Programmazione ad oggetti in Java

E.Mumolo, DEEI

mumolo@units.it

Ricapitolando ