C1

- Istoric Java.
- Avantaje.
- Rularea aplicaţiilor Java şi JVM.
- JDK, pachete şi împachetări Java.
- Concepte de bază: cuvinte cheie, variabile, convenţii de nume, definiţia de metodă, blocul, instrucţiuni.
- Compilare şi rulare.

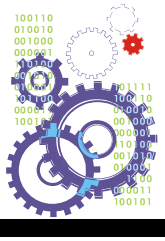
Obiective

Dupa parcurgerea acestui curs ar trebuie sa puteti:

- ☐ descrie caracteristicile si avatanjele limbajului Java;
- ☐ descrie masina virtuala Java (JVM);
- ☐ identifica componentele de baza ale kitului de dezvoltare Java (JDK)
- ☐ identifica componentele cheie ale limbajului
- ☐ instala, compila si rula o aplicatie Java simpla

Ce este Java?

- **Proiectat de Sun pentru programarea aparaturii electronice**
- **Limbaj orientat pe obiect cu biblioteci de clase**
- **Folosește o mașină virtuală pentru rularea programelor**



Proiectat de Sun

Java este un limbaj de programare orientat pe obiect dezvoltat la Sun Microsystems, Inc. A fost creat de James Gosling pentru a fi utilizat la programarea aparaturii electronice. Ca urmare a robusteții și a independenței de platformă, Java a trecut de bariera industriei "aparaturii electronice" la aplicațiile de Web, apoi la aplicaţii locale şi client-server

Biblioteci de clase

Java conține o mulțime de clase predefinite și metode care pot trata majoritatea cerințelor fundamentale ale unei aplicații. Kitul Dezvoltatorului Java (JDK - Java Developer's Kit) include clase pentru gestionarea de ferestre, intrare/ieșire și comunicație în rețea.

Java mai conține un număr de utilitare care ajută la dezvoltarea aplicațiilor. Aceste utilitare tratează operații cum sînt: depanarea, descărcarea și instalarea, documentarea.

Folosește Mașina Virtuală Java

Unul dintre punctele forte ale lui Java este independența de platformă (mașină + sistem de operare). O aplicație Java scrisă pe o platformă poate fi dusă și rulată pe orice altă platformă. Facilitatea este deseori referită ca "write once, run anywhere (WORA)". Ea este dată de folosirea Mașinii Virtuale Java - Java Virtual Machine (JVM). Aceasta rulează pe o mașină locală și interpretează codul de biți (byte code) convertindu-l într-un cod mașină specific platformei.

Echipa lui Gosling a pornit de la C++, dar programele nu erau sigure, parte a complexității limbajului, dar și datorită unor facilități distructive cum sunt poantorii. Pentru evitarea problemelor echipa lui Gosling a inventat un limbaj nou orientat pe obiect numit Oak (în memoria unui stejar impunător vizibil din biroul lui Gosling). Pentru creșterea robusteții au eliminat construcțiile de limbaj problematice, iar pentru a face aplicațiile neutre arhitectural, au specificat complet semantica limbajului și au creat o mașină virtuală pentru rularea programelor. Deși Oak a fost proiectat să semene cu C++ pentru a reduce timpul de învățare, el a fost reproiectat intern pentru a elimina pericolele și elementele redundante ale lui C++. Oak nu a reușit să se impună în domeniul pentru care a fost creat, dar, după apariția WWW-ului și-a găsit un nou drum pe care să evolueze. Oak a fost redenumit în Java (exista deja un limbaj cu numele Oak). Robust, compact, independent de platformă și flexibil a devenit ideal pentru crearea de aplicații Web. Fiind interpretat este mai lent ca C++, dar scăderea vitezei este neimportantă atunci cînd domeniul de utilizare este cel de interacțiune cu utilizatorul.

Avantaje Java

- **Orientat pe obiect**
- **Interpretat și independent de platformă**
- **Dinamic și distribuit**
- **Cu fire de execuție multiple**
- **Robust și sigur**

Orientat pe obiect
Obiectul este o entitate caracterizată prin atribute şi un set de funcţii folosite la manipularea obiectului. Java este puternic tipizat, aproape totul fiind, la nivel de limbaj, un obiect. Principalele excepţii sînt tipurile primitive (întregii şi caracterele).

Interpretat şi independent de platformă
Programele Java sînt interpretate în setul de instrucţiuni ale maşinii native în timpul rulării. Deoarece Java se rulează sub controlul JVM, programele Java pot rula sub orice sistem de operare care dispune de JVM.

Dinamic şi distribuit
Clasele Java pot fi descărcate dinamic prin reţea atunci cînd este cazul. În plus, Java asigură un suport extins pentru programarea client-server şi distribuită.

Cu fire multiple de execuţie (Multithreaded)
Programele Java pot conţine mai multe fire în vederea execuţie concurente a mai multor sarcini. Multithreadingul este o facilitate internă lui Java şi a aflată sub controlul JVM care este dependent de platformă.

Robust şi sigur
Java are facilităţi interne de prevenire a coruperii memoriei. Java gestionează automat procesul de alocare a memoriei şi verificarea limitelor tablourilor. Interzice aritmetica poantorilor şi restricţionează obiectele la zone impuse de memorie.


Termenul de multithreading (fire de execuţie multiple) este specific calculului paralel şi apare în contextul mediilor de execuţie care pot intercala execuţia unor instrucţiuni din mai multe surse independente de execuţie. Diferenţa faţă de multitasking constă într-o partajare mai profundă a mediului de execuţie la nivel de thread-uri în comparaţie cu multitasking-ul. Thread-urile pot fi identificate numai prin registrul numărător de programe (PC) şi registrul pointer la stivă (SP) atunci cînd partajează un singur spaţiu de adrese şi o singură mulţime de variabile globale. Acesta este motivul pentru care comutarea între-thread-uri se face foarte repede deoarece informaţia de stare de salvat şi de refăcut este scurtă.

Independența de platformă

- **Codul Java este stocat în fișiere `.java`**
- **Programul `.java` este compilat în fișiere `.class`**
- **Codul de biți este interpretat la rulare**

Problema limbajelor compilate
La compliarea unei aplicații într-un limbaj tradițional, cum ar fi C, codul scris de programator este convertit în limbajul mașină al platformei pe care are loc compilarea. Programul compilat poate fi rulat numai pe mașini care au același procesor (Intel, SPARC, Alpha) cu cea pe care s-a făcut compilarea.

