate duce la căderea în truism: este cunoscută paralela cu industria de automobile, care dacă ar fi evoluat echivalent ar fi dus la automobile costând cîţiva cenţi şi de dimensiunea a cîţiva milimetri!

Principalul motiv al acestei schimbări stupefiante este ieftinirea puterii de calcul, care a dus imediat la o creştere şi o diseminare a acesteia, cu consecinţe imediate de diversificare hardware şi, ulterior, software.

Primul produs al acestei faze au fost calculatoarele tip PC sau echivalentele. Entuziasmul de a avea la îndemînă, numai pentru tine, un calculator (a cărui aură de excepţionalitate din timpul mainframe, când erai norocos dacă apucai măcar să vezi unul, nu se estompase complet în perioada minicalculatoarelor), a condus la câteva greşeli ale căror consecinţe le suportăm şi azi şi de care căutăm să scăpăm fără prea mari şanse de succes. Cea mai elocventă eroare de acest fel se numeşte DOS (sună cunoscut, nu?). De ce a fost DOS aproape un dezastru? Pentru că a fost gândit şi realizat cu faţa la trecut. Iată pe scurt ce s-a întîmplat: la originea DOS a fost CP/M, ideea de bază fiind crearea unui CP/M îmbunătăţit, astfel:

memoria s-a limitat la de 10 ori cea accesată de CP/M, adică 64KO*10 = 640KO ("640KO trebuie să fie suficienți pentru oricine" zicea plin de emfază patronul Microsoft, Bill Gates); hard-disk-ul putea să aibă maximum 1024 de cilindri şi cel mai grav lucru dintre toate, deşi chiar microprocesorul 286 avea protecție de memorie, sistemul rula fără validarea acesteia. Structura internă a DOS este la fel de improvizată şi cîrpită, dar acest subiect depăşeşte intenția materialului de față. Să mai menționăm doar consecințele soluțiilor de mai sus: datorită lipsei de protecție a taskurilor în memorie s-a putut dezvolta o întreagă pletoră de viruşi de toate felurile şi culorile. Memoria internă insuficientă a dus la improvizații de trei-patru alte categorii de memorii (expandată, extinsă, "superioară" etc.) cu consecințele respective de compatibilitate, etc. Limitarea la 1024 de cilindri a dus ulterior la soluția LBA (reducerea artificială a cilindrilor combinată cu creşterea corespunzătoare a numărului de capete, prin intermediul BIOS), partițiile create în acest fel nefiind recunoscute de către alte sisteme şi enumerarea ar putea continua pe încă multe pagini. Tarele sistemului DOS s-au transmis apoi cu succes spre bomba comercială Windows.

Prin contrast, un bătrîn venerabil al anilor '60, conceput însă corespunzător (UNIX), este bine merci şi, după unele păreri, chiar cel mai serios şi fiabil sistem de operare disponibil la momentul actual.

Foarte curând s-a observat că PC-ul izolat nu prea corespundea unor aplicații serioase şi astfel s-a pus problema comunicării între PC-uri sau, mai general, între feluritele unități de prelucrare. Rețelele de calculatoare de toate categoriile au proliferat şi s-au diversificat rapid.

Pe de altă parte, centralizarea de care toți se grăbeau să scape a revenit în forță prin prezența unor servere puternice, care permiteau rularea rapidă şi fiabilă a aplicațiilor: concomitent au revenit în actualitate prelucrarea tip client-server şi prelucrarea distribuită.

Creşterea volumului şi complexității datelor gestionate de calculatoare, lucru posibil prin ieftinirea suportului de date simultan cu creşterea capacității şi vitezei de acces, au dus la necesitatea unui software capabil să exploateze baze de date din ce în ce mai mari şi mai complexe.

Modul de realizare al software-ului a suferit evoluții contradictorii: în ceea ce priveşte analiza se manifestă, din păcate, o tendință involutivă, fiindcă dispariția constrîngerilor asupra disponibilității resurselor din perioada main/mini au condus de multe ori la un mod superficial de abordare a proiectelor, cu consecințe negative previzibile. Programarea s-a axat pe punerea la dispoziția utilizatorului a unor interfețe cât mai atrăgătoare şi mai intuitive şi pe un consum de resurse nerestrictiv (acum apare expresia "resource hogging" - însemnând textual "consum porcesc al resurselor").

În privința tehnicilor de programare, singurul fenomen notabil este cel al programării obiectuale. Concepută din dorința de a oferi creatorilor de aplicații complexe un instrument mai sigur de stăpânire a programelor decât programarea structurată, programarea obiectuală s-a dovedit nu numai un succes de natură tehnică, ci chiar o nouă filosofie de abordare a programării.

Sintetizând ideile de mai sus în dorința de a releva cerințele de bază ale unui sistem de dezvoltare pentru informatica economică în prezent şi viitor, putem spune că sistemul respectiv trebuie: să permită o portabilitate comodă între diferite platforme software/hardware; să includă posibilități de comunicare prin rețele de date; să permită elaborarea de aplicații distribuite; să fie structurat obiectual.

## 4. Mediul JAVA

În cadrul unui curs ținut în vara acestui an la Universitatea Politehnică din Brooklyn, New York (S1, "What is Java?" - /week1/07.html) Elliotte Rusty Harold defineşte Java ca un limbaj de programare de generația a treia, la fel cu C++ sau Perl. Totodată, pentru a-i sublinia caracterul mai complex decât acela al unui simplu limbaj, acelaşi autor afirmă că Java este o platformă (S1, "Java is a Platform" + /week1/08.html). Diferiți autori (R1, R3) dau definiții mai

mult sau mai puțin apropiate celei de mai sus, consensul general stabilindu-se în final la exprimarea mediul de dezvoltare multi-platformă Java.

În contextul nostru, definiția nu este esențială, mai importantă fiind prezentarea caracteristicilor care fac din Java un mediu potrivit pentru dezvoltarea aplicațiilor moderne de informatică economică.

Astfel, Java este un mediu independent de platformă, independență realizată prin utilizarea unei forme intermediare numită "coduri de octeți" (bytecodes). Codurile pot fi considerate instrucțiuni pentru un procesor virtual (S4), adică emulat prin software, numit mașină virtuală Java (Java VM). Java VM citește instrucțiunile Java și generează cod nativ corespunzător platformei de rulare. Întrucât acest proces este relativ lent s-au conceput ulterior îmbunătățiri, sub forma unor compilatoare specializate (Just-In-Time JIT) sau chiar cip-uri Java (R2). Toate componentele mediului, inclusiv auxiliare de felul debugger sau dezasamblor, se numesc kit de dezvoltare Java (Java Development Kit - JDK) sau pe scurt Java.

Așadar, etapele parcurse de către un program Java sunt următoarele: din sursă, compilatorul Java creează codurile de octet, identice pentru orice platformă care implementează Java. În continuare, interpretorul Java (java) transcrie "on-line" codurile de octet în limbajul nativ corespunzător platformei respective printr-una din metodele de mai sus.

În altă ordine de idei, mediul Java este strîns legat de procesul de comunicare de date, având înglobate în structură toate funcțiile necesare accesului Internet. Având în vedere răspândirea tot mai largă a procedeelor intranet (rețele locale create cu tehnologii tip Internet), această caracteristică permite utilizatorului rezolvarea imediată a tuturor problemelor de comunicare din cadrul unei aplicații.

În privința relației între Java și prelucrările distribuite, ea cuprinde două aspecte esențiale: primul, care se referă de fapt la prelucrarea client-server, este realizarea de programe specializate pentru a fi exploatate prin browser-e în cadrul Internet, numite applet-uri și care vor fi detaliate puțin mai departe; al doilea, introdus odată cu versiunea JDK 1.1., permite apelul metodelor din cadrul unor obiecte din rețea (Remote Method Invocation - RMI) sau trimiterea unei copii "imagine bytes" a unui obiect la un alt calculator din rețea, prin aceasta obținându-se o prelucrare distribuită în adevăratul sens al cuvîntului.

În sfîrșit, Java este un mediu pur obiectual, fundamental mai apropiat de Smalltalk decât de C++, cu a cărui sintaxă se aseamănă. Deși o prezentare a programării obiectuale nu constituie obiec-tul materialului de față, credem că nu este lipsi de interes să detaliem puțin lucrurile pentru cei care cunosc acest mod de programare. Astfel, toate clasele Java sunt derivate dintr-o singură clasă (Object). Există clase abstracte (de exemplu Com-ponent, Container), clase obișnuite (publice, de exemplu Window, Frame) și clase finale, care interzic subclasarea ulterioară (de exemplu String). În cadrul unui program pot exista mai multe clase, totuși numai una poate fi declarată de tip public. Față de C++, Java prezintă următoarele simplificări nota-bile:

- nu admite moștenirea multiplă;
- nu admite pointeri și referințe.

Din punct de vedere formal, clasele Java sunt grupate logic în pachete. În cadrul unui pachet există trei categorii de clase:

- clase obișnuite;
- excepții, care tratează cazurile de incident apărute ca urmare a unor situații deosebite (de exemplu clasa StringIndexOutOfBoundsException);
- erori, care tratează cazurile de eroare din cursul rulării (de exemplu clasa NoClassDefFoundError).

Excepțiile și erorile se concretizează prin instanțierea unor obiecte ale claselor, prin intermediul constructorilor respectivi; constructorii sunt singurele metode ale acestor clase, având două forme:

*public nume_clasa_exceptie_sau_eroare();*
*public nume_clasa_exc_sau_err(String s);*

A doua formă permite transmiterea unui mesaj utilizator. Întrucât clasele nu sunt finale, ele se pot subclasa opțional în cadrul

programului pentru elaborarea unor acţiuni mai complexe.

Pentru a sesiza dinamica Java, să amintim cu titlu informativ că varianta JDK 1.0.2. cuprindea 8 pachete cu 211 clase, pe când JDK 1.1.1. cuprinde 22 de pachete cu 477 de clase (în afara claselor de tratare excepţii şi erori). În afara completărilor de felul RMI, sau îmbunătăţirilor, cum este pachetul de acces la reţea, noua versiune reconsideră complet mecanismul de control al evenimentelor, o soluţie interesantă, care însă depăşeşte scopul materialului nostru.

Denumirile claselor şi ale interfeţelor au forma:

java.numep1.clasa

iar cele ale metodelor:

java.numep1.clasa.metoda();

Când nu sunt posibile confuzii, nu este necesară precizarea completă a denumirii respective.

Pachetele utilizate, cu excepţia pachetului de bază, java.lang, trebuie importate explicit, prin instrucţiunea import; evident se pot importa toate clasele unui pachet sau numai cele necesare. Totalitatea claselor şi metodelor Java destinate elaborării de aplicaţii formează interfaţa de programare a aplicaţiilor Java, cunoscută ca Java API (Application Programming Interface).

Programele Java sunt de două categorii: *aplicaţii* şi *applet*-uri. Aplicaţiile sunt programe obişnuite în sensul C++. Applet-urile sunt aplicaţii Java destinate exploatării prin Internet, astfel: applet-ul este rezident pe server şi la apelul său prin intermediul unei pagini HTML accesate de browser-ul utilizator, este trimis prin reţea la destinaţia respectivă şi activat. Pentru limitarea posibilităţilor de deteriorare a unor destinaţii, browser-ul limitează drastic accesul unui applet la resursele utilizatorului. De exemplu, Netscape nu permite nici citirea unui disc, chiar dacă sistemul destinatar acceptă aceasta. În ultima vreme există utilitare pentru execuţia unui applet (la distanţă sau local) în condiţii mai puţin restrictive decât cele ale unui browser.

Se pot concepe programe care se comportă fie ca applet, fie ca aplicaţie, după modul în care sunt lansate. Pentru a fi însă funcţionale în ambele situaţii ele trebuie să ţină seamă de restricţiile arătate.

În spaţiul Java au apărut şi alte categorii de programe asemănătoare applet-urilor: servlet-urile sunt applet-uri destinate în principal server-elor şi care nu au restricţiile de acces ale applet-urilor. Applet-urile sunt entităţi mobile circulând în reţea, în vederea strîngerii de informaţii utile unui anumit punct: ele pot fi considerate variante pozitive ale viruşilor de croazieră, de care tehnic nu se deosebesc prea mult.

Să încheiem succinta noastră prezentare a mediului Java cu câteva detalieri asupra interfeţei utilizator. Ea este cuprinsă în mai multe pachete cu denumirea java.awt.*, de la Abstract Window Toolkit - AWT. Fără a avea deocamdată sofisticarea de imagini a unui Borland C++ sau Microsoft Visual Basic, ea cuprinde toate elementele necesare pentru conceperea unei interfeţe utilizator GUI la standardele software-ului modern: există list-boxuri, butoane, ferestre, dialoguri modale etc. Este rafinat şi modul de control al acţiunilor utilizatorului (click-uri, introduceri de date etc.). Să amintim că pentru a-şi atrage cât mai rapid adepţi, Java a prezentat de la început mici programe vesele de animaţie bazate tocmai pe AWT. Deşi element indispensabil pentru proliferarea mediului Java, AWT nu este totuşi principala calitate a acestuia.

## 5. JDBC

JDBC este Java API pentru executarea instrucţiunilor SQL (Structured Query Language - un limbaj standard de acces pentru bazele de date relaţionale). Deşi este, ca şi Java, o marcă şi nu un acronim, JDBC este deseori considerat o prescurtare pentru Java DataBase Connectivity. Formal, JDBC este constituit din mai multe clase şi interfeţe scrise în Java şi care permit utilizatorilor elaborarea de aplicaţii de baze de date relaţionale utilizând exclusiv Java.

Principalul avantaj al utilizării JDBC este portabilitatea imediată a programului indiferent de structura bazei de date căreia i se adresează: va exista astfel un singur program pentru acces la o bază de date Informix sau

Sybase sau Oracle. Permițând aplicațiilor Java să se adreseze simultan unei multitudini de categorii de baze de date, JDBC are aplicabilitate imediată în nenumărate situații concrete. Astfel, utilizatori ai unor sisteme de operare diferite (Unix, OS/2 sau Windows) pot accesa prin Internet o bază de date pe o platformă RSX, se pot concepe pagini de Web cuprinzând date preluate din diferite baze de date din rețea etc.

Alte avantaje rezultate în urma utilizării JDBC-Java sunt:

- reorganizarea unor baze de date, cu schimbarea eventuală a software-ului suport, nu afectează programele aflate deja în exploatare;
- timpul de elaborare a aplicațiilor este scurt și depanarea ușoară;
- o aplicație nouă sau actualizată este încărcată pe server și au acces imediat toți utilizatorii;
- pentru departamentele de lucru cu beneficiarii, dinamica evidențelor nu mai reprezintă nici o problemă.

În principal, **JDBC** execută trei categorii de acțiuni:

- se conectează la o anumită bază de date;
- trimite instrucțiuni SQL;
- prelucrează rezultatul.

Iată un fragment de cod didactic, preluat din (S3), care ilustrează simplificat toate aceste acțiuni:

```
Connection con = DriverManager.getCon-nection
("jdbc:odbc:wombat", "login", "pas-sword");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a, b, c
FROM Table1");
while (rs.next()) {
    int x = getInt("a");
    String s = getString("b");
    float f = getFloat("c");
    }
```

JDBC se poate considera o API de nivel elementar, care poate fi folosită pentru elaborarea unor API-uri de nivel ridicat. La nivelul elementar la care a fost concepută, ea poate apela comenzile SQL direct. Performanțele sunt foarte bune și este, în general, mai ușor de utilizat față de alte API echivalente. Se pot concepe totuși API de nivel superior, mai sugestive ("user-friendly"), care să fie ulterior traduse în apeluri elementare JDBC. La momentul actual există două soluții pentru aceasta, prin extinderea JDBC:

1. SQL inclus în Java, astfel: DBMSs (DataBase Management System) implementează SQL. JDBC pretinde, conform exemplului de mai sus, ca instrucțiunile SQL să fie furnizate metodelor Java ca șiruri (String). Un preprocesor SQL permite introducerea comenzilor SQL direct în cadrul Java. De exemplu, o variabilă Java se poate folosi într-o comandă SQL pentru a furniza sau primi o valoare SQL. Preprocesorul traduce această îmbinare Java/SQL în Java cu apeluri JDBC.
2. Maparea directă a tabelelor bazei de date relaționale în clasele Java, soluție care va fi implementată de către JavaSoft și, probabil, de către alți producători de software. Prin maparea "obiectual/relațională", fiecare rând al tabelei devine o instanță a clasei, și valoarea din fiecare coloană un atribut al instanței respective. În acest mod, programatorul operează direct în cadrul unor obiecte Java, apelurile SQL de acces și stocare generându-se direct în background. Se propun și mapări mai evoluate, în care o clasă Java să cuprindă rânduri ale mai multor tabele.

O altă posibilitate care trebuie luată în considerare este ecranarea utilizatorului de sintaxa SQL, prin aplicații care să solicite date și să afișeze rezultate dintr-o bază de date generând comenzile SQL respective prin intermediul JDBC/Java. Folosind un asemenea program, un utilizator poate exploata sau actualiza o bază de date fără să cunoască ceva despre bazele de date.

O altă soluție promițătoare pentru facilitarea accesului Java la bazele de date relaționale este utilizarea legăturii JDBC-ODBC (JDBC ODBC Bridge). ODBC (Open DataBase Connectivity) este un API al companiei Microsoft, utilizat pe scară largă pentru programarea accesului la aceste baze. Se pot conecta aproape toate categoriile de baze de date relaționale de pe majoritatea platformelor în exploatare. Logic, se ridică întrebarea de ce să nu se simplifice soluția apelându-se direct ODBC din Java? De fapt, acest lucru este posibil, dar utilizarea JDBC este susținută de mai mulți factori:

1. *ODBC utilizează interfața C.* Apelul codului C din Java are serioase dezavantaje privind securitatea aplicației, implementarea, coerența și portabilitatea ulterioară.
2. *Pentru că ODBC utilizează structuri interzise de către Java* (de exemplu void*), apelul direct sau transcrierea ODBC în Java nu se poate face; JDBC are tocmai rolul de a traduce apelurile ODBC în apeluri compatibile Java.
3. *ODBC este complex și dificil de învățat*, având opțiuni multiple pentru cele mai simple cereri. JDBC este structurat mai bine, apelând la caracteristici avansate numai când este necesar.
4. *Utilizarea ODBC presupune instalări manuale de drivere proprii platformelor utilizator.* JDBC, fiind o interfață Java, asigură automat instalarea, portabilitatea și securitatea aplicațiilor în rețea.

În fond, JDBC este concepută pe baza aceloraşi caracteristici ca și ODBC (X/Open SQL Call Level Interface (CLI)), cu deosebirea că respectă stilul simplu și clar al Java. JDBC se poate considera o interfață naturală Java pentru conceptele și abstracțiunile SQL. Evoluțiile ulterioare ale ODBC (RDO, ADO, OLE DB etc.) converg spre JDBC în ceea ce privește aspectul obiectual, dar se consideră că nu are rost să se insiste în această direcție, care ar complica și mai mult lucrurile, fără a aduce nimic pozitiv în relația cu Java.

JDBC suportă modele de acces cu două sau trei niveluri. În modelul cu două niveluri aplicația sau applet-ul Java se adresează direct bazei de date. În acest caz este necesar un driver JDBC capabil să comunice cu DBMS-ul bazei respective. Comenzile SQL sunt transmise bazei, iar rezultatele sunt returnate utilizatorului. Baza de date poate fi localizată fizic într-un alt nod al rețelei Internet sau intranet.

Modelul cu trei niveluri interpune între baza de date și utilizator un nivel intermediar de servicii, care controlează cererile adresate de către utilizator bazei și rezultatele furnizate de către aceasta. Modelul are o serie de avantaje:

- permite un control global al actualizărilor și aplicațiilor;
- se pot utiliza API-uri de nivel înalt, traduse de către nivelul intermediar de servicii în API-uri de nivel scăzut;
- prin configurări avantajoase se pot obține performanțe superioare.

Nivelul intermediar de servicii, elaborat până acum în C sau C++, se poate elabora în Java, cu toate avantajele deja arătate. În acest caz, JDBC permite accesul bazei de date din cadrul nivelului intermediar Java.

O problemă importantă a JDBC este că, deși majoritatea DBMS suportă o formă standard SQL pentru implementarea funcțiunilor de bază, ele nu recunosc forme mai noi sau mai avansate ale sintaxei SQL sau le interpretează diferit. Pînă când SQL va deveni un standard propriu-zis, JDBC trebuie să facă față situației actuale. Pentru aceasta există trei posibilități.

- Trimiterea oricărui șir de cerere driverului DBMS, cu riscul de a primi în loc de răspuns un mesaj de eroare, dacă DBMS-ul nu "înțelege" cererea res-pectivă; de fapt, alinierea la SQL nici nu este necesară.
- Prevederea unei secvențe escape, pentru tratarea variantelor particulare de SQL.
- Pentru aplicațiile complexe, trimiterea spre aplicație a descrierii DBMS-ului apelat prin intermediul unei structuri numite interfață Database-MetaData; în continuare, aplicația se poate adapta particularităților DBMS-ului respectiv.

Întrucât JDBC API se va utiliza ca bază pentru API-uri și utilitare de acces de nivel superior, JDBC al oricărei platforme trebuie să răspundă unui standard, numit "JDBC COMPLIANTTM" echivalentă unui SQL tip ANSI SQL-2 Entry Level. Deși JavaSoft nu impune acest standard, este de dorit ca toate implemen-tările JDBC să ofere utilizatorilor o structură minimală, pe care aceștie să se poată baza.

Produsele JDBC care fac parte din JDK sunt: JDBC driver manager; JDBC driver test suite; JDBC-ODBC bridge.

Primul produs este de fapt coloana vertebrală a JDBC și are ca unic rol conectarea aplicației Java cu drive-rul potrivit JDBC.

Al doilea produs este destinat testării faptului că driverele JDBC pot rula aplicația

Java. Ele asigură caracterul minimal de JDBC COMPLIANTTM.
Al treilea produs permite utilizarea driverelor ODBC ca drivere JDBC.
Credem că prin cele expuse am demonstrat că JDBC este o alternativă promiţătoare pentru accesul bazelor de date relaţionale din cadrul mediului Java.

## 5. Concluzii

Rapiditatea cu care s-a impus mediul Java în informatica modernă este numai aparent surprinzătoare. Am văzut că dezideratele cărora Java le răspunde cu succes datează încă din timpul mainframe-urilor. Dacă un mediu de acest fel a apărut abia acum, întîrzierea este datorată probabil lipsei unui context software-hardware suficient de matur şi complex pînă la momentul actual. Informatica economică este o parte prea importantă a informaticii pentru a ignora acest mediu sau să întîrzie în cunoşterea şi folosirea lui.
De aceea, am încercat să facem o pledoarie tocmai în acest sens, cu accent special asupra relaţiei dintre mediul Java şi accesul la bazele de date relaţionale. Java poate fi un succesor al COBOL-ului pentru informatica economică modernă. În plus, Java oferă o perspectivă de realizare a aplicaţiilor mult superioară COBOL-ului cu facilităţi de neimaginat pentru acea perioadă. Este suficient să reamintim două: portabilitatea pe orice platformă nu numai existentă, dar chiar şi viitoare şi accesul Internet şi intranet încorporat.
Sperăm că demersul nostru a fost convingător şi va contribui la dezvoltarea şi proliferarea aplicaţiilor Java într-o economie care are mare nevoie de ele.

## Bibliografie

**Reviste**
R1. Dr.Dobb's Journal (USA) - 1997 (http://www.ddj.com)
R2. BYTE (USA) - 1997 (http://byte.com)
R3. PC World (UK) - 1997 (http://pcw.co.uk)
**Site-uri Internet**
S1. http://sunsite.unc.edu/javafaq/course/
S2. ttp://java.sun.com/products/jdk/1.1/docs/index.html
S3.http://java.sun.com/products/jdk/1.1/docs/guide/jdbc/index.html
S4. http://www8.zdnet.com/pcmag/features/pctech/1518/java02.htm
S5. http://java.sun.com/products/jdbc