Programare Java

Curs – 4

REFERINTE LA OBIECTE

Pe masura ce lucram cu obiecte un lucru important de inteles il reprezinta folosirea referintelor .

O referinta este un tip de pointer folosit pentru a indica valoarea unui obiect .

Atunci cand atribuim un obiect unei variabile sau trasmitem un obiect ca argument pentru o metoda nu folosim de fapt obiecte . Nu folositi nici macar copii ale obiectului . De fapt folosim referinte catre acele obiecte .

```
Import java.awt.Point;
class TestReferinte {
public static void main(String arg[]) {
Point pt1,pt2;
pt1=new Point(100,100);
pt2=pt1;

pt1.x=200;
pt1.y=200;
System.out.println("Punct 1: "+pt1.x+", "+pt1.y);
System.out.println("Punct 2: "+pt2.x+", "+pt2.y);
}
}
```

Desi la o prima vedere variabilele pt1 si pt2 ar trebui sa aiba valori diferite totusi nu este asa . Variabilele x si y pentru pt2 au fost si ele schimbate chiar daca in program nu se vede nimic explicit . Motivul este ca in linia 6 s-a creat o referinta de la pt2 la pt1 , in loc sa se creeze pt2 ca un nou obiect , copiat din pt1.
pt2 este o referinta la acelasi obiect ca si pt1 . Oricare dintre variabile poate fi folosita pentru a referi obiectul sau pentru a-I modifica variabilele .

Daca doream ca pt1 si pt2 sa se refere obiecte separate , trebuiau folosite instructiuni new Point() separate in liniile 5 si 6 :

```
pt1=new Point(100,100);
pt2=new Point(100,100);
```

Folosirea referintelor in Java devine si mai importanta atunci cand transmitem argumentele metodelor .

CASTINGUL SI CONVERSIA OBIECTELOR SI TIPURILOR PRIMITIVE

Atunci cand transmitem argumente metodelor sau cand folosim variabile in expresii , trebuie sa folosim variabile cu tipuri de date corespunzatoare . Daca metoda creata necesita un int , compilatorul Java raspunde cu eroare daca incercam sa transmitem o valoare float . La fel , daca incercam sa setam o variabila cu valoarea alteia , ele trebuie sa fie de acelasi tip .
Uneori obtinem intr-un program Java o valoare care nu este de tipul necesar . Poate fi dintr-o alta clasa sau dintr-un tip de data necorespunzator – cum ar fi float cand avem nevoie de un int .
Pentru a inlatura acest neajuns trebuie sa folosim casting-ul .

Casting-ul reprezinta procesul de generare a unei valori de un alt tip decat sursa .

Atunci cand efectuam un cast valoarea variabilei nu se schimba ; de fapt se creaza o noua variabila de tipul dorit .

Chiar daca conceptul de casting este relativ simplu , folosirea sa este destul de complicata de faptul ca Java poseda atat tipuri primitive (int , float , boolean , etc) cat si tipuri obiect (String , ZipFile , Point , etc.) . Exista trei tipuri de cast si de conversie despre care vom vorbi :

- castingul intre tipuri primitive (int – float , float-double , etc.)
- castingul intre o instanta a unei clase si o instanta a altei clase
- conversia tipurilor primitive in obiecte si apoi extragerea valorilor primitive din aceste obiecte .

CASTINGUL INTRE TIPURI PRIMITIVE

Cea mai des intalnita situatie este in cazul tipurilor numerice . Exista un tip de date primitiv care nu poate fi niciodata folosit pentru cast ; variabilele booleene sunt fie true fie false si nu pot fi folosite intr-o operatie de cast .
In multe operatii de cast intre tipuri primitive destinatia poate stoca valori mai mari decat sursa , astfel incat valoarea sa fie convertita cu usurinta . Un exemplu ar fi castingul unui byte la int . Deoarece byte-ul stocheaza informatii intre -128 si 127 iar un int informatii intre -2.1 milioane si 2.1 milioane este loc mai mult decat suficient pentru a face cast de la byte la int .

De multe ori putem folosi automat un byte sau un char drept un int ; putem folosi un int drept un long , un int drept un float sau orice drept un double . In majoritatea cazurilor , deoarece tipul de dimensiuni mai mari ofera mai multa precizie decat cel cu dimensiuni mai mici , nu apare nici o pierdere de informatie . Exceptia apare atunci cand facem un cast de la intregi la valori in virgula mobila – castingul unui int sau al unui long intr-un float sau al unui long intr-un double poate duce la pierderi ale preciziei .

Trebuie sa folosim un cast explicit pentru a converti o valoare mai mare intr-una mai mica , deoarece conversia valorii poate duce la o pierdere de precizie . Cast-urile explicite au sintaza :

(numetip)valoare

numetip este numele tipului de data catre care facem conversia ; valoare reprezinta o expresie care are ca rezultat valoarea tipului sursa .

Deoarece precedenta castingului este mai mare decat a operatiilor aritmetice trebuie sa folosim paranteze – altfel putand apare probleme .

CASTINGUL OBIECTELOR

Instantelor claselor li se poate aplica operatia de cast catre instante ale altor clase , cu o restrictie ; clasele sursa si destinatie trebuie sa fie inrudite prin mostenire . O clasa trebuie sa fie subclasa a alteia .

Asemanator conversiei unei valori primitive catre un tip de dimensiuni mai mari , unele obiecte nu au nevoie de cast explicit . In particular , deoarece subclasele contin toate informatiile superclasei lor putem folosi o instanta a unei subclase oriunde se asteapta folosirea unei superclase .
De exemplu , sa luam o metoda care preia doua argumente : unul de tip Object si altul de tip Window . Putem transmite o instanta a oricarei clase drept argument Object (pentru ca toate clasele Java sunt subclase ale clasei Object) ; pentru argumentul Window putem transmite si instante ale subclaselor sale (cum ar fi Dialog , FileDialog sau Frame) .

Reciproca este si ea adevarata ; putem folosi o superclasa acolo unde se asteapta o subclasa . Exista totusi in acest caz o problema – deoarece subclasele au un comportament mai complex decat superclasele lor exista o pierdere de precizie . Acele obiecte superclasa s-ar putea sa nu posede intregul comportament necesar pentru a putea actiona in locul obiectului subclasa . De exemplu , daca avem o operatie care apeleaza metode din obiectele apartinand clasei Integer , folosirea unui obiect din clasa Number (generica) nu va dispune de multe metode dintre cele definite in clasa Integer .
Pentru a folosi obiecte superclasa acolo unde se asteapta obiecte subclasa , trebuie sa facem cast explicit . Nu vom pierde nici o informatie prin cast , dar vom castiga toate metodele si variabilele pe care le defineste subclasa . Pentru a face cast pentru un obiect dintr-o alta clasa folosim aceeasi operatie ca si in cazul tipurilor primitive :

(numeclasa)obiect

numeclasa este numele clasei destinatie , iar obiect este o referinta catre obiectul sursa . Retinem ca operatia de cast creaza o referinta catre vechiul obiect , de tip numeclasa ; vechiul obiect continua sa existe ca si inainte .

In exemplul de mai jos putem vedea un cast de la o instanta a clasei VicePresedinte catre o instanta a clasei Angajat ; VicePresedinte este o subclasa a clasei Angajat , cu informatii suplimentare care definesc faptul ca VicePresedinte are anumite drepturi suplimentare :

```
Angajat ang=new Angajat();
VicePresedinte vip=new VicePresedinte();
```

```
ang=vip;    //nu este nevoie de cast de jos in sus
vip=(VicePresedinte)ang;   // e nevoie de cast explicit
```

In afara de castingul obiectelor catre clase putem de asemenea face castingul obiectelor catre interfete – insa numai daca clasa obiectului respectiv sau una dintre superclasele sale implementeaza de fapt interfata . Castingul unui obiect catre o interfata inseamna ca putem apela una dintre metodele interfetei , chiar daca clasa obiectului nu implementeaza de fapt interfata .

CONVERSIA TIPURILOR PRIMITIVE IN OBIECTE SI INVERS

Unul dintre lucrurile pe care nu le putem face in nici un caz este un cast intre un obiect si un tip de date primitiv sau invers . Tipurile primitive si obiectele sunt lucruri foarte diferite in Java si nu putem face cast automat intre cele doua sau sa le folosim unul in locul celuilalt .

Ca alternativa , pachetul java.lang contine clase care corespund fiecarui tip de date primitive : Integer , Float , Boolean s.a.m.d . Remarcam faptul ca numele acestor clase incep cu litera mare . Java trateaza foarte diferit tipurile de date si versiunile clasa ale acestora ; programele nu se vor compila cu succes daca se folosesc una in locul celeilalte .
Folosind metodele de clasa definite in aceste clase putem crea un obiect pentru fiecare dintre tipurile primitive , folosind operatorul new . Urmatoarea instructiune creaza o instanta a clasei Integer cu valoarea 4455 :

```
Integer numar=new Integer(4455);
```

Odata creat un obiect in acest fel putem sa il folosim ca pe orice alt obiect . Atunci cand dorim sa folosim valoarea din nou ca primitv exista metode speciale , ca mai jos :

```
Int numarNou=numar.intValou();   // returneaza 4455
```

O conversie de care este adesea nevoie in programe este conversia unui sir intr-un tip numeric , cum ar fi un intreg . Acest lucru se poate face cu metoda parseInt a clasei Integer , ca in exemplul de mai jos :

```
String nume=”12000”;
Int numar=Integer.parseInt(nume);
```

COMPARAREA VALORILOR OBIECTELOR SI ALE CLASELOR

In afara de casting exista si alte operatii ce pot fi aplicate asupra obiectelor :

- compararea obiectelor
- gasirea clasei de care apartine un obiect
- testarea daca un obiect reprezinta o instanta a unei clase date

COMPARAREA OBIECTELOR

Aceasta operatie se realizeaza in Java cu ajutorul operatorilor “== “ si “!=” . Atunci cand sunt folositi cu obiecte acesti operatori nu realizeaza ceea ce ne-am astepta . In loc de a verifica daca unul dintre obiecte are aceeasi valoare cu celalalt obiect ele determina daca obiectele sunt de fapt acelasi obiect .
Pentru a compara instantele unei clase si a obtine rezultate folositoare trebuie implementate metode speciale in cadrul clasei si apoi apelate aceste metode .

Un bun exemplu este clasa String . Este posibil sa avem doua obiecte String diferite care sa contina aceeasi valoare . Daca folosim operatorul “==” pentru a compara aceste obiecte ele vor fi considerate diferite . Pentru a vedea daca doua obiecte String au valori identice se foloseste o metoda a clasei , numita equals() . Metoda testeaza fiecare caracter din sir si returneaza valoarea true daca cele doua siruri contin aceleasi valori .
Exemplul de mai jos ilustreaza cele comentate mai sus :

```
class TestEgalitate {
public static void main(String arg[]) {
String str1,str2;
str1=”Test de egalitate intre siruri.”);
str2=str1;

System.out.println(“Sir1 : “+str1);
System.out.println(“Sir2 : “+str2);
System.out.println(“Acelasi obiect ? “+(str1==str2));

Str2=new String(str1);

System.out.println(“Sir1 : “+str1);
System.out.println(“Sir2 : “+str2);
System.out.println(“Acelasi obiect ? “+(str1==str2));
System.out.println(“Aceeasi valoare ? “+str1.equals(str2));
}
}
```

Literalele sir sunt optimizate in Java – daca am crea un sir folosind un literal si apoi folosim un literal cu aceleasi caractere , Java stie suficient pentru a ne oferi acelasi obiect String . Ambele siruri reprezinta acelasi obiect – trebuie sa actionam diferit pentru a crea doua obiecte separate .

