Universitatea Politehnică Bucureşti

Facultatea de Electronică, Telecomunicații şi Tehnologia Informației

Tema :

Diferentele intre modelarea dotNET si JAVA.

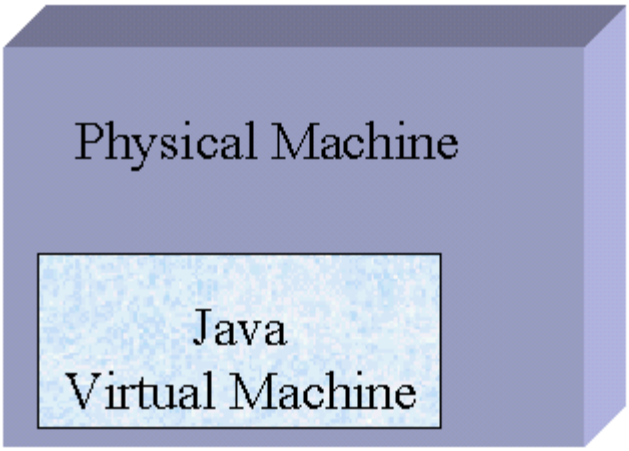
Zanfir Catalin
441A

Java si JVM

In programarea de nivel inalt ce foloseste limbaje precum C si C++, scriem un program intr-o forma usor de citit pentru oameni , iar un program intern demunit compiler translateaza codul in format binar denumit cod executabil pe care computerul nostru il poate intelege si executa. Codul executabil depinde de masina computerului pentru a executa programul, adica este dependent de masina. In Java, acest procesul de scriere de cod este foarte similar , dar exista o diferenta importanta ce ne permite sa scriem programe Java ce sunt independente de masina folosita.

Toate programele Java sunt compilate intr-un nivel intermediar denumit bytecode folosind un interpretor. Putem rula codul compilat bytecode pe orice computer ce are mediul Java runtime instalat. Mediul Java runtime consta dintr-o masina virtuala si codul sustinator.

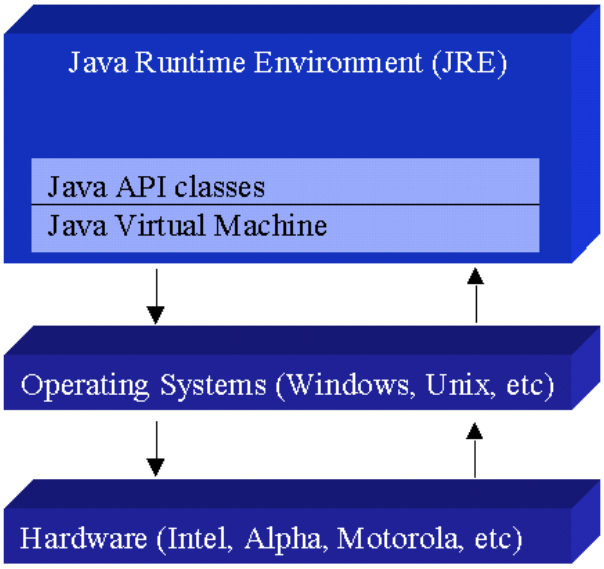
Partea dificila din crearea unui bytecode Java este faptul ca, codul sursa este compilat pentru o masina care nu exista. Aceasta masina este denumita Masina Virtuala Java (Java Virtual Machine –JVM), si ea exista numai in memoria calculatorului nostru. O cale de a trece peste acest obstacol este pacalirea compilatorului Java pentru a crea un bytecode, ar interpretorul Java trebuie sa faca computerul si fisierul bytecode sa creada ca ele sunt rulate pe o masina reala. El face asta prin jucarea rolului de intermediar intre Masina virtuala si masina computerului .



[sursa]

Aceasta figura reprezinta emularea unei masini virtuale Java pe o masina fizica.

Masina virtuala Java este responsabila pentru interpretarea bytecode-ului Java si translatarea acestuia in actiuni sau apeluri ale sistemului de operare. De exemplu, o cerere pentru stabilirea unei conexiuni socket catre o masina indepartata va implica un apel al sistemului de operare. Sistemele de operare pot fi diferite si pot manipula socket-urile in diferite feluri, dar programatorul nu trebuie sa se ingrijoreze de asemenea detalii. Este responsabilitate masinei virtuale Java pentru a manipula aceste translatari astfel incat sistemul de operare si arhitectura computerului pe care software-ul Java ruleaza este complet irelevant pentru proiectant.

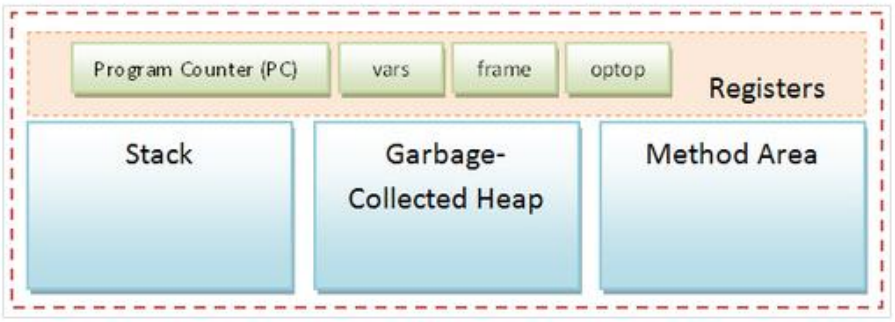


Translatarile pe care masina virtuala Java le manipuleaza [sursa]

Componentele de baza ale masine virtuale Java: [6][7]

Crearea unei masini virtuale in memoria computer-ului nsotru necesita construirea fiecarei functii majore ale unui calculator real pana la mediul in care programele opereaza. Aceste functii pot fi impartite in sapte parti de baza:

- un set de registrii
- stiva
- mediu de executie
- garbage collector
- rezerva de constante
- blocuri de stocare
- set de instructiuni



[sursa]

Registrii. [6] [7]

Registrii masinii virtuale Java sunt similari cu registrii din calculatorul nostru, insa , deoarece masina virtuala e baza pe stiva, registrii nu sunt folositi pentru pasa si a receptiona argumente. In java, registrii retin starea masinii si sunt actualizati dupa executia fiecarei linii din bytecode pentru a-si mentine starea. Urmatorii patru registrii retin starea masinii virtuale

- Frame, contine un pointer pentru mediul de executie al metodei curente
- Optop: contine un pointer pentru varful stivei de operatori si este folosit pentru evaluarea expresiilor aritmetice.
- pc: numarator de program , contine adresa urmatorului byte din codul ce se executa
- vars: registrul variabilelor , contine un pointer catre variabilele locale.

Toti acesti registrii au 32 de biti si sunt alocati imediat. Acest lucru este posibil deoarece compilatorul stie dimenisiunea variabilelor locale si ale stivei de operatori, si pentru ca interpretorul stie dimensiunea spatiului de executie.

Stiva. [6] [7]

JVM foloseste o stiva de operatori pentru a alimenta parametri catre metode si operatii si pentru a receptiona rezultatele de la ele. Toate instructiunile bytecode iau operanzi din stiva , opereaza cu ele, si returneaza rezultate catre stiva. Precum registrii din VM , dimensiunea stivei de operanzi este de 32 biti.

Stiva foloseste o metodologie LIFO – last in, first out, ultimul intrat- primul iesit si se asteapta ca operanzii din stiva sa fie pusi intr-o ordine specifica. De exemplu, bytecode-ul isub se asteapta ca doi intregi sa fie stocati in varful stivei, ceea ce inseamna ca operanzii trebuie sa fie plasati in stiva de setul precedent de instructiuni. isub scoate operanzii din stiva , ii scade,si apoi pune rezultatul inapoi in stiva.
In Java , integer-ii sunt un tip de date primitive. Fiecare tip de data primitiva are instructiuni unice ce spun cum se manipuleze operanzii de acel tip. De exemplu: bytecode-ul lsub este folosint pentru a face o operatie de scadere de tip long integer, fsub bytecode este folosit pentru a face operatii de defirenta cu date de tip float. Din cauza asta, este ilegal sa punemi doua variabile integer in stiva si apoi sa le tratam ca o singura variabila de tip long integer, insa este legal sa pune o variabila de 64 de biti in stiva si sa ocupe doua slot-uri de 32 de biti.

Fiecare metoda din programul nostur java are un cadru de stiva asociat cu el. Cadrul stiva retine starea fiecarei metode cu trei seturi de date: variabilele locale ale metodei, mediul de executie al metodei, stiva de operanzi a metodei. Desi dimensiunea variabilelor locare si ale mediului de executie sunt intotdeauna fixe la inceputul apelarii metodei, marimea stivei de operanzi se schimba cand instructiunile metodei bytecode sunt executate. Deoarece stiva java este de 32 de biti, numerele pe 64 de biti nu sunt garantate sa fie aliniate in 64 de biti.

Mediul de executie. [6] [7]

Mediul de executie este mentinut in cadrul stivei ca un set de date si este folosit pentru a manipula legarea dinamica, iesirile metodelor si generarea exceptiilor. Pentru a manevra legarea dinamica, mediul de executie contine referinte simbolice catre metode si variabile pentru metoda curenta si clasa curenta. Aceste apelari simbolice sunt translatate in apelari actuale de metode prin legarea dinamica pentru un tabel de simboluri.

Oricand o mentoda se termina normal , o valuare este returnata catre metoda care a apelat. Mediul de executie manevreaza iesirile metodelor normale prin restaurarea registrilor ale apelatorului si incrementeaza contorul de programar al apelatorului pentru a sari peste instructiunile de apel a metodei.

Daca executia metodei curente se termina normal, o valoare este returnata metodei apelatoare. Asta se intampla in cazul in care metoda apelatoare executa o instructiune de returnare adecvata tipului de retur.

Daca metoda apelatoare executa o instructiune de returnare care nu este adecvata tipului de returnare , metoda arunca o exceptie sau o eraore. Erorile ce apar pot include greseli are legarii dinamice, exemplu: erori in gasirea unei clase, sau erori runtime : cum ar fi referinta unei variabile in afara limitelor unui vector. Cand erorile apar , mediul de executie genereaza o exceptie.

Stiva Garbage-Collector [6] [7]

Fiecare program ce ruleaza in mediul Java are o stiva de ‘colectare a gunoiului ‘ asociat. Deoarece instantierile claselor sunt alocate din aceasta stiva, alt mod de a denumi aceasta stiva este ‘rezerva alocarii de memorie’ . in general dimensiunea ei este de 1MB.

Desi stiva este setata pentru a avea o dimensiune specifica, cand lansam un program, ea poate creste; de exemplu , cand se aloca obiecte noi. Pentru a asigura faptul ca stiva aceasta nu este prea mare, obiectele ce nu mai sunt folosite sunt automat dealocate memorie sau colectate de garbage-collector al al masiniii virtuale Java.

Java efectuaza colectari de gunoaie in mod automat , ca operatie pe fundal. Fiecare fir de executie ce ruleaza in mediul Java runtime are doua stive asociate lui: prima stiva este pentru cod Java, iar a doua este pentru cod C. Memoria folosita de aceste stive este extrasa din rezerva totala de memorie a stivei. Oricand un nou fir isi incepe executia ii este atribuit o stiva maxima pentru cod java si cod C. In mod implicit, pe celel mai multe sisteme, dimensiunea maxima pentru stiva este de 400kB , iar dimensiunea stivei C ese de 128kB.

Rezerva de constante. [6] [7]

Fiecare clasa din stiva are o rezerva de constante asociata cu ea. Deoarece constantele nu se schimba, ele sunt de obicei create la compilare. Elementele din rezerva de constante codifica toate numele folosite de orice metode intr-o clasa particulara. Clasa contine un contor care numara cate constante exista si un offset care specifica unde se afara o listare particulara de constante din descrierea unei clase.

Setul de instructiuni al bytecode-ului: [6] [7]

Desi programatorii prefera sa scrie cod intr-un limbaj de nivel inalt calculatoarele noastre nu pot executa acest cod direct si de aceea noi trebuie sa compilam programele Java inainte de a le rula. In general, codul compilat este fie intr-un format care poate fi inteles de masina si se numeste “limbaj masina”, fie este intr-un format de nivel mediu, cum ar fi limbajul de asamblare sau Java bytecode.

Instructiunile bytecode-ului utilizate de Java Virtual Machine seamana cu instructiunile din limbajul de asamblare. Daca ai folosit vreodata limbajul de asamblare stii ca setul de instructiuni este transmis la un nivel minim de dragul eficientei si acele task-uri (cum ar fi imprimarea ecranului), sunt indeplinite folosind o serie de instructiuni. De exemplu limbajul Java permite sa printam ecranul utilizand o singura linie de cod, cum ar fi:

```
System.out.println("Hello world!");
```

La compilare compilatorul Java converteste aceasta linie de imprimare in urmatorul bytecode:

```
0 getstatic #6 <Field java.lang.System.out Ljava/io/PrintStream;>
   3 ldc #1 <String "Hello world!">
   5 invokevirtual #7 <Method java.io.PrintStream.println(Ljava/lang/String;)V>
   8 return
```

JDK asigura o unealta pentru examinarea bytecode-ului numita “Java class file disassembler”. Putem rula dezasamblorul prin scrierea comenzii “javap” in linia de comanda.

