Metode Avansate de Programare

Curs 1

Obiective

- Aprofundarea programarii orientata obiect (limbajele Java si C#).
- Programare dirijata de evenimente – Interfete grafice.
- Sabloane de proiectare.
- Programare paralela bazata pe threaduri.
- Introducere in analiza si proiectarea sistemelor soft folosind paradigma orientata obiect.

Activitati asociate cursului

- Curs
  - Prezenta obligatorie
- Laborator:
  - Prezenta obligatorie
  - Teme de laborator – nota pentru fiecare L1, L2, …
    - Media (ponderata) lor va fi notata cu LL
  - Proiect – nota LP
  - Atentie: nu se pot recupera in sesiunea de restante !!! reprezinta activitate din timpul semestrului
- Seminar:
  - Prezenta obligatorie
  - Activitatea – poate conduce la cresterea finala a notei finale cu pana la 1 punct– S

Evaluarea

- Examen partial ~ saptamana 8
  - Nota P
  - Atentie!!!

  examenul partial se poate repeta doar in sesiunea de restante!

- Examen final in sesiune:
  - Scris – nota ES
  - Practic– nota EP

  - Nota Finala

  N = (LL * 25 + LP * 10 + P * 15+ ES * 20 + EP * 30)/100 +S

Cateva referinte…

- Bruce Eckel, *Thinking in Java*, Ed. Prentice Hall, 4thedition, 2006.
- Larry O'Brien and Bruce Eckel, *Thinking in C#*, Ed. Prentice Hall, 2002.
- E. Gamma, R. Helm, R. Johnson, J. Vlissides, *DesignPatter ns – Elements of Reusable Object Oriented Software,* Ed. Addison Wesley, 1994.
- Kent Beck, *Test Driven Development: By Example*, Ed. Addison-Wesley Professional, 2002.
- Craig Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development,* Ed. Addison Wesley,2004.
- Tutoriale Java si C#

http://download.oracle.com/javase/tutorial/

http://msdn.microsoft.com/en-us/library/aa288436%28v=vs.71%29.aspx

Date contact si informatii curs/laborator/exemple/slide-uri

Virginia Niculescu

- Web www.ubbcluj.ro/~vniculescu
- Email vniculescu@cs.ubbcluj.ro

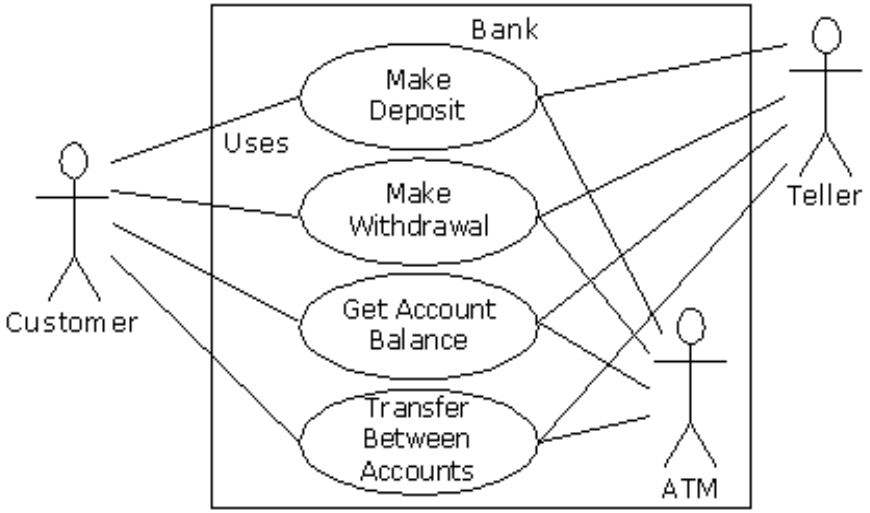
Informatii curs MAP:

www.ubbcluj.ro/~vniculescu/didactic/APM/

Analiza si Proiectare

- “Cine va folosi sistemul?" (pentru a descoperi actorii)
- “Ce pot face acesti actori cu sistemul?"
- “Cum poate sistemul reactiona daca altcineva face acestea?" (pentru a descoperi variatiile)
- “Ce probleme pot apare in sistem?" (pentru a descoperi exceptiile)

1. Descoperirea obiectelor
2. Asamblarea obiectelor
3. Constructia sistemului
4. Extensia sistemului
5. Reutilizarea obiectelor