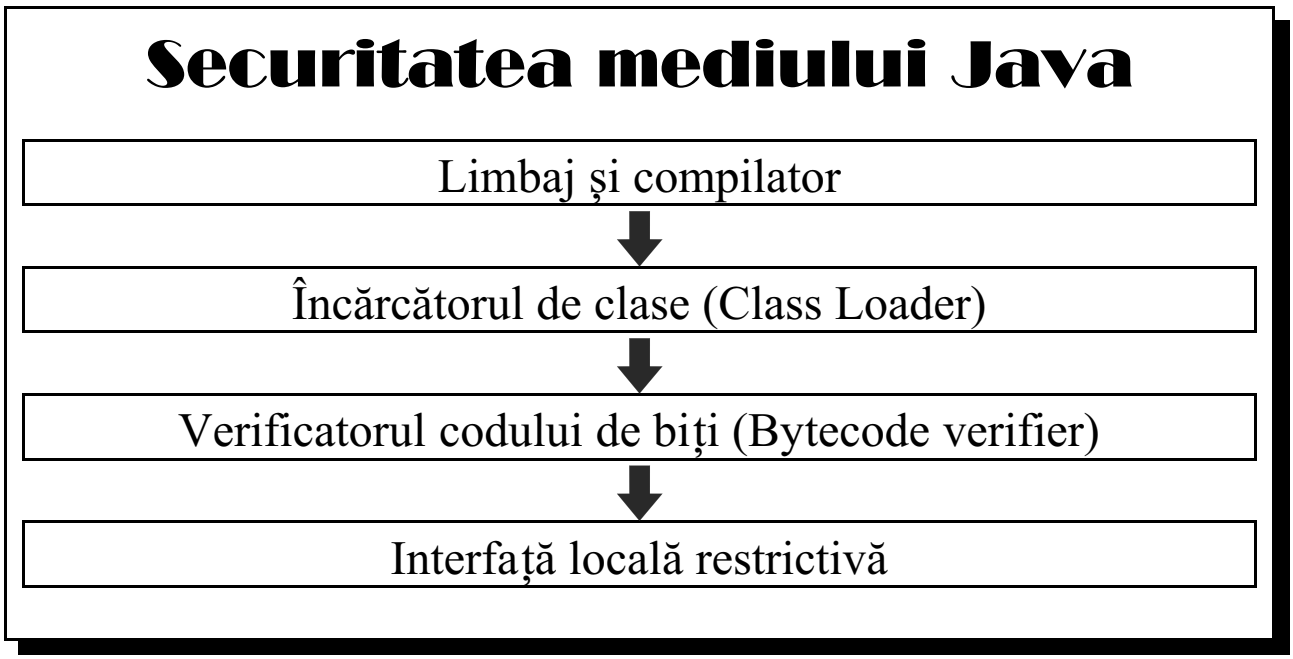
Java este un limbaj independent de platformă
Programele sursă Java sînt stocate în fișiere cu extensia `.java`. De exemplu, o aplicație Java care gestionează studenții unei secții ar putea conține fișerele `studenti.java`, `materii.java` și `note.java`. Fiecare fișier `.java` este compilat într-un fișier `.class` cu același nume. De exemplu, `studenti.java` se va compila în `studenti.class`. Fișierele `.class` conțin cod executabil Java numit "cod de biți" (bytecode) care are instrucțiuni mașină independente de platformă.

Mașina Virtuală Java (JVM - Java Virtual Machine)
Mașina Virtuală Java asigură mediul de execuție pentru programele Java. JVM interpretează codul de biți în instrucțiuni mașină native ale mașinii pe care programul este în curs de rulare.

Aceleași fișiere .class pot fi rulate fără modificare pe orice platformă pentru care există o JVM. Din acest motiv JVM este uneori numită și procesor virtual.


Se numeşte compilator un program care converteşte un alt program, dintr-un limbaj numit sursă în limbaj maşină (unicul limbaj înţeles de direct de către "calculator"). După realizarea completă a conversiei, programul în limbajul maşină va rula foarte rapid pe calculator. Procesul de compilare se realizează în mai multe faze: analiza lexicală (colectează caracterele programului sursă sub forma atomilor lexicali), analiza sintactică (preia atomii lexicali şi construieşte o reprezentare a sintactică a programului în memorie), analiza semantică (verifică existenţa erorilor de tip), generare de cod (aici se va obţine prima formă a limbajului maşină).

Deşi limbajul maşină generat de compilator ar putea fi rulat direct pe calculator deseori el trebuie rulat împreună cu alte programe dependente de sistemul de operare pe care urmează să fie rulată aplicaţia finală. De exemplu, în cazul operaţiilor de intrare/ieşire, compilatorul va genera apeluri catre funcţiile de intrare/ieşire ale sistemului de operare (în Java acesul la sistemul de opeare se face pe baza unor biblioteci numite “pachete standard”). Pentru ca aplicaţie să funcţioneze în contextul respectivului sistem de operare, compilatorul va trebui să lege respectivele porţiuni ale sistemului de operare la programul tradus. Operaţia de legare conectează programele scrise de utilizatori la programele sistemului de operare prin plasarea adreselor punctelor de intrare în sistemul de opeare în programul scris de utilizator. Se numşte interpretor un program care rulează alte programe. Interpretorul este un simulator al unei unităţi centrale care extrage şi execută instrucţiuni ale unui limbaj de nivel înalt (şi nu limbaj maşină). Din punctul de vedere al conversiei în limbajul maşină, Java a implemetat un compromis între compilatoarele şi interpretoarele pure. Aici, limbajul sursă este compilat la o formă intermediară (între limbajul Java şi limbajul maşină) foarte uşor de interpretat (deoarece a fost deja decodificat o dată) numită “cod de biţi” cu ajutorul lui javac apoi, acest cod intermediar este rulat pe interpretorul codului de biţi, java (se zice că acest limbaj maşină este pentru o maşină virtuală - Maşina Virtuala Java - care este implementată nu la nivel hardware ci la nivelui interpretorului codului de biţi). Deci, în Java programul sură trece prin analiza lexicală, sintactică generarea de cod intermediar, apoi interpretare pentru a obţine relutate.



Nivele de securitate în Java

Limbaj şi compilator

Limbajul este sigur prin proiectare. Construcțiile care permit manipularea directă a poantorilor au fost eliminate pentru evitarea erorilor în execuție și a fragmentării memoriei.

Încărcătorul de clase

Încărcătorul de clase garantează separarea stocării claselor ce provin de la surse locale de cele de rețea. În timpul execuției, motorul mașinii caută mai întîi referințele la clasele locale şi numai apoi clasele referite din interiorul acestora. Din acest motiv, clasele locale nu vor fi suprascrise de cele încărcate din rețea. Astfel, se previne modificarea comportamentului deja cunoscut al claselor locale ca urmare a unei intervenții externe (prin rețea).

Verificatorul codului de biți

În timpul rulării unui program Java, JVM poate importa cod de oriunde. Java trebuie să fie sigur de codul importat respectiv că el este dintr-o sursă de încredere. Pentru realizarea acestei sarcini, motorul va realiza o serie de teste denumite "verificarea codului de biți".

Interfaţă locală restrictivă

Accesul la sistemul de fișiere local şi la resursele de rețea este controlat prin clase locale şi metode. Clasele sunt prin definiție restrictive. Dacă un cod importat încearcă să acceseze sistemul de fişiere local, sistemul de securitate lansează un dialog cu utilizatorul.