DETERMINAREA CLASEI UNUI OBIECT

In exemplul de mai jos este exemplificata aflarea clasei pentru un obiect atribuit variabilei obj :

```
String nume=obj.getClass().getName();
```

Metoda getClass()este definita in clasa Object , deci va fi disponibila pentru toate obiectele . Rezultatul metodei este un obiect Class (unde Class este el insusi o clasa) , care poseda o metoda numita getName() care returneaza un sir reprezentand numele clasei .

Un alt test care poate fi folositor este operatorul instanceof . Acesta are doi operanzi : un obiect in stanga si un nume de clasa in dreapta . Expresia intoarce true sau false in functie daca obiectul este instanta a clasei numite sau a oricarei subclase a ei :

```
“peste_sabie” instanceof String   // va returna valoarea true
Point pt=new Point(10,10);
pt instanceof String  // returneaza valoarea false
```

Operatorul instanceof poate fi folosit si pentru interfete ; daca un obiect implementeaza o interfata , operatorul instanceof cu numele interfetei respective in partea dreapta va intoarce valoarea true .

INSPECTAREA CLASELOR SI A METODELOR PRIN REFLEXIE

Una dintre imbunatatirile aduse limbajului Java dupa versiunea 1.0 a fost introducerea reflexiei , cunoscuta si sub numele de introspectie . Sub orice nume s-ar folosi , reflexia permite unei clase Java – cum sunt toate programele scrise pana acum – sa afle detalii despre orice alta clasa .
Prin reflexie un program Java poate incarca o clasa despre care nu stie nimic , sa afle despre variabilele , metodele si constructorii clasei si apoi sa lucreze cu ele .

Listingul de mai jos prezinta o aplicatie Java care creaza un obiect de tip Random si apoi foloseste reflexia pentru a afisa toate metodele publice care fac parte din clasa :

```
import java.lang.reflect.*;
import java.util.Random;

class AflaMetode {
public staic void main(String[] arg) {
Random rd=new Random();
Class numeClasa=rd.getClass();
Method[] metode=numeClasa.getMethods();
for (int i=0;i<metode.length;i++) {
 System.out.println(“Metoda : “+metode[i]);
}
}
}
```

Folosind reflexia , aplicatia AflaMetode poate afla informatii despre fiecare metoda a clasei Random si despre toate metodele pe care le-a mostenit de la superclasa Random .
Aplicatia AflaMetode poate functiona pentru orice clasa de obiecte .
Reflexia este folosita de obicei de utilitare ca browserele de clasa sau depanatoarele , ca o modalitate de a afla mai multe despre clasa de obiecte analizata sau depanata .

Este de asemenea folosita de JavaBeans , unde posibilitatea unui obiect de a interoga un alt obiect asupra a ceea ce poate sa faca (urmata de o cerere de a efectua ceva) este folositoare in crearea aplicatiilor mai mari .

Pachetul java.lang.reflect contine urmatoarele clase :

- Field – gestioneaza si afla informatii despre variabilele de instanta si de clasa
- Method – gestioneaza metodele de clasa si de instanta
- Constructor – gestioneaza metodele speciale de creare a noilor instante de clasa
- Array – gestioneaza tablouri
- Modifier – decodifica informatii de modificare despre clase , variabile si metode .

In plus exista un numar de noi metode disponibile intr-o clasa de obiecte numita Class , care ajuta la conectarea diferitelor clase de reflexie .
Reflexia reprezinta un element avansat de programare pe care este posibil sa nu il folosim in programe prea des dar care devine foarte importanta atunci cand se lucreaza cu JavaBeans si alte elemente de programare Java avansate .