LOO Pur

- **C++** → limbaj de programare orientat-obiect **hibrid**

Programare OO pura (Alan Kay):
ex. - Smalltalk

- Orice este un obiect!!!
- Un program este o colectie de obiecte care isi trimit mesaje unul altuia.
- Fiecare obiect are propria memorie care contine alte obiecte.
- Orice obiect are un tip.
- Toate obiectele de un tip particular pot primi acelasi set de mesaje. (Substitutie)

Java → 'aproape' un LOO pur
C# -> ~ LOO pur

Limbajul JAVA

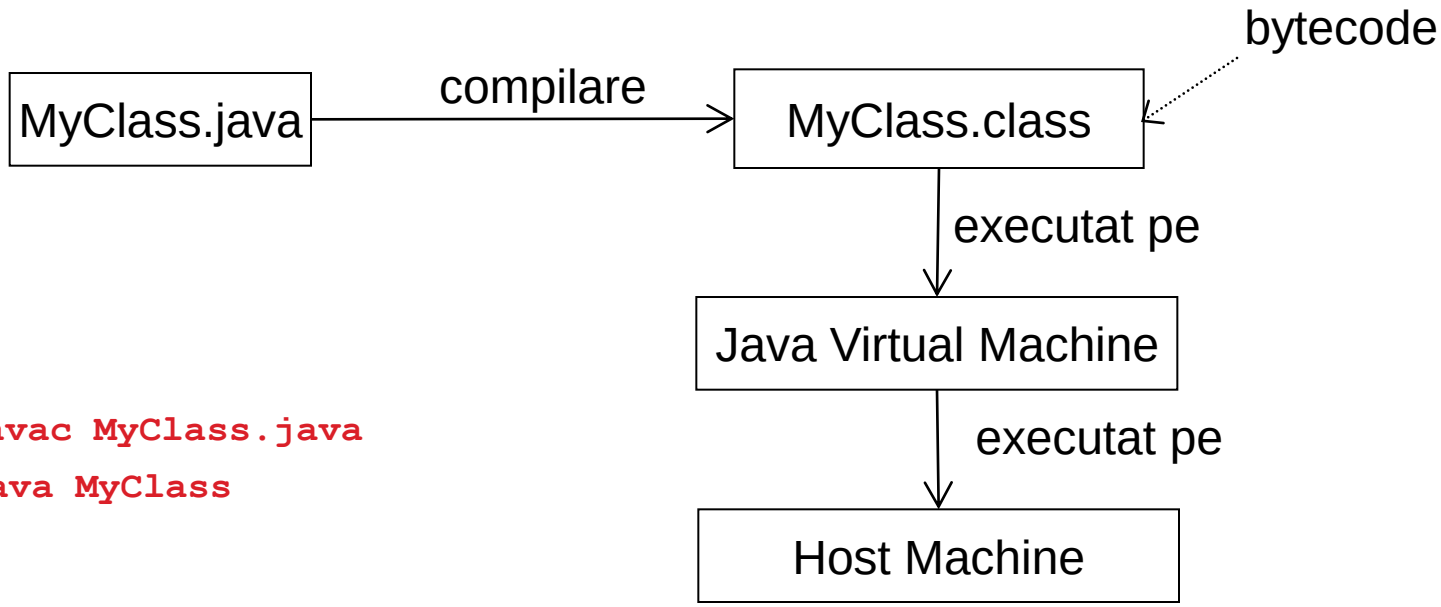
Caracteristici:

- **simplu,**
- **object-oriented,**
- **distribuit si dinamic,**
- **interpretat,**
- **robust,**
- **sigur,**
- **independent de arhitectura,**
- **portabil,**
- **high-performance,**
- **multithreaded**

Java este un limbaj interpretat

- Compilatorul Java genereaza ***byte-code*** pentru Java Virtual Machine (JVM – interpretorul si sistemul run-time), si nu cod masina.
- **A executa un program Java** = interpretorul Java executa byte-codul compilat
- **Byte-codul Java este independent de platforma** ⇒ Programele Java se pot executa pe orice platforma pe care exista JVM.
- Intr-un mediu interpretat, faza standard de **`"link-edit"`** nu mai exista.
- Daca e sa consideram ca **Java** are faza **link**, atunci aceasta inseamna doar procesul de ***incarcare a noilor clase in mediu***, proces care este unul incremental, si apare la **run-time (executie)**.