Miniaplicații Java (Applets)

- **Cele mai vechi tipuri de aplicații**
- **Folosite în pagini HTML**
- **Pot include conținut activ (formulare, imagini, sunet, filme)**
- **Vizualizate în navigator și pot comunica cu un server**

Miniaplicații Java

O mare parte din popularitatea lui Java pornește de la posibilitatea dezvoltării unor aplicații mici (applets, în engleză) care pot fi înglobate în pagini HTML și descărcate prin rețea atunci cînd pagina este vizualizată într-un navigator de Web.

Există două tipuri de miniaplicații Java: cu și fară încredere (trusted și untrusted, în engleză). Miniaplicațiile "fără încredere" sînt restricționate din punctul de vedere al accesului local, cele "de încredere" au permisiunea accesării resuselor locale.

Caracteristicile miniaplicațiilor

Miniaplicațiile sînt de natură grafică, tinzînd să conțină controale cum sînt butoanele, cutiile de text, listele de selecție. Totuși, miniaplicațiile pot să facă mult mai mult decît simpla afișare a unor controale (obiecte de interfață grafice). Iată o listă cu cîteva dintre posibilitățile miniaplicațiilor:

- vizualizarea de animații, imagini și redare de sunete;
- conectarea la baza de date aflată pe serverul de Web de pe care s-a descărcat miniaplicația;
- comunicația cu un server de Web prin socluri (soket, în engleză);
- comunicația cu o aplicația Java rulată pe sereverul de Web;
- pot utiliza CORBA (Common Object Request Broket Architecture) pentru a comunica cu Java sau cu aplicații ne-Java de pe server-ul de Web.

OMG (Object Management Group) este un consorțiu creat în scopul dezvoltării de standare în programarea orientată pe obiect. CORBA specifică condițiile în care un obiect respectă specificațiile OMG, în principiu, specificațiile descriu standardul de interfață pentru aceste obiecte.

Aplicații Java

- **Crearea de aplicații de sine stătătoare:**
  - ☐ **JVM rulează aplicații independente**
  - ☐ **nu se încarcă clase prin rețea**
- **Crearea de aplicații client-server:**
  - ☐ **deservește mai mulți clienți dintr-o singură sursă**
  - ☐ **se încadrează în modelul multi-nivel pentru calcule pe Internet.**

Aplicații Java de sine stătătoare

Deși, Java și-a câștigat popularitatea datorită miniaplicațiilor, e posibilă și crearea unor aplicații de sine stătătoare (nu este nevoie de un navigator pentru rularea lor). Acestea sunt oarecum echivalentul programelor executabile din C şi C++.

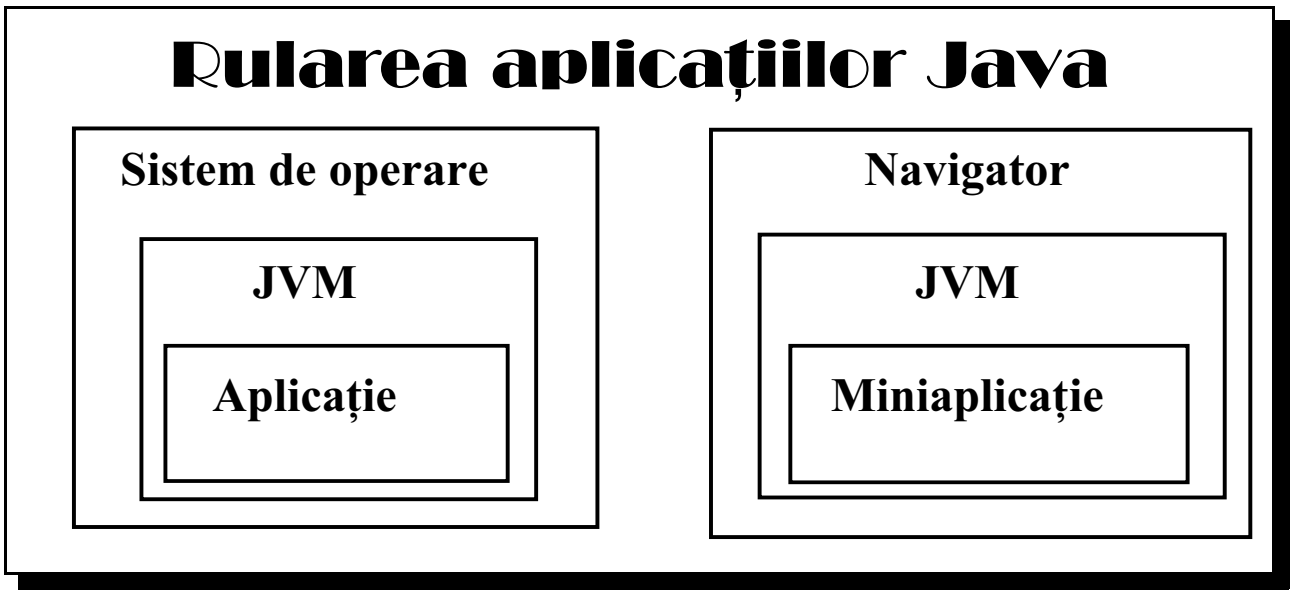
Aplicații client

Miniaplicațiile Java pot fi rulate pe un sistem de operare local sub forma unor aplicații de sine stătătoare. Acest mod de lucru este specific dezvoltării de aplicații client (aplicaţia este descarcătă de pe un server dar rulată în navigator, care este clientul serverului de Web) şi este, uzual, fară restricțiile de securitate specifice miniaplicațiilor. De exemplu, aplicațiile Java pot accesa sistemul local de fişiere sau să stabilească conexiuni cu alte maşini prin rețea. Miniaplicațiile pot beneficia de aceste facilități doar dacă au fost înregistrare că sînt de încredere; restricția previne coruperea sistemului local de fişiere de către miniaplicații malițioase. Caracteristici adiționale ale miniaplicațiilor de încredere sînt:

- accesul la sistemul de fişiere local;
- accesul la o maşină distinctă de server-ul de pe care s-a descărca miniaplicaţia;
- lansarea în execuție a unor programe noi sau încărcarea de biblioteci locale.

Aplicații sever

Aplicațiile Java pot fi, de asemenea, rulate pe o maşină server dacă JVM este disponibilă pe platforma respectivă. Rularea pe server asigură dezvoltarea de aplicații după modelul celor 3 nivel specifice calculului pe Internet. Aplicaţia server rulează în mod continuu pe server şi trimite un răspuns unui client ca urmare a unei cerei efectuate de acesta.



Rularea aplicațiilor Java
Toate aplicațiile Java, inclusiv miniaplicațiile, se rulează la nivelul JVM. JVV este pornită diferit, după cu programul este o aplicație Java sau o miniaplicație.