CREAREA CLASELOR

DEFINIREA CLASELOR

Ca o scurta recapitulare prezentam mai jos o definitie de clasa :

```
class Stiri {
//  corpul clasei
}
```

In mod prestabilit toate clasele mostenesc clasa Object , care este superclasa tuturor claselor din ierarhia Java .
Daca respectiva noastra clasa este o subclasa folosim cuvantul cheie extends pentru a indica superclasa noii clase :

```
class StiriSportive extends Stiri {
// corpul clasei
}
```

CREAREA VARIABILELOR DE CLASA SI DE INSTANTA

Atunci cand cream o clasa care mosteneste o superclasa trebuie precizat comportamentul noii clase , care o face sa difere de superclasa sa . Acest comportament este definit prin specificarea variabilelor si metodelor noii clase .

DEFINIREA VARIABILELOR DE INSTANTA

Variabilele de instanta sunt declarate si definite cam la fel cu variabilele locale . Principala diferenta consta in localizarea acestora in cadrul clasei , in afara metodelor , ca in exemplul de mai jos :

```
class Jabberwock extends Reptile {
String culoare;
String sex;
boolean flamand;
int varsta;
}
```

CONSTANTE

Variabilele sunt folositoare atunci cand avem nevoie sa pastram informatii ce vor fi modificate pe parcursul rularii programului . Daca valoarea nu se schimba niciodata pe parcursul executiei programului putem folosi un tip special de variabila , numit constanta .

O constanta , denumita si variabila constanta , este o variabila a carei valoare nu se modifica niciodata .

Constantele se folosesc pentru definirea valorilor comune pentru toate metodele unui obiect , cu alte cuvinte , pentru denumirea unor valori ce nu se vor schimba in cadrul obiectului . In Java putem crea constante pentru toate tipurile de variabile : de instanta , de clasa sau locale .

Pentru a declara o constanta folosim cuvantul cheie final inainte de declararea variabilei si atribuim valoarea initiala pentru variabila respectiva , ca in exemplele de mai jos :

```
final float pi=3.1415;
final booleandebug=false;
final int telefon=8675309;
```

Constantele pot fi folositoare pentru denumirea diferitelor stari ale unui obiect sau pentru testarea acestor stari . Folosirea constantelor usureaza de cele mai multe ori intelegerea programului .

VARIABILE DE CLASA

Acestea se aplica unei clase in intregul sau (dupa cum am vazut si in cursurile precedente) , nefiind stocate individual in obiectele (instantele) clasei .
Variabilele de clasa folosesc la comunicarea intre diferite obiecte ale aceleiasi clase sau pentru pastrarea unor informatii comune la nivelul intregii clase .

Pentru a declara o variabila de clasa se foloseste cuvantul cheie static , ca mai jos :

```
static int suma;
```

static final int maxObiecte=10;

CREAREA METODELOR

Metodele definesc comportamentul unui obiect – actiunile efectuate la crearea obiectului sau pe parcursul duratei sale de viata .

DEFINIREA METODELOR

Aceasta cuprinde patru parti :

- numele metodei
- tipul obiectului sau tipul de date primitiv returnat de metoda
- o lista de parametri
- corpul metodei

Primele trei parti ale unei definitii de metoda formeaza ceea ce se numeste semnatura metodei .
In definirea metodei mai exista si doua alte parti optionale : un modificator (cuvintele cheie public sau private) si cuvantul cheie throws (care indica exceptiile pe care le poate semnala metoda) .

In alte limbaje numele metodei – numita si functie , subrutina sau procedura – este suficient pentru a o distinge fata de celelalte metode din program .
In Java , in aceeasi clasa putem folosi mai multe metode cu acelasi nume , dar care difera prin valoarea returnata sau prin lista de parametri . Aceasta practica este denumita supraincarcarea metodei (overloading) .
Mai jos putem vedea o definitie generala a unei metode :

```
tipRetur numeMetoda(tip1 argument1 , tip2 argument2 , … ) {
// corpul metodei
}
```

tipRetur poate fi un tip primitiv , un nume de clasa sau cuvantul cheie void , atunci cand metoda nu returneaza o valoare .