Cum se executa un program Java?




```
> javac MyClass.java
> java MyClass
```


Program.java
Compiler
Program.class
Java VM
0100101...
Program

Portabilitate si independenta de arhitectura

- O aplicatie Java **se poate executa pe orice sistem**, cu conditia ca pe acel sistem sa fie **implementata Java Virtual Machine**.
- Acest lucru este **important pentru aplicatiile distribuite pe Internet** sau pe **retele heterogene**.
- Formatul **Byte-code** este important pentru *portabilitate*.
- Nu exista aspecte ***"dependente de implementare"*** ale limbajului.

(De exemplu, Java specifica explicit dimensiunea pentru fiecare tip primitiv de date, si deasemenea si aritmetica corespunzatoare.
In C tipul ***int*** poate fi reprezentat pe 16, 32, or 64 biti in functie de platforma.)

- **"Write Once, Run Anywhere."**

Dinamic si Distribuit

- Java este un limbaj *dinamic*.
  - Orice clasa Java poate fi incarcata de catre interpretorul Java la orice moment.
  - Toate aceste clase incarcate dinamic pot fi instantiate dinamic.
  - Bibliotecile de cod nativ pot fi incarcate dinamic de asemenea.
  - Clasa Class: se pot obtine informatii in mod dinamic despre orice clasa.

- Java este un limbaj pentru programare *distribuita* =
  furnizeaza suport de nivel inalt pentru programare distribuita.
  - clasele URL si cele conexe sunt in pachetul java.net,
  - Remote Method Invocation (RMI)
  - suport pentru operatii in retele traditionale de nivel jos (datagrams si stream-based connections prin sockets)

- Aceste caracteristici permit interpretorului Java sa incarce si sa execute cod de pe Internet (ex. Java applets)

Simplitate

- Java este un limbaj ***simplu***
  - poate fi invatat usor
  - numarul constructiilor limbajului a fost pastrat relativ mic
  - sunt multe similaritati cu C++

- Un numar de elemente din C si C++ au fost eliminate:
  - Nu exista `goto;` exista insa instructiuni etichetate `break` si `continue` mecanisme de tratare a *exceptiilor* ,
  - Java nu foloseste fisiere header si elimina preprocesarea din C,
  - Nu exista `struct` si `union` ,
  - Nu exista supraincarcarea *operatorilor si mostenire multipla.*

- **Java nu foloseste pointeri**
  - Java face automat referentierea si dereferentierea obiectelor
  - *Automatic garbage collection*

Robustete

Capacitatea unui sistem/program de a se executa/reactiona corect/""bine" nu doar in conditii optime dar si in conditiile unui input care nu respecta toate conditiile impuse.

- Java este un **limbaj puternic tipizat** ⇒
  verificare la compilarea compatibilitatii tipurilor.
  - Java este mai tipizat decat C++.
  - Java cere declararea explicita a metodelor; nu permite stilul C implicit pentru declararatii C-style ⇒ compilatorul poate verifica erorile de apel.
- Nu sunt **pointeri** ⇒ cresterea robustetii programelor Java prin eliminarea unei intregi clase de erori datorate pointerilor.
- **Toate accesarile la tablouri si string-uri sunt verificate** pentru a se asigura ca sunt in interiorul limitelor si astfel se elimina posibilitatea de suprascriere a memoriei si a distrugerii datelor.
- **Operatiile Cast** ale obiectelor de la un tip la altul sunt de asemenea verificate pentru a se asigura ca sunt legale.
- Mecanismul **automatic garbage collection** previne **'umplerea'** memoriei si alte erori datorate alocarii si dealocarii memoriei.
- **Mecanism de tratare a exceptiilor** puternic!

Securitate

Securitatea este foarte importantă pentru aplicaţiile distribuite!

- Java furnizeaza câteva niveluri de control:

1. La nivelul cel mai de jos, securitatea are legatura cu robustetea.

1. Interpretorul Java incarca un **proces de verificare** a *byte-codului* pentru *orice cod susceptibil* .

(Acest pas de verificare asigura ca este un cod bine format— de exemplu, codul nu va supraincarca stiva si nici nu o va goli si nu contine byte-code ilegal)

3. Modelul *"sandbox":* codul suspect este plasat in "sandbox," unde se poate executa in mod sigur, fara a periclita “lumea reala”, sau mediul Java (este executat cu anumite restrictii.)

3. Prin atasarea de **semnaturi digitale** codului Java, originea codului respectiv poate fi stabilita intr-un mod sigur criptografic.