Deoarece intructiunile bytecode-ului sunt intr-un format de nivel scazut, programele noastre sunt executate la o viteza apropiata de cea a programelor compilate in limbaj masina. Toate instructiunile in limajul masina sunt reprezentate de siruri de biti 0 si 1. Intr-un limbaj de nivel scazut, sirurile de biti de 0 si 1 sunt inlocuite de mnemonice potrivite, cum ar fi intructiunea de bytecode “isub”. Ca si in limbajul de asamblare, formatul de baza al unei instructiuni de bytecode este:

```
<operation>   <operands(s)>
```

Prin urmare, o instructiune din setul de instructiuni al bytecode-ului contine 1 bit opcode care specifica operatia care trebuie sa fie realizata si zero sau mai multi operanzi care furnizeaza parametrii sau datele care vor fi folosite de operatie.

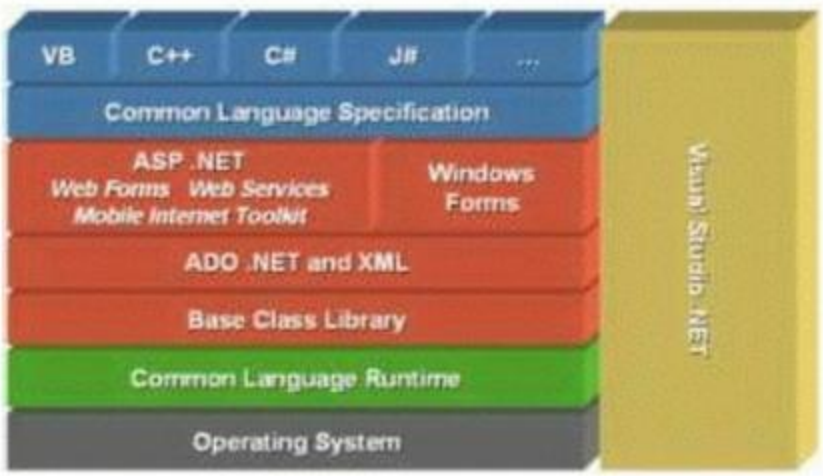
Zona de metode: [6] [7]

Zona de metode Java este similara cu zonele de cod compilat ale mediilor de rulare utilizate de alte limbaje de programare. Aceasta stocheaza instructiuni bytecode care sunt asociate cu metodele din codul compilat si tabelul de simboluri de care are nevoie mediul de executie pentru a putea realiza legarea dinamica. Orice depanare sau alta informatie aditionala care trebuie sa fie asociata cu o metoda este stocata in aceasta zona.

Framework-ul dotNET

Framework-ul .NET este o tehnologie ce suporta crearea si rularea generatiilor urmatoare de aplicatii XML Web. Constituie un nivel de abstractizare intre aplicatia noastra si nucleului sistemului de operare (sau alte programe folosite), pentru a asigura portabilitate codului. Frameworkul .Net este proiectat pentru a indeplini urmatoarele obiective :

- pentru a asigura un mediu consistent de programare obiect-orientata, fie ca obiectul este stocat si executat local, executat local dar distribuit prin internet, sau executat de la distanta
- pentru a asigura un mediu de executie al codurilor care minimizeaza lansarea software-ului
- pentru a asigura un mediu de executie al codurilor ce promoveaza executarea codului in siguranta,incluzand codurile create de o persoana necunoscuta sau de programe terta cu credibilitate scazuta
- pentru a construi toata comunicatia pe standardele industriei pentru a asigura codul bazat pe .net framework poate lucra cu orice aplicatie bazata web.
- Pentru a face experienta de proiectant consistenta pe mai multe tipuri de aplicatii
- Pentru a asigura un mediu de executie al codurilor ce limina problemele de performanta ale mediilor scriptate sau interpretabile



Arhitectura .net [sursa]

Aceasta figura arata arhitectura platformei Microsoft .NET. Orice program scris intr-un dintre limbajele .NET este compilat mai intai in CIL – Common Intermediate Language, si sa fie corect scris din punct de vedere al structurii CLS – Common Language Specification. Cea mai importanta componenta a acestui limbaj .NET Framework este CLR-ul: Common Language Runtime, fiind cel responsabil de executia fiecarui program. In final, ultimul strat este reprezentat de sistemul de operare.

Common Intermediate Language (CIL): [5]

Abstractizarea este unul din instrumentele de care dispune ingineria software. Abstractizarea este folosita in cazurile in care vrem sa nu prezentam utilizatorului toate detaliile programului sau sa punem la dispozitia altor persoane mecanisme generale care sa le rezolve problemele fara a fi nevoie sa se cunoasca toate dedesubturile. Se pot modifica toate detaliile interne ale unui program fara ai modifica interfata si fara a afecta actiunile celorlalti beneficiari ai codului.

In cazul limbajelor de programare abstractizarea a evoluat in decursul anilor, ajungandu-se la nivele de abstractizare a codului rezultat la compilare, cum ar fi: “p-code” (care este produs de compilatorul Pascal-P) si “bytecode” (care este folosit in Java). Bytecode-ul din Java este generat la compilarea programului sursa si este un cod scris intr-un limbaj de nivel mediu care suporta POO. Totodata bytecode-ul Java este o abstractizare care permite executarea de cod Java fara sa depinda de platforma pentru care este destinat, atat timp cat aceasta platforma are implementata o masina virtuala Java care poate sa traduca in continuare fisierul class in cod masina.

Pe de alta parte Microsoft a creat propria sa abstractizare de limbaj care se numeste Common Intermediate Language. Toate limbajele de programare de nivel inalt (C#, Managed C++, Visual Basic .NET si altele) produc la compilare cod de nivel intermediar: Common Intermediate Language (CIL). In mod asemanator cu bytecode-ul de la Java, CIL are caracteristici obiect-orientate, precum: abstractizarea datelor, polimorfismul, mostenirea sau alte concepte necesare, cum ar fi: exceptiile si evenimentele. Se poate remarca ca abstractizarea limbajului ofera posibilitatea de a rula aplicatia in mod independent de platforma (avand aceeasi conditie ca si la Java: sa avem pe platforma respectiva o masina virtuala instalata).

Common Language Specification [2]

Tinta .NET-ului este de a suporta integrarea limbajelor in asemenea fel incat programele pot fi scrise in orice limbaj, dar pot interopera folosindu-se de proprietatile limbajului obiect orientat : mosternire, incapsulare, exceptii, etc. Totusi, limbajele nu pot fi presupuse ‘egale’ sau ‘acelasi lucru’ pentru ca un limbaj poate suporta o trasatura a unui alt limbaj. De aceea Microsoft a lansat CLS-ul; el specifica un set de reguli simple ce sunt necesare pentru integrarea limbajelor

In mod evident, limbajele nu sunt la fel , exista anumite diferente in procesul de codare, declare, dictionar etc, astfel ca se fac remarcate si tipurile de diferenta cum ar fi : de exemplu, unele sunt case sensitive, altele nu; un constructor trebuie sa fie formulat astfel incat sa fie recunoscut si utilizabil de toate limbajele de programare integrate iar cel al clasei de baza (main) trebuie, bineinteles, sa fie executat primul , sau printre primii; trebuie specificate care entitati pot fi supraincarcate si care nu pot fi supraincarcate, diferentele de notare sau parcurgere a vectorilor/tablourilor, daca se incepe cu indici 0 sau 1. Luand toate lucrurile in considerare , Microsoft lanseaza CLS si CTS (common type system) ce contin regulile necesare pentru a facilitate si a efectua cu succes integrarea limbajelor si pentru a asigura interoperabilitatea codului scris in diferite limbaje.

Aceasta combinatie Common Language Specification- Common Type System efectuaza, in esenta in , procesul de interoperare a limbajelor. Regulile emise pentru compilatoarele .NET garanteaza ca orice compilator genereaza un cod care opereaza in mod independent de limbajul sursa.

Common Language Runtime (CLR): [1]

Cea mai importanta componenta a lui .NET Framework este CLR-ul. CLR-ul se ocupa de managementul si executia codului care este scris in limbajele de programare .NET,care se afla in format CIL si este destul de asemanator cu Java Virtual Machine. CLR-ul realizeaza instantierea obiectelor, face verificarile de securitate, stocheaza obiectele in memorie si elibereaza memorie folosind garbage collection.