In cazul in care metoda returneaza un obiect tablou , trebuie folosite parantezele patrate fie dupa tipRetur fie dupa lista de parametri :

```
int[] creareDomeniu(int inf, int sup) {
//  corpul metodei
}
```

Lista de parametri a metodei este un set de definitii de variabile , separate prin virgula si incadrate intre paranteze rotunde . Acesti parametri devin variabile locale in corpul metodei , primind valori la apelarea metodei .

In afara cazurilor cand este declarata cu tipul de retur void , o metoda returneaza la terminarea sa o valoare de un anumit tip . Aceasta valoare trebuie specificata explicit in punctele de iesire ale metodei , prin cuvantul cheie return .

In listingul de mai jos avem un exemplu de clasa care defineste o metoda care preia doi intregi si creaza un tablou care contine toate numerele intregi aflate intre cele doua limite :

```
class ClasaDomeniu {
int[] creareDomeniu(int inf,int sup) {
int tabl[]=new int[(sup-inf)+1];

for (int i=0;i<tabl.length;i++) {
 tabl[i]=inf++;
}
return tabl;
}

public static void main(String argumente[]) {
int unTablou[];
ClasaDomeniu unDomeniu=new ClasaDomeniu();

unTablou=unDomeniu.creareDomeniu(1,10);
System.out.print("Tabloul : [ ");
for (int  i=0;i<unTablou.length;i++) {
 System.out.println(unTablou[i]+"  ");
}
}
```

Metoda main() a clasei apeleaza metoda creareDomeniu() prin crearea unui domeniu marginit inferior , respectiv superior , de valorile 1 si 10 , dupa care foloseste un ciclu for pentru a afisa valorile noului tablou .

CUVANTUL CHEIE THIS

In corpul unei definitii de metoda putem sa ne referim la obiectul curent – obiectul a carei metoda a fost apelata . Aceasta poate avea scopul de a folosi variabilele de instanta ale obiectului sau de a transmite obiectul curent ca argument unei alte metode .
Pentru a referi obiectul curent in astfel de cazuri se foloseste cuvantul cheie this acolo unde in mod normal s-ar folosi numele unui obiect .
Acest cuvant cheie se refera la obiectul curent si poate fi folosit oriunde poate aparea un obiect : in notatia cu punct , ca argument al unei metode , ca valoare de retur pentru metoda curenta s.a.m.d . Mai jos avem un exemplu de folosire a lui this :

```
t=this.x;    // variabila de instanta x pentru acest obiect
this.resetDate(this);  //  apeleaza metoda resetDate() definita in clasa curenta si
                       //  transmite obiectul curent
return this;   // returneaza obiectul curent
```

In multe cazuri s-ar putea sa nu fie nevoie sa utilizam explicit cuvantul cheie this , deoarece este presupus . De exemplu , putem sa ne referim atat la variabilele de instanta cat si la apelurile de metode definite in clasa curenta prin simpla folosire a numelui lor , deoarece this este implicit folosit in aceste referinte :

```
t=x;    //  variabila de instanta x pentru acest obiect
resetDate(this);          //  apeleaza metoda resetDate() definita in clasa curenta si
                          //  transmite obiectul curent
```

Deoarece this este o referinta a instantei curente a clasei trebuie sa o folosim doar in cadrul corpului unei definitii de metoda de instanta . Metodele de clasa , declarate prin cuvantul cheie static , nu pot folosi this .

DOMENIUL DE VIZIBILITATE AL VARIABILELOR

Domeniul de vizibilitate este acea parte a programului in care o variabila sau o alta informatie poate fi folosita . Daca acea parte a programului care defineste domeniul de vizibilitate al unei variabile isi termina executia variabila nu mai exista .

O variabila cu domeniu de vizibilitate local poate fi folosita doar in cadrul blocului in care a fost definita . Variabilele de instanta si de clasa poseda un domeniu de vizibilitate extins la intreaga clasa , deci ele pot fi accesate de oricare dintre metodele din cadrul clasei .