High-Performance and Multithreaded

- Java este un limbaj interpretat, deci este dificil a se obtine o performatnta ca timp de executie similara programelor C sau C++ (nu este insa exclusa).
- Codul C compilat se executa in general mai repede decat bytecodul Java interpretat.
  - Viteza este insa mai mult decat adecvata pentru aplicatiile interactive(GUI) si pentru aplicatiile retea, pentru care aplicatia asteapta deseori imput de la utilizator sau date din retea.
  - Sectiunile critice d.p.d.v. ale vitezei sunt implementate prin cod nativ eficient
- Multe interpretoare Java includ acum **"just in time compilers"** care pot transforma Java byte-codes in cod-masina pentru un CPU particular.
- Java este un limbaj ***multithreaded*** .

Internationalizare – Unicode

- ***Internationalizare*** este procesul de proiectare a unei aplicatii astfel incat sa poata fi adaptata diferitelor limbi si conventii de notare, fara reprogramare.

UNICODE (www.unicode.org)

- Caracterele Java sunt caractere **16-bit Unicode**.
- Deoarece majoritatea mediilor nu suporta codificarea Unicode, Java foloseste o faza de pre-procesare pentru a asigura faptul ca toate caracterele unui program sunt in Unicode.
- Java defineste secvente **escape** care permite folosirea tuturor caracterelor (**\u*xxxx***, unde ***xxxx*** este o secventa de patru cifre hexazecimale)
- **Unicode** defineste codurile de la 0 la 127 in mod **consistent cu ASCII**.

```
char newline = '\n', apostrophe = '\", delete = '\377‘;
char aleph='\u05D0‘, a = ‘A’ , aa=\u0103;
```

Un exemplu simplu

Hello World!

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

- Compilare

> javac HelloWorld.java

- Executie

> java HelloWorld

Limbajul C#

Limbajul C#

```
using System;
class HelloWorld
{
   public static void Main()
   {
     Console.WriteLine(``Hello world!'');
   }
}
```

C# limbaj folosit in cadrul platformei .NET

In **cmd**

- Compilare

>c:\windows\Microsoft.NET\Framework\v3.5\bin\csc.exe /t:exe /out:MyApplication.exe MyApplication.cs

- Executie

> MyApplication

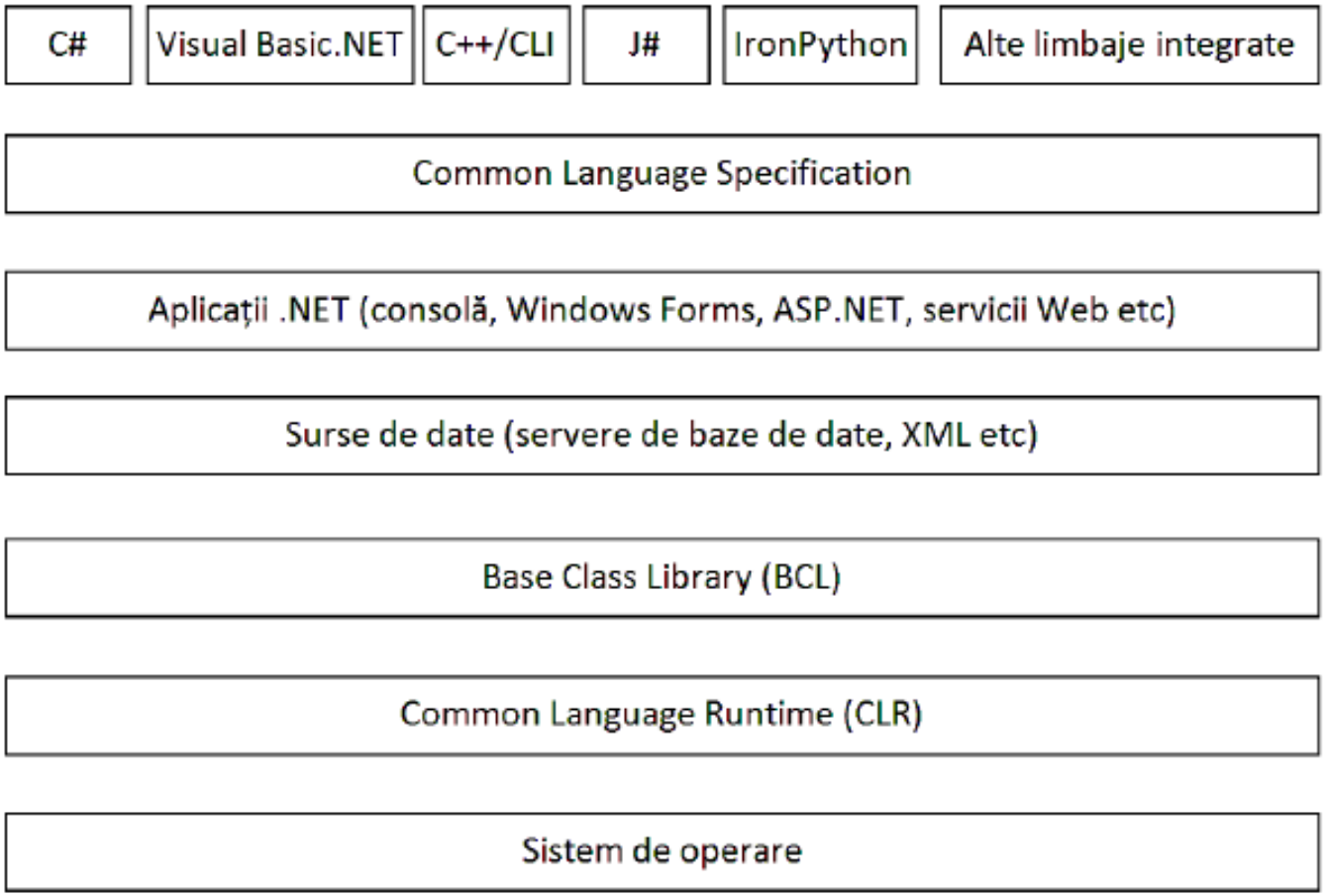
Platforma Microsoft .NET

- .NET Framework constituie un nivel de abstractizare intre aplicatie si nucleului sistemului de operare (sau alte programe), pentru a asigura portabilitatea codului;
  - integreaza tehnologii care au fost lansate de catre Microsoft incepand cu mijlocul anilor 90 (COM, DCOM, ActiveX, etc) sau tehnologii actuale (servicii Web, XML).

- Motivatie (pe scurt)

1. *Aplicatiile distribuite* – sunt din ce ın ce mai numeroase aplicatiile de tip client / server sau cele pe mai multe nivele (n−tier).
2. *Dezvoltarea orientata pe componente* –
3. *Modificari ale paradigmei Web* –
4. *Alti factori de maturizare a industriei software* – constientizarea cererilor de interoperabilitate, scalabilitate, disponibilitate;

Arhitectura platformei .NET



Common Intermediate Language

Abstractizare…

- Microsoft a realizat propria sa abstractizare de limbaj – CIL
- (C#, Managed C++, Visual Basic .NET, etc), la compilare toate vor produce cod ın acelasi limbaj intermediar: CIL
- Asemanator cu bytecod-ul (java), CIL are trasaturi OO
- Aceasta abstractizare de limbaj permite rularea aplicatiilor independent de platforma (cu aceeasi conditie ca la Java: sa existe o masina virtuala pentru acea platforma).

Common Language Specification

- Unul din scopurile .NET :
  - integrarea limbajelor astfel incat
  - programele, desi scrise ın diferite limbaje, pot interopera!

- Common Language Specification (CLS), un subset al lui CTS (Common Type System)

- Contine specificatii de reguli necesare pentru integrarea limbajelor.

Common Language Runtime

- CLR este cea mai importanta componenta .NET Framework.

- Este responsabila cu managementul si executia codului scris in limbaje .NET, aflat in format CIL;
- Este foarte similar cu Java Virtual Machine.
- CLR instantiaza obiectele, face verificari de securitate, depune obiectele in memorie, disponibilizeaza memoria prin garbage collection.
- In urma compilarii unei aplicatii poate rezulta un fisier cu extensia `.exe`, dar care nu este un executabil portabil Windows, ci un executabil portabil .NET (.NET PE). Acest cod nu este deci un executabil nativ, ci se va rula de catre CLR, intocmai cum un fisier `.class` este rulat in cadrul JVM.
- CLR folosete tehnologia compilarii JIT

JIT = Just-in-Time compilation

- JIT
  - o implementare de masina virtuala, in care o metoda sau o functie, in momentul in care este apelata pentru prima oara, este tradusa in cod masina.
  - Codul translatat este depus intr-un cache, evitand-se astfel recompilarea ulterioara.

1. *Normal JIT – a se vedea descrierea de mai sus.*
2. *Pre-JIT – compileaza ıntregul cod in cod nativ o singura data; f*olosit la instalari.
3. *Econo-JIT – folosit pt. dispozitive cu resurse limitate. Compileaza* codul CIL bit cu bit, eliberand resursele folosite de codul nativ ce este stocat in cache.