Rularea aplicațiilor
Aplicațiile se rulează prin unei JVM locale direct din sistemul de operare. JVM interpretează programul Java şi îl converteşte în instrucțiuni mașină specifice platformei. Rularea programului se face plecând de la metoda statică numită `main()` pentru o aplicație de sine stătătoare sau printr-o referință la o clasă de miniaplicație dintr-un fişier HTML atunci cînd un program este încărcat din navigator.

Rularea miniaplicațiilor
O miniaplicație Java este un tip special de program folost în paginile Web. Atunci cînd navigatorul de Web citește o pagină HTML cu marcajul de miniaplicație (<applet>) ve descărca miniaplicația prin rețea pe sistemul local şi va porni miniaplicația în JVM care vine inclusă în navigator.

Funcționarea JVM

- **Încărcătorul de clase al JVM încarcă clasele necesare fucționării aplicației**
- **Verificatorul JVM verifică secvențele ilegale de cod de biți**
- **Gestionarul de memorie al JVM eliberează memoria după terminarea aplicației**

Încărcătorul de clase al JVM
Rularea unui fișier cu extensia `.class` poate necesita și alte clase pentru îndeplinirea scopului propus. Aceste clase sînt încărcate automat de încărcătorul (class loader) de clase în JVM. Clasele pot să fie stocate pe discul local sau pe un alt sistem, caz în care accesul se face prin rețea. JVM folosește variabila de sistem `CLASSPATH` pentru a determina poziția fișierelor locale cu extensia `.class`.

Clasele încărcate prin rețea sunt păstrate într-un spațiu de nume separate de cel al claselor locale. Astfel se previn conflictele de nume și înlocuirea sau suprascrierea unor clase standard prin clase malițioasă.

Verificatorul JVM
Sarcina verificatorului este să determine dacă codul de biți interpretat nu violează regulile de bază ale limbajului Java și că acesta este dintr-o sursă de încredere. Validarea asigură inexistența violării accesului la memorie sau execuția unor acțiuni ilegale.

Gestionarea memoriei
JVM urmărește toate instanțele (variabilele obiect) utilizate. Dacă o instanță nu mai este folosită JVM are obligația să elibereze memoria folosită de obiect. Eliberarea memoriei se face după ce obiectul nu mai este necesar nu neapărat imediat după aceea. Procesul prin care JVM gestionează obiectele care nu mai sînt referite se numește colectare de gunoi (garbage collection).

Compilatoare JIT (Just-in-Time)

- **Cresc performanțele**
- **Utile dacă același cod de biți se execută repetat**
- **Optimizează codul repetitiv (ciclurile)**

Compilatoare JIT

JVM traduce codul de biți Java în instrucțiuni native al mașinii pe care se rulează. Ce se petrece dacă același cod trebuie executate din nou ceva mai tîrziu în program? În lipsa unui compilator JIT codul interpretat de la fiecare reluare a lui, chiar dacă a fost deja interpretata o dată ceva mai devreme deja.

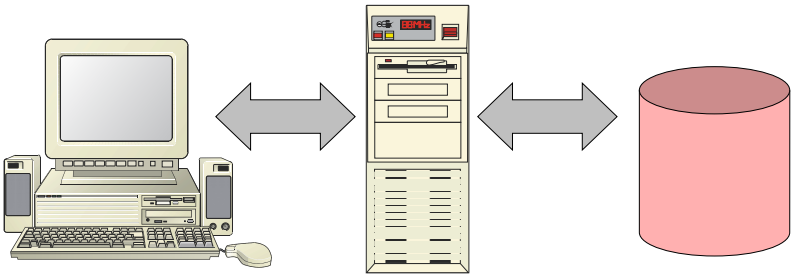
Avantajele compilatoarelor JIT

Majoritatea JVM suportă deja compilare JIT. Compilatoarele JIT traduc codul de biți numai la prima lui apariție; dacă același cod va fi executat mai tîrziu, va fi automat asociat codului corespondent în limbaj mașină.

Compilatoatele JIT fac ca Java să funcționeze mai repede în momentele în care se pune problema traducerii în mod repetat a unui cod de biți în instrucțiuni mașină native. Efectul este spectaculos în cazul ciclurilor sau a funcțiilor recursive. Unele compilatoare JIT sînt suficient de inteligenta ca să optimizeze traducerea unor secvențe de cod de biți în instrucțiuni native ale mașinii pe care se rulează aplicația.

Java şi calculul pe Internet

- **Calculul pe Internet are loc pe 3 nivele:**



Client **Server de aplicatii** **Server de date**

- **Java poate fi utilizat pentru oricare dintre aceste nivele**

Java şi calculul pe Internet

Produsele firmelor serioase pentru calcule pe Internet trebuie să suporte construcţia de componente Java pentru trei nivele de aplicaţie, precum şi instrumete pentru esenţiale pentru dezvoltare şi gestiune a alicaţiilor. Suportul se implementează sub forma de instrumente, aplicaţii, baze de date, servere de aplicaţii şi interfeţe pentru progrmarea aplicaţiilor.

Nivelul client

Atunci cînd Java trebuie să fie rulat pe o maşină client este tipic implementat într-o miniaplicaţie executata din navigator care poate comunica cu server-ul de pe care afost descărcat.

Nivelul serverului de aplicaţii

Java poate fi utilizate pe server-ul de aplicaţii pentru implementarea unor componente partajate şi reutilizabile ce ţin de logica aplicaţiei. Java mai poate fi utilizat la acest nivel de miloc pentru a furniza interfeţede control pentru aplicaţii generate sub forma unor pagini Web.

Nivelul serverului de date

Acest server stochează date, dar mai poate stoca şi executa cod Java, în particular acest cod manipulează intensiv date sau forţează reguli de validare specifice datelor.

Termenul de client-server apare în contextul sistemelor distribuite. Aplicaţia distribuiă este o colecţie de aplicaţii a căror distribuţie este transparentă la nivelul utilizatorului astfel încît la aplicaţia pare să fie stocată pe o singură maşină locală. Într-o reţea de calculatoare utilizatorii simt că lucrează pe o maşină particulară, iar poziţia lor în reţea, stocarea datelor încărcarea şi funcţionalitatea maşinii lor nu este transparentă. Sistemele distribite folosesc, uzual, o anumită formă de organizare cliet-server.

Clientul este o aplicaţie sau un proces care solicită un serviciu de la un alt calculator sau proces, denumit “server”. Server-ul este o aplicaţie care furnizează un anumit serviciu altor programe, numite “client”. Conexiunea între client şi server se face la nivel de mesaje transmise prin reţea şi foloseşte un protocol pentru codificareea cererilor clientului şi al rapsunsurilor server-ului. Server-ul poate să ruleze continuu în aşteptarea cererilor sau

să fie pornit de o aplicație care controlează un grup de server-e. Acest model permite plasarea clenților și a serevr-elor independent, în nodurile rețelei, posibil pe hardware-uri și pe sisteme de operare diferite pentru a-și îndeplini funcțiile (server rapid/client lent).