Atunci cand referim o variabila in cadrul unei metode , Java verifica definitia acesteia mai intai in domeniul de vizibilitate local , dupa aceea in domeniul exterior imediat urmator si , in cele din urma , in domeniul metodei curente . Daca variabila nu este locala Java verifica existenta unei definitii a acesteia ca variabila de instanta sau de clasa in clasa curenta . Daca Java tot nu gaseste definitia variabilei ocauta pe rand in fiecare superclasa .

TRANSMITEREA ARGUMENTELOR CATRE METODE

Atunci cand apelam o metoda cu parametri obiect , obiectele sunt transmise in corpul metodei prin referinta . Orice vom face cu obiectele in cadrul metodei le va afecta direct . Aceste obiecte pot fi tablouri si alte obiecte continute in tablouri . Atunci cand transmitem un tablou ca argument intr-o metoda si continutul lui se modifica , este afectat tabloul original .
Pe de alta parte trebuie sa mentionez ca tipurile de date primitive sunt transmise prin valoare .

METODE DE CLASA

Relatia dintre variabilele de instanta si cele de clasa este comparabila cu diferenta dintre modurile de lucru ale metodelor de instanta si de clasa .

Metodele de clasa sunt disponibile oricarei instante a clasei si pot fi facute disponibile altor clase . In plus , spre deosebire de o metoda de instanta , o clasa nu necesita o instanta a clasei pentru a-I putea fi apelate metodele .

De exemplu, bibliotecile Java contin o clasa denumita Math . Clasa Math defineste un set de operatii matematice pe care le putem folosi in orice program sau pentru diferite tipuri numerice :

```
float radical=Math.sqrt(453.0);
System.out.println("Cel mai mare dintre x si y este : "+Math.max(x,y));
```

Pentru a defini metode de clasa se foloseste cuvantul cheie static , pozitionat in fata definitiei metodei , la fel ca in cazul definirii variabilelor de clasa . De exemplu , metoda de clasa max() , folosita in exemplul precedent , ar putea avea urmatoarea semnatura :

```
static int(int arg1 , int arg2) {
// corpul metodei
}
```

Java contine clase de impachetare (wrapper) pentru fiecare dintre tipurile de baza . Lipsa cuvantului cheie static din fata numelui unei metode face ca aceasta sa fie o metoda de instanta .

CREAREA APLICATIILOR JAVA

Aplicatiile Java sunt programe care ruleaza individual . Aplicatiile difera de appleturile Java care au nevoie pentru rulare de un browser compatibil Java .

O aplicatie Java consta dintr-una sau mai multe clase ce pot avea orice dimensiune dorim . Singurul element de care avem neaparat nevoie pentru a rula o aplicatie Java este o clasa care sa serveasca drept punct de plecare pentru restul programului Java. Clasa de pornire a aplicatiei are nevoie de un singur lucru : o metoda main() . Aceasta metoda este prima apelata .
Semnatura metodei main() arata totdeauna astfel :

```
public static void main (String argumente[]) {
// corpul metodei
}
```

- public – inseamna ca metoda este disponibila altor clase si obiecte . Metoda main() trebuie declarata public .
- static – inseamna ca metoda main() este o metoda de clasa
- void – inseamna ca metoda main() nu returmeaza nici o valoare
- main() preia un parametru care este un tablou de siruri . Acesta se foloseste pentru argumentele programului .

TRASMITEREA DE ARGUMENTE APLICATIILOR JAVA

Pentru a transmite argumente unui program Java acestea trebuie adaugate la executie dupa numele programului :

java ProgramulMeu argument1 argument2

Pentru a transmite argumente care contin spatii acestea trebuie incadrate de ghilimele duble .

Atunci cand o aplicatie este rulata cu argumente , Java le memoreaza sub forma unui tablou de siruri , pe care il transmite metodei main() a aplicatiei . Pentru a putea trata aceste argumente drept altceva decat siruri trebuie mai intai sa le convertim .