Dupa ce se realizeaza operatia de compilare a unui program poate rezulta un fisier care are extensia exe, dar care sa nu fie executabil portabil in Windows, ci unul portabil in .NET (NET PE). Codul care rezulta nu este un executabil propriu zis, ci va fi rulat de catre CLR, la fel cum un fisier class este rulat de catre Java Virtual Machine. Tehnologia folosita de CLR este cea a compilarii JIT, adica o implemenatare a unei masini virtuale, care presupune ca atunci cand o metoda este apelata pentru prima data ea este convertita in cod masina. Codul rezultat in urma conversiei este stocat intr-o memorie cache pentru a nu fi recompilat.

Exista trei categorii de compilatoare JIT:

- Normal JIT: care se comporta exact cum este descris mai sus
- Pre-JIT: realizeaza compilarea in cod masina o singura data pentru tot codul. In general este folosit atunci cand se realizeaza o instalare.
- Econo-JIT: este utilizat pentru sisteme cu resurse limitate. Realizeaza compilarea instructiune cu instructiune a codului CIL, fara sa mai memoreze intr-o memorie cache instructiunile scrise in cod masina.

De obicei un compilator JIT este folosit pentru a creste performanta executiei unui program, fiind o alternativa la compilarea repetata a unei portiuni din program in situatia in care avem apelari multiple. Situatia in care codul ce a fost compilat se executa pe diverse procesoare reprezinta un avantaj al mecanismului JIT; daca masina virtuala care este folosita este compatibila cu noua platforma, in acest caz codul va dispune de toate optimizarile posibile, fara sa mai fie nevoie de recompilarea lui (la fel cum se intampla si in C++).

Metadate: [3]

Metadatele pot fi interpretatate ca “date despre date”. In cazul .NET metadatele reprezinta acele detalii care sunt destinate pentru platforma de dezvoltare pentru a le citi si a le folosi. Acestea sunt memorate inpreuna cu codul pe care il descriu.Folosind metadatele, CLR-ul stie cum sa faca instantierea obiectelor , cum sa realizeze apelul metodelor sau cum sa obtina acces la proprietati. Aceste metadate pot fi interogate de o aplicatie pentru a afla ce expune un anumit tip (structura, clasa) prin intermediul unui mecanism care se numeste reflectare.

Metadatele contin cate o declaratie pentru fiecare tip si cate o declaratie a fiecarei clase, proprietate, camp sau eveniment care este atasat tipului respectiv. Metadatele au informatii despre fiecare metoda implementata permitand astfel celui care incarca respectiva clasa sa gaseasca corpul metodei. Totodata mai pot contine declaratii despre “cultura” respectivei aplicatii, cu alte cuvinte despre locul unde se gaseste (folosit in interfata cu utilizatorul).

Garbage collection: [4]

Se stie ca unul dintre procesele care consuma cel mai mult timp in programare este managementul memoriei. Pentru a rezolva aceasta problema se foloseste garbage collection care este un mecanism care se declanseaza atunci cand mecanismul de alocare de memorie raspunde negativ la o cerere de alocare de memorie. Implementarea garbage collection-ului este de tipul “mark and sweep” si se realizeaza astfel: se porneste de la premiza initiala ca intreaga memorie care a fost alocata poate fi disponibilizata, apoi se determina care sunt obiectele care au fost referite de variabilele aplicatiei; acele obiecte care nu mai sunt referite sunt dealocate, iar toate celelalte zone de memorie sunt compactate. Pentru a nu mai creste foarte mult deficienta de performanta nu mai sunt mutate acele obiecte care au o dimensiune de memorie mai mare decat un anumit prag.

In general mecanismul care se ocupa de apelarea garbage collection-ului este CLR-ul. Totodata exista posibilitatea ca si programatorul sa faca apelarea garbage collection-ului daca doreste.

Executia CLR.

Componentele majori ale CLR-ului includ : incarcatorul de clasa , verificatorul, compilatorul JIT, si alte suporturi pentru executie, cum ar fi managementul de cod, managementul de securitate, garbage collector, managementul exceptiilor, debugg-ingul etc.

Class Loader. Cand rulam o aplicatie standard Windows, sistemul de operare incarca aplicatia inainte de a o executa. Cand se ruleaza o aplicatie .NET sistemul de operare recunoaste aplicatia, si o paseaza catre CLR. CLR-ul gaseste punctul de start, de obicei aflat in Main() si il executa pentru a lansa aplicatia. Dar inainte ca functia Main() sa fie executa, incarcatorul de clase trebuie sa gaseasca clasa care

expune clasa Main() si apoi incarca clasa respectiva. In plus, cand Main() instantiza un obiect al unei clase specifice, incarcatorul de clasa se activeaza si el.

Incarcatorul de clasa incarca clasele .NET in memorie si le pregateste pentru executie, dar inainte de a face cu succes asta, trebuie sa localizeze clasa target. Pentru a gasit clasa target, incarcatorul de clasa trebuie sa se uite in cateva locuri diferite, incluzand si fisierul de configuratie a aplicatiei (.config) din directorul curent, si metadata. Odata ce incarcatorul a gasit si a incarcat clasa target , pune in cache tipul de informatie pentru clasa pentru nu o mai incarca inca o data. Cand clasa target este incarcata, incarcatorul injecteaza un ciot mic, ca un prolog de functie,in fiecare metoda din clasa respectiva. Acest ciot este folosit din doua motive : pentru a urmari statusul compilatorul JIT si pentru a face tranzitia dintre cod gestionat si negestionat. In acest punct, clasa incarcata referentiaza alte clase, astfel ca incarcatorul va incerca sa incarce tipurile referite. Daca tipurile referite au fost deja incarcate, incarcatorul de clase nu are nimic de facut. In final, incarcatorul de clase foloseste metadate adecvate pentru a initializa variabilele statice si sa instantieze un obiect al clasei incarcate.

Verificatorul . este componenta ce se executa la runtime ce verifica faptul daca codul este scris corect. Principala diferenta dintre .net si alte medii e ca verifcare se face la runtime. Prin verificarea typesafety la runtime , CLR-ul previne executia unui cod nesigur si garanteaza ca el va fi folosit dupa cum a fost proiectat. Pe scurt, typesafety se refera la fiabilitate.

In contextul executiei CLR , dupa ce incarcatorul de clase a incarcat o clasa si inainte ca codul CIL sa fie executat, verifcatorul se activeaza pentru codul ce trebuie verificat, astfel el se face responsabil pentru verificarea ca : metadatele sunt bine formate, adica ele trebuie sa fie valide, iar codul CIL este typesafe. Amandoua aceste criterii trebuie indeplinite inainte ca codul sa fie executat deoarece compilarea JIT va avea loc numai daca codul si metadatele au fost verificate cu succes. Deoarece verificatorul este o parte componenta a compilatoarelor JIT, el se activeaza numai cand o metoda este invocata, nu cand o clasa este incarcata. Verifcatorul este un pas optional, deoarece un cod de incredere nu va fi niciodata verificat si va trece imediat la compilatorul JIT .

Compilatoarele JIT. Ele joaca un rol important in platforma .NET deoarece toate fisierele .NET contin CIL si metadate, nu cod nativ. Compilatoarele JIT convertesc CIL in cod nativ pentru ca acesta sa fie executat pe sistemul de operare. Pentru fiecare metoda care a fost verificata cu succes, un compilator JIT din CLR va compila metoda si o va converti in cod nativ gestionat. Codul nativ gestionat este necesar deoarece numai codul gestionat este executat pe sistemul de operare. Codul nativ compilat sta in memorie pana cand procesul se opreste si pana cand garbage collector-ul sterge toate referintele si memoria asociata cu procesul.

Bibliografie:

.net
http://en.wikipedia.org/wiki/Common_Intermediate_Language
http://msdn.microsoft.com/en-us/library/8bs2ecf4.aspx
http://www.codeproject.com/Articles/1825/The-Common-Language-Runtime-CLR-and-Java-Runtime-E
http://msdn.microsoft.com/en-us/library/w0x726c2.aspx
http://www.c-sharpcorner.com/uploadfile/puranindia/net-framework-and-architecture/

java
http://en.wikipedia.org/wiki/Java_virtual_machine
http://docs.oracle.com/javase/specs/jvms/se7/html/index.html
http://viralpatel.net/blogs/java-virtual-machine-an-inside-story/
http://www.javacoffeebreak.com/articles/inside_java/insidejava-jan99.html