Ce este JDK?

Mediul de dezvoltare JDK Sun conține:

- **Compilator**
- **Componentă de vizualizare a miniaplicațiilor**
- **Interpretor al codului de biți**
- **Generator de documentație**

Componentele Sun JDK sunt:

- **compilatorul** java `javac`, el compilează codul sursă Java în codul de biți (bytecode) Java;
- `appletviewer` este componenta pentru **vizualizarea miniaplicațiilor**, aceasta citește un document HTML, descarcă miniaplicațiile referite în document și le afișează într-o fereastră proprie. Se folosește ca o alternativă la navigatorul de Web pentru testarea miniaplicațiilor Java;
- **interpretorul** codului de biți Java, `java`, care este motorul ce rulează aplicațiile Java;
- **generatorul de documentație** în format HTML pe baza codului sursă Java este `javadoc`.

Alte scule JDK

- **depanatorul de clase** Java este `jdb` (asemănător cu depanatoarele dbx sau gdb din UNIX);
- `javakey` pentru generarea cheilor de **certificare** și a codului Java de încredere (trusted);
- `jar` pentru generarea de arhive de calse și de resurse;
- **dezasamblorul** codului de biți Java într-un format inteligibil pentru oameni, `javap`;
- aplicația de **versionare** a claselor, `serialver`.

Pachete Java

- **echivalentul bibliotecilor C**
- **pachetele standard sunt pentru lucrul cu:**
  - ✓ **Limbajul**
  - ✓ **Ferestre**
  - ✓ **Miniaplicaţii**
  - ✓ **Intrare/Ieşire**
  - ✓ **Comunicaţie în reţea**

Pachete Java

Pachetele asigură bazele pentru funcționarea lui Java și sunt implementate sub forma unor serii de clase cu metode grupate după funcționalitate. Pachetele Java sunt echivalentul bibliotecilor din C. De exemplu, există un grup de clase care ajută la crearea și utilizarea conexiunilor de rețea, acestea sunt toate conținute în pachetul `java.net`. Pachetul de bază al limbajului Java este `classes.zip`.

Pachete Java standard

Aceste pachete conțin clasele fundamentale pentru toate aplicațiile și miniaplicațiile Java, cîteva mai importante sunt:

Pachet Java	Rol	Clase incluse
`java.lang`	Colecţia claselor de bază din Java. Conţine rădăcina ierarhiei de clase Java, `Object`, definiţiile tuturor tipurilor primitive etc.	`Object, String, Thread, Runtime System`
`javax.swing`		`Window, Button, Menu, Graphics`
`java.applet`	Clase pentru suportul scrierii miniaplicaţiilor.	`Applet, AudioClip`
`java.io`	Colecţia de clase de intrare/ieşire (inclusiv pentru accesul la fişiere)	`File, InputStream, OutputStream`
`java.net`		`Socket, DatagramPacket, URL, InetAddress`

s

Împachetări, platforme şi versiuni Java

Împachetări:

- **J2SE - Java 2 Platform, Standard Edition**
- **J2EE - Java 2 Platform, Enteprise Edition**
- **J2ME - Java 2 Micro Edition**

Platforme (SO + UCP):

- **Solaris, Windows ... + SPARC, Intel ...**

Versiuni:

- **Java release 1.0, 1.1; Java release 2, versiunea 1.2, 1.3, 1.4, 1.5, 1.6**

Împachetări Java

Tehnologia Java bazată pe JVM este strucurată pe trei tipuri de produse proiectate în funcţie de cerinţele particulare ale pieţei de software:

- J2SE - Java 2 Platform, Standard Edition: se foloseşte pentru "applet" şi aplicaţii care rulează pentru un singur calculator;
- J2EE - Java 2 Platform, Enteprise Edition: pentru aplicaţii client/server distribuite;
- J2ME - Java 2 Micro Edition: pentru crearea de aplicaţii care rulează pe un dispozitiv consumator (PDA, telefon celular etc.).

Fiecare pachet de tehnologii Java are o ediţie SDK (Software Development Kit) prin care se pot crea, compila, executa programele în tehnologie Java pentru o platformă particulară.

Platforme Java

Familia de produse a tehnologiilor Java este strâns legată de J2SE SDK deoarece majoritatea programatorilor îşi încep cariera pe PC-uri prin scrierea de applet-uri. Sun a dezvoltat Java 2 Platform, Standard Edition SDK pentru următoarele platforme:

Sistem de operare	Procesor
Solaris OE	chip SPARC pe 32 biţi
Microsoft Windows	chip Intel pe 32 de biţi
Solaris OE	chip SPARC pe 64 de biţi

Sistem de operare	Procesor
Linux	Intel

Versiuni Java

Prima versiune a lui Java a fost 1.0. Acesta a apărut pe piaţă sub denumirea de JDK 1.0, în anul 1996 şi conţinea maşina virtuală Java, bibliotecile de clase şi instrumentele specifice pentru dezvoltare (compilator etc.) aplicaţiilor. Evoluţia versiunilor de limbaj este prezentată în tabelul următor:

Versiune	An
JDK 1.0	1996
JDK 1.1	1997
JDK 1.2 (cunoscut şi sub denumirea de Java 2)	1998
JDK 1.3	2000
JDK 1.4	2002
JDK 1.5 (cunoscut şi sub denumirea de Java 5)	2004
JDK 1.6 (Java 6)	2006

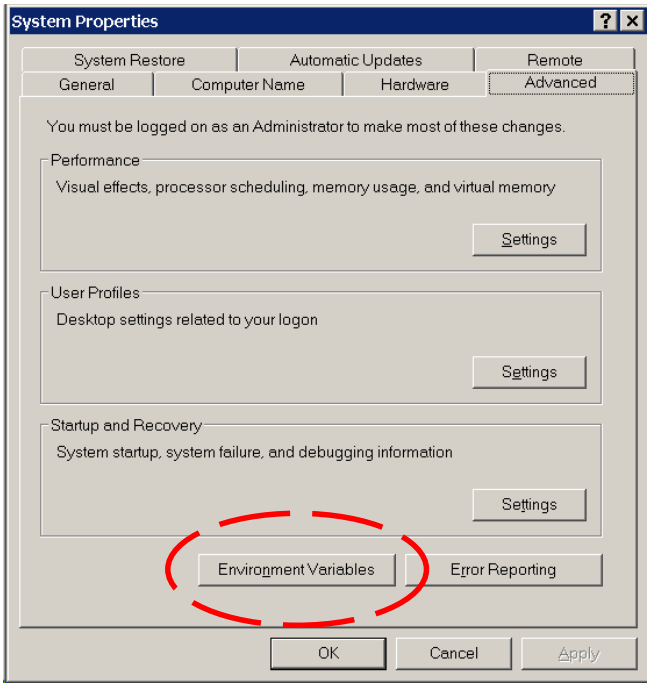
Începând cu JDK 1.2 toate versiunile au fost numite, pe scurt, Java 2. În timpul evoluţiei lui Java, modificările aduse limbajului iniţial au fost puţine, schimbări majore au avut însă loc la nivelul bibliotecilor acestuia şi al integrării limbajului în contextul unor tehnologii software strâns legate de Java. Numerotarea de 2 (din Java 2) arată că limbajul a progresat, intrând oficial în perioda lui "modernă", însă din dorinţa de a păstra continuitatea numerotării versiunilor de bibliotecă s-a acceptat că întregul limbaj să fie versionat pe baza acestora. Dincolo de modificările şi adăugirile de limbaj, în Java 2, apare pentru prima oară conectarea lui la un mediu de dezvoltare al aplicaţiilor. În afară de JDK, mai nou, Sun furnizează, şi separat componentele (JVM şi bibliotecile) necesare rulării de aplicaţii Java (fără posibilitatea dezvoltării de aplicaţii) sub denumirea de JRE (Java Runtime Environement).

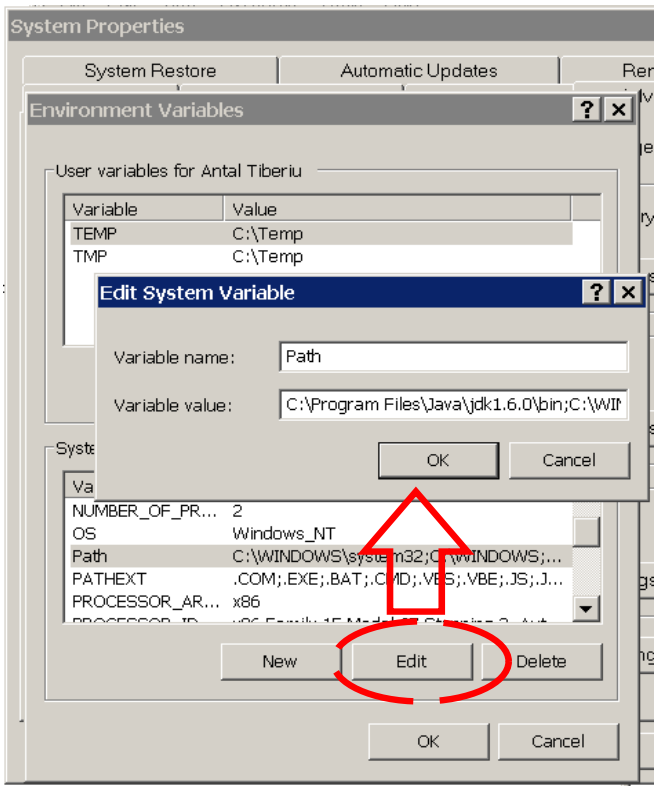
Instalrea şi configurarea JDK J2SE

1. **Descărcare J2SE de la http://java.sun.com/j2se:** `jdk-6-windows-i586.exe`
2. **Instalare JDK în C:\Program Files\Java\jdk1.6.0**
3. **Despachetare surse (**`src.zip`**) şi documentaţie (**`jdk-6-doc.zip` **de la** http://java.sun.com/docs **)**
4. **Configurare SO: Control Panel > System > Advanced > Environment Variables ➡ PATH**
5. **Verificarea configurării: Start > Run > cmd cu path şi java- version**

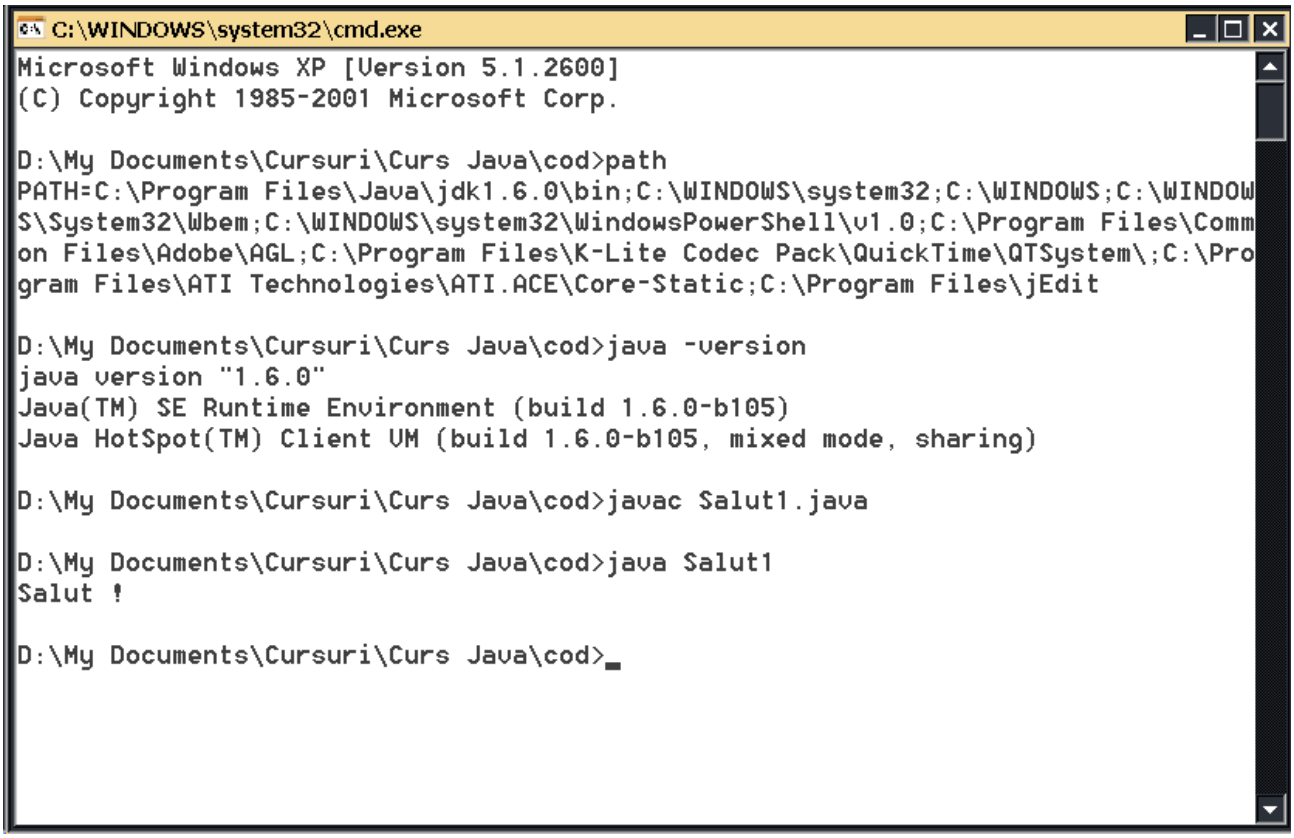
Instalarea JDK

Ultima versiune de JDK se descarcă de pe site-ul Sun-ului (http://java.sun.com/j2se), pentru Windows, împachetarea J2SE fiind sub forma fişierului executabil **jdk-6-windows-i586.exe**. Acesta conţine şi JVM-ul care se va instala automat împreună cu JDK-ul. După instalare calea jdk/bin trebuie adăugată în lista fişierelor executabile pentru ca sistemul de opearare să găsească automat pe javac, java etc. Pentru aceasta, în Windows XP, mergem la *Control Panel > System > Advanced > Environment Variables*.



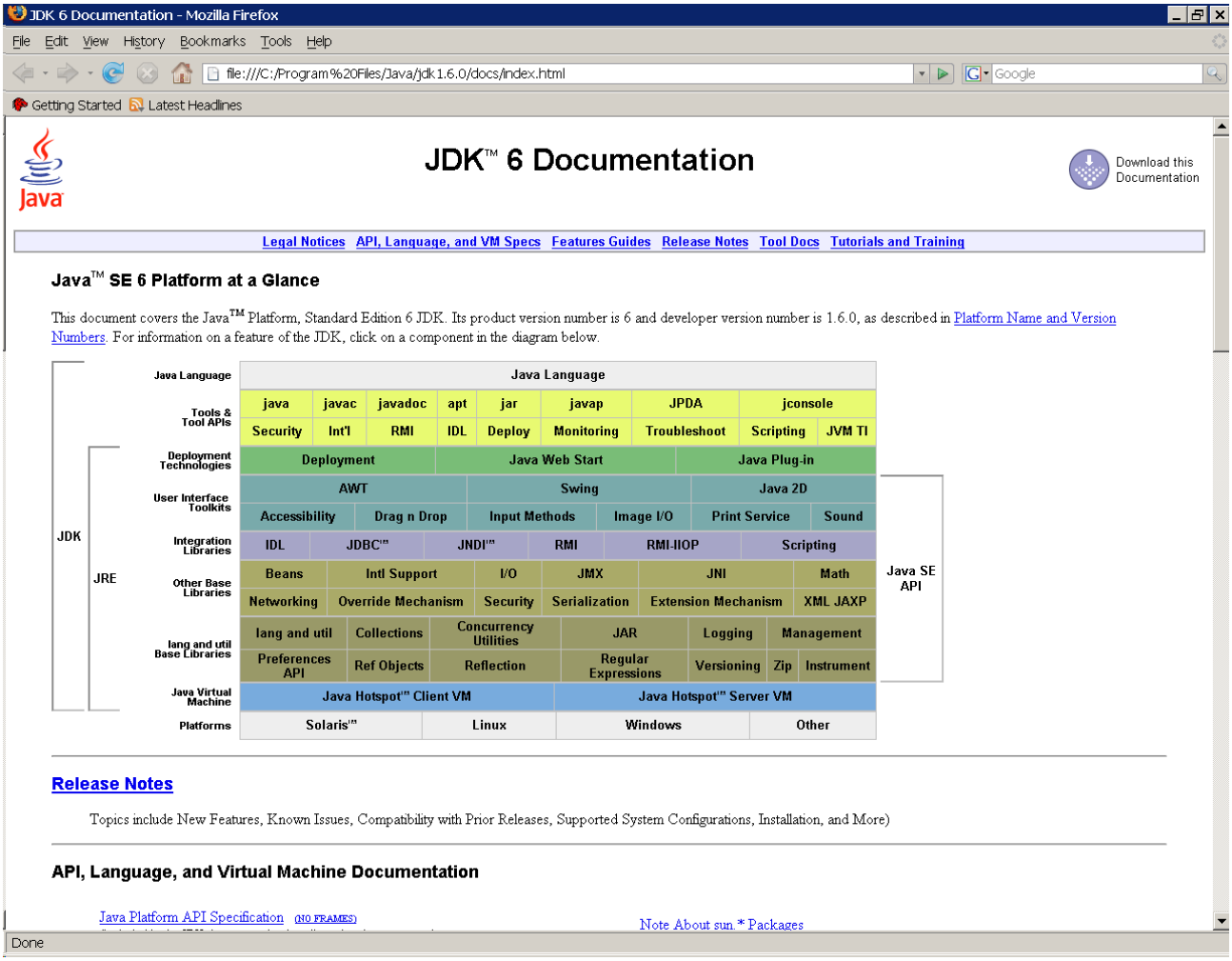


Aici, căutăm prin User Variables până găsim pe PATH, unde vom scrie pe `C:\Program Files\java\jdk1.6.0\bin`, apoi apăsam butonul **OK.** Deschideţi o consolă windows nouă (din `Start > Run > ... >cmd`) în directorul în care se află stocat fişierul sursă Java (mai jos acest director este `D:\My Documents\Cursuri\Curs Java\cod`), apoi verificaţi dacă instalarea şi configurarea este corectă prin `path` şi `java -version`. Path ne arată că am introdus corect calea, iar java -version că JVM-ul s-a instala corect.



Instalarea surselor de biblioteci şi documentaţiei Java

Fişierele care conţine sursele bibliotecilor din JDK se instalează într-o formă comprimată în fişierul `src.zip`. Este recomandată decomprimarea şi instalarea lui pentru accesul la codul sursă în directorul `src`. Documentaţia este conţinută într-un fişier separat de JDK ce trebuie descărcat de pe http://java.sun.com/docs , pentru J2SE, versiunea 6 acesta este `jdk-6-doc.zip`. Acest fişier se va decomprima în directorul `docs` şi aeste în format HTML. Va putea fi vizualizate cu orice navigator de Internet începând cu `index.html`. Cel mai simplu este să se pună un bookmark în navigator către acest fişier.



Structură de directori ai JDK-ului va fi în final:

jdk (numele versiunii actuale este jdk1.6.0, se modifică funcţie de versiune)
- bin (compilator java şi alte instrumente de dezvoltare)
- demo (exemple de aplicaţii Java)
- docs (documentaţia bibliotecile în HTML, după decomprimarea lui `jdk-6-doc.zip`)
- include (fişiere pentru compilarea metodelor native)
- jre (fişierele mediului de execuţie Java)
- lib (fişierele de bibliotecă)
- src (fişierele sursă de bibliotecă, după decompimarea lui `src.zip`)

Etapele generării unei aplicații Java

Etapa:

1 Editarea: Salut1 ➡ salvare pe disc cu extensia .java
2 Compilarea: javac Salut1.java ➡ Salut1.class
3 Rularea: java Salut1
   A. Încărcătorul de clase copiază conţinutul lui Salut1.class de pe disc în RAM
   B. Verificatorul codului de biţi se asigură că sistemul de securitate Java nu este violat
   C. Interpretorul citeşte din RAM şi traduce codul de biţi în limbaj maşină şi îl execută.

Etapele necesare rulării unei aplicaţii Java

Stocarea textul sursă.

Toate programele Java sunt formate din obiecte. Există două tipuri de obiecte: tipurile de date fundamentale (numite unori, primitive sau de bază) şi tipurile de date definite de utilizator, numite clase. Tipurile primitive sunt definite la nivelul limbajului Java, iar clasele sunt definite de către programator sau de către alţi dezvoltatori. În continuare se prezintă o aplicaţia Java, formată dintr-o clasa numită **`Salut1`**, care va afişa pe ecran textul **`Salut!`**.

Extensia fişierului sursă trebuie sa fie **.java**, iar numele acestuia **Salut1**. Numele clasei şi cel al fişierului sursă trebuie să fie acelaşi.

Conţinutul fişierului sursă Java Salut1.java este:

```
// Salut1.java
public class Salut1 {
   // executia unei aplicatii Java incepe cu metoda main.
   public static void main( String args[] )
   {
      System.out.println("Salut !" );
   } // terminare metoda main
} // terminare clasa Salut1
```

Explicarea liniilor codului sursă Java

Linia `// Salut1.java` reprezintă un comentariu care se întinde pe o singură linie (începând cu primul carater de după `//` şi până la trecerea la linia nouă) din textul sursă. Comentariile ajută la înţelegerea codului Java explicând rolul unor secvenţe de cod. Ele nu apar în programul executabil,

liniile în cauză fiind ignorate de către compilator. Atunci când este nevoie de comentarii mai lungi, ce trebuie să se întindă pe mai multe linii, se va folosi perechea de delimitatori `/*`, `*/`.

Limbajul Java este "case sensitive" adică face diferenţa între scrierea cu litere mari şi mici. Dacă în loc de main se va scrie Main aplicaţia nu se va rula.

`public` se numeşte **modificator de acces** , acesta permite definirea controlul accesului altor porţiuni de program la această porţiune de cod.

`class` reprezintă începutul unei declaraţii de clasă şi este urmată de numele clasei (mai sus, `Salut1`). Orice aplicaţie Java trebuie conţină cel puţin o singură declaraţie de clasă. Clasa reprezintă un şablon pentru descrierea stării şi a comportamentului asociat unui obiect din acea clasă. Procesul de creare a unei obiect pe baza unei clase se numeşte instanţiere. Starea unui obiect este stocată în variabilele membri, iar comportamentul ei se implementează prin metode.

`{` - acolada deschisă marchează începutul corpului declaraţiei de clasă şi va avea întotdeauna corespondentă o acoladă închisă - `}` ce marchează terminarea declaraţie de clasă. Porţiunea de cod sursă delimitată de aceste acolade poartă denumire de bloc.

Linia este locul de pornire a programului `public static void main( String args[] )`. Numele `main` este urmat de o paranteză rotundă `(`, un parametru şi o paranteză rotundă închisă `)`. Prezenţa parantezei spune compilatorului Java că `main` este o metodă şi nu o altă construcţie de limbaj. Atunci când dorim să rulăm o clasă pe JVM trebuie să-i specifică doar numele, interpretorul Java apelând automat metoda main a clasei. Metoda main este precedată de trei modificatori:

`public` - care permite ca orice altă clasă să poată apela metoda main;
`static` - spune că metoda main nu operează cu obiecte (deoarece în momentul pornirii aplicaţiei încă nu avem obiecte), tehnic vorbind aceasta nu-l are pe `this` (vezi referinţa this). În general, metoda main este cea care construieşte obiectele necesare programului. Dacă însă dorim să apelăm numai metode ale clasei nu mai este necesară construcţia (aceasta se face cu operatorul `new`) lui; construcţia este obligatorie dacă se doreşte accesarea de metode sau de variabile de instanţă (vezi conceptul de clasă);
`void` - indică faptul că metoda main nu întoarce o valoare.

Linia `{System.out.println("Salut !");}` este echivalentă cu liniile (în sensul că din punctul de vedere al compilatorului Java efectul compilării este acelaşi - rezultă acelaşi cod de biţi - dar pentru o citire şi înţelegre mai uşoară se preferă cea de a două scriere - cu spaţii şi pe mai multe linii):

```
{
  System.out.println(  "Salut !"  );
}
```

Aici acoladele marchează începutului şi terminarea corpului metodei main. Metoda are o singură instrucţiune care foloseşte un pachet de intrare/ieşire pentru afişarea în fereastra din care s-a rulat aplicaţia a textului `Salut !`. Textul de afişat se scrie între ghilimele şi poartă denumirea de şir de caractere sau mai pe scurt, şir. Metodele în Java pot avea parametri, metoda `System.out.println` face afişarea parametrului şir. Caracterul `;` se numeşte terminator şi face ca metoda să fie considerată o instrucţiune Java. Clasa `System` este din pachetul `java.lang` şi conţine, printre altele, metode pentru interacţiune între JMV şi mediul extern (recunoaşte carecteristicile platformei pe care rulează).

Prin `System` se pot accesa fişierele standard de intrare (`in`), ieşire (`out`) şi de erorare (`err`).

Este o eroare de sintaxă dacă un şir nu este cuprins în ghilimele la nivelul programului sursă sau dacă se omite un caracter terminator de instrucţie ;.

Compilarea programului sursă Java

Pentru compilarea programului Java stocat în fişierul **`Salut1.java`** se va folosi compilatorul **`javac`** astfel: **`javac Salut1.java`**. Compilatorul Java va crea fişierul **`Salut1.class`** ce conţine codul de biţi al aplicaţiei.

Rularea aplicaţiei Java

Deoarece codul de biţi este o reprezentare intermediară programului Java el nu poate fi rulat direct de sub sistemul de operare. Rularea necesită JVM care se porneşte cu **`java Salut1`**. JVM trebuie să primească numele clasei de rulat, în exemplul prezentat acesta este **`Salut1`**. Dacă numele clasei este acelaşi cu cel al fişierului în care acesta este stocat, căutarea clasei se va face automat pe baza numelui de fişier care are extensia .**`class`**.

C1- Întrebari

1. Masina Virtuala Java este un dispozitiv hardware?
2. Care sunt avantajele limbajului Java?
3. Ce sunt JDK si JRE?
4. Care sunt numele compilatorului si masinii virtuale?
5. Care este impachetarea pe care incep sa dezvolte majoritatea programatorilor?

BIBLIOGRAFIE

1. http://www.oracle.com/technetwork/java/javase/documentation/index.html

2. http://docs.oracle.com/javase/6/docs/

3. Stefan Tanasa, Cristian Olaru, Stefan Andrei, Java de la 0 la expert, Polirom, 2003, ISBN: 973-681-201-4.

4. Herber Schild, Java 2 - The Complete Reference, Fourth Edition, Osborne, 2001, ISBN: 0-07-213084-9.

5. Deitel H.M., Deitel P. J., Java - How to programm, Fith Edition, Prentice Hall, 2003, ISBN: 0-13-120236-7.

6. http://www.east.utcluj.ro/mb/mep/antal/downloads.html