Common Type System

- Facilitati comune tuturor limbajelor .NET

1. *Tipuri valoare –*
2. *Tipuri referinta –*
3. *Boxing si unboxing –*
4. *Clase, proprietati, indexatori*
5. *Interfete*
6. *Delegati – inspirati de pointerii la functii din C*

Assemblies

- Un assembly reprezinta un bloc functional al unei aplicatii .NET.
- El formeaza *unitatea fundamentala de distribuire, versionare, reutilizare si permisiuni de securitate.*

- *Exista o oarecare similitudine cu fisiere .jar din Java…?!*

  - *Ambele pot contine resurse si metadate*
- *diferente:*
  - *un assembly este compilat*
  - *Putem avea o aplicatie Java fara fisiere JAR; dar nu putem avea o aplicatie .NET fara un assembly..*
- *Exista fisiere JAR executabile => acestea sunt similare cu Assemblies*

Caracteristici ale platformei .NET

- Dezvoltarea multilimbaj
- Independenta de procesor si de platforma
- Managementul automat al memoriei
- Suportul pentru versionare
- Sprijinirea standardelor deschise: (XML,SOAP,HTTP)
- Distribuirea usoara:
- Arhitectura distribuita
- Interoperabilitate cu codul “unmanaged”: Codul “unmanaged” se refera la cod care nu se afla ın totalitate sub controlul CLR.
- Securitate

Recap. ->Tipuri de locatii de stocare

- **Registrii:** (cel mai rapid) in procesor
  - numarul de registrii este limitat
  - fara control direct
- **Stiva:** RAM
  - Procesorul foloseste pointerul de stiva
  - Pointerul stivei este mutat in ‘jos’ pt. a se aloca memorie si in ‘sus’ pentru a se elibera.
  - Rapida si eficienta pentru alocare
  - Referintele la obiecte si valorile primitive
- **Heap**: RAM
  - flexibilitate
  - alocarea necesita mai mult timp decat alicarea pe stiva
  - obiectele Java sunt stocate in heap
- **Stocare statica**: “in locatie fixa” (RAM).
  - datele statice au durata de viata egala cu cea a programului
- **Stocarea constantelor:** constantele sunt stocate in general in codul programului (nu intotdeauna)
- **Stocare Non-RAM**:
  - *streamed objects, persistent objects*

- Elemente de baza de limbaj
  - Java
  - C#

  - Tipuri simple
  - Variable

Java

Tipuri primitive – Java

Tip	Nr. octeti	Valori	Valoare implicita
boolean	-	true, false	false
byte	1 octet	-128 … +127	(byte)0
short	2 octeti	-2^{15} …$2^{15}-1$	(short)0
int	4 octeti	-2^{31} …$2^{31}-1$	0
long	8 octeti	-2^{63} …$2^{63}-1$	0L
float	4 octeti	IEEE754	0.0f
double	8 octeti	IEEE754	0.0d
char	2 octeti	Unicode 0, Unicode $2^{16}-1$	'\u0000' (null)

Declarare Variabile:

```
tip nume_var1[=expresie1][, nume_var2[=expresie2]…];
```

Java operators

Precc.	Operator	Operand Type(s)	Assoc.	Operation Performed
1	++	arithmetic	R	pre-or-post increment (unary)
	--	arithmetic	R	pre-or-post decrement (unary)
	+, -	arithmetic	R	unary plus, unary minus
	~	integral	R	bitwise complement (unary)
	!	boolean	R	logical complement (unary)
	(*type*)	any	R	cast
2	*, /, %	arithmetic	L	multiplication, division, remainder
3	+, -	arithmetic	L	addition, subtraction
	+	string	L	string concatenation
4	<<	integral	L	left shift
	>>	integral	L	right shift with sign extension
	>>>	integral	L	right shift with zero extension
5	<, <=	arithmetic	L	less than, less than or equal
	>, >=	arithmetic	L	greater than, greater than or equal
	instanceof	object, type	L	type comparison
6	==	primitive	L	equal (have identical values)
	!=	primitive	L	not equal (have different values)
	==	object	L	equal (refer to same object)
	!=	object	L	not equal (refer to different objects)
7	&	integral	L	bitwise AND
	&	boolean	L	boolean AND
8	^	integral	L	bitwise XOR
	^	boolean	L	boolean XOR
9	|	integral	L	bitwise OR
	|	boolean	L	boolean OR
10	&&	boolean	L	conditional AND
11	||	boolean	L	conditional OR
12	?:	boolean, any, any	R	conditional (ternary) operator
13	=	variable, any	R	assignment
	*=, /=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=	variable, any	R	assignment with operation

Scope = {…} (Domeniu de vizibilitate)

- Nu este permisa “ascunderea unei variabile intr-un sub-scope

```
{int x = 12;
  { int x = 96; /* illegal */
  }
}
```

Object scope

```
{
String s = new String("a string");
} /* end of scope */
```

C#

Tipuri simple – tipuri valoare C#

- In C# exista *tipuri valoare si tipuri referinta.*
  - *Tipurile valoare includ :*
    - *tipurile simple (ex. char, int, float),*
    - *tipu*l enumerare si
    - tipul structura
  - **au ca principale caracteristici faptul ca ele contin direct datele referite si sunt alocate pe stiva sau *inline intr-o structura.***

- *Ti*purile referinta includ tipurile:
  - clasa,
  - interfata,
  - delegat
  - tablou,
  - **Au proprietatea ca variabilele de acest tip stocheaza referinte catre obiectele care sunt alocate in heap.**

Ierarhie unica de clase

- Toate tipurile de date sunt derivate (direct sau nu) din tipul System.Object,
- **Mod unitar de tratare !**

Tipuri predefinite

- Set de tipuri predefinite, pentru care nu este necesara referirea vreunui spatiu de nume via directiva using sau calificare completa.
- Exemple:
  - string,
  - object,
  - tipurile ıntregi cu semn ¸si fara semn,
  - tipuri numerice ın virgula mobila,
  - tipurile bool si decimal.

- Tipul string este folosit pentru manipularea ¸sirurilor de caractere codificate Unicode; continutul obiectelor de tip string nu se poate modifica

Tipuri primitive in C#

Tip	Descriere	Exemplu
object	rădacina oricărui tip	object a = null;
string	o secvenţă de caractere Unicode	string s = “hello”;
sbyte	tip întreg cu semn, pe 8 biţi	sbyte val = 12;
short	tip întreg cu semn, pe 16 biţi	short val = 12;
int	tip întreg cu semn, pe 16 biţi	int val = 12;
long	tip întreg cu semn, pe 64 biţi	long val1 = 12; long val2=34L;
byte	tip întreg fără semn, pe 8 biţi	byte val = 12;
ushort	tip întreg fără semn, pe 16 biţi	ushort val = 12;
uint	tip întreg fără semn, pe 32 biţi	uint val = 12;

ulong	tip întreg fără semn, pe 64 de biţi	ulong val1=12;
		ulong val2=34U;
		ulong val3=56L;
		ulong val4=76UL;

float	tip cu virgulă mobilă, simplă precizie	float val=1.23F;
double	tip în virgulă mobilă, dublă precizie	double val1=1.23; double val2=4.56D;
bool	tip boolean	bool val1=false; bool val2=true;
char	tip caracter din setul Unicode	char val='h';
decimal	tip zecimal cu 28 de cifre semnificative	decimal val=1.23M;

Tipuri valoare

- Toate tipurile valoare sunt derivate din clasa System.ValueType, care la randul ei este derivata din clasa object (alias pentru System.Object).
- Nu este posibil ca dintr-un tip valoare sa se deriveze!
- Atribuirea pentru un astfel de tip ınseamna copierea valorii!!!

- Tipurile simple sunt identificate prin cuvinte rezervate, dar acestea reprezinta doar alias-uri pentru tipurile struct corespunzatoare din spatiul de nume System.

Tipuri simple si corespondentele lor cu tipurile din spatiul de nume System

Cuvânt rezervat	Tipul alias
sbyte	System.SByte
byte	System.Byte
short	System.Int16
ushort	System.UInt16
int	System.Int32
uint	System.UInt32
long	System.Int64
ulong	System.UInt64
char	System.Char
float	System.Single
double	System.Double
bool	System.Boolean
decimal	System.Decimal

Exemple

```
int i = int.MaxValue; //constanta System.Int32.MaxValue
string s = i.ToString(); //metoda System.Int32.ToString()
string t = 3.ToString(); //idem
double d = Double.Parse("3.14");
```

Tipul decimal

- Este un tip de date reprezentat pe 128 de biti,
- gandit a fi folosit ın calcule financiare sau care necesita precizie mai mare.
- Poate reprezenta valori aflate ın intervalul 1.0×10^{-28} ¸ 7.9×10^{28}, cu 28 de cifre semnificative.
- nu poate reprezenta zero cu semn, infinit sau NaN.
- nu se fac conversii implicite ıntre nici un tip ın virgula mobila si decimal
- nu este posibila mixarea variabilelor de acest tip ıntr-o expresie, fara conversii explicite.
- Literalii se exprima folosind ca sufix caracterele m sau M.

Tipul bool

- Este folosit pentru reprezentarea valorilor de adevar true si false.
- Nu exista conversii standard ıntre bool si nici un alt tip.

Instructiuni de control

- Java
- C#

- Instructiunea compusa:

```
{
instructiune1;
instructiune2; …
}
```

- Instructiunea Daca:

```
if (expresie_logica)
   instructiune;
if (expresie_logica)
   instructiune1;
else
   instructiune2;
```

Obs: **expresie_logica trebuie sa se evalueze la true sau false. Nu se** accepta valori numerice.

Instructiunea switch:

```
switch (integral-selector) {
    case integral-value1 : statement; [break;]
    case integral-value2 : statement; [break;]
    case integral-value3 : statement; [break;]
    case integral-value4 : statement; [break;]
    case integral-value5 : statement; [break;]
    // ...
    default: statement;
}
```

Instructiuni de ciclare(Java)

- Instructiunea while:

while(expresie_booleana)
instructiune

- Instructiunea do-while:

do
instructiune
while(expresie_booleana);

- Obs: **Instructiunea este executata pana cand expresia_booleana** devine **false.**
- Instructiunea for:

for(initializari; expresie_booleana; pas)
instructiune

- Obs: Oricare din expresiile **initializari, expresie_booleana, pas** poate sa lipseasca.

- Instructiunea **return**:

```
return;
return valoare;
```

- Instructiunea **break**: **opreste executia unei iteratii.**

```
int[] x= { 2, 8, 3, 5, 12, 8, 62};
int elem = 8;
boolean gasit = false;
for (int i = 0; i < x.length; i++) {
    if (x[i] == elem) {
        gasit = true;
        break;
    }
}
```

- Instructiunea **continue**:
  - sare peste iteratia curenta din interiorul unui ciclu (for, while, dowhile)
  - opreste executia instructiunilor din interiorul unui ciclu si determina reevaluarea expresiei booleene.

```
int[] x= { 2, 8, 3, 5, 12, 8, 62};
int elem = 8;
int nrApar=0;
for (int i = 0; i < x.length; i++) {
    if (x[i] != elem)
        continue;
    nrApar++;
}
```

C#

- if (expresie logica) instructiune;
- if (expresie logica) instructiune; else instructiune;

- switch (expresie)

{
case eticheta: instructiune;
case eticheta: instructiune;
...
default: instructiune;
}

Exemplu

```
switch (i)
{
case 0:
    Console.WriteLine("0");
    break;
case 1:
    Console.Write("Valoarea ");
    goto case 2;
case 2:
case 3:
    Console.WriteLine(i);
    break;
case 4:
    goto default;
default:
    Console.WriteLine("Numar in afara domeniului admis");
    break;//neaparat, altfel eroare de compilare
}
```

Exemple instructiuni de ciclare

```
while (r != 0)
{
    r = a%b; a = b; b = r;
}

do
{
S += i++;
}while(i<=n)

for (int i=0; i<n; i++)
{
Console.WriteLine("i={0}", i);
}
```

Instructiuni de salt (NERECOMANDATE)

Edsger W. Dijkstra, "Go To Statement Considered Harmful": http://www.acm.org/classics/oct95/

- Permit schimbarea ordinii de executie a instructiunilor
  - *Break,continue, goto, return, throw.*
- Instructiunea break

Produce iesirea fortata dintr-un ciclu de tip *while, do, for, foreach.*

- Instructiunea continue

Porneste o noua iteratie ın interiorul celui mai apropiat ciclu continator de tip *while, do, for, foreach.*

- Instructiunea goto

Goto permite saltul al o anumita instructiune. Are 3 forme:

- goto eticheta;
- goto case expresieconstanta;
- goto default;

Cerinta este ca eticheta la care se face saltul sa fie definita ın cadrul functiei curente si saltul sa nu se faca ın interiorul unor blocuri de instructiuni.

Instructiunile checked si unchecked (C#)

- Controleaza contextul de verificare de depasire a domeniului pentru aritmetica pe ıntregi si conversii. Au forma:

```
checked
{
//instructiuni
}
unchecked
{
//instructiuni
}
```

- Verificare se va face la run-time.

Exemplu

```
byte i=255;
unchecked
{
    i++;
}//i va avea valoarea 0, nu se semnaleaza eroare
checked
{
    i=255;
    i++;//se va arunca exceptie System.OverflowException
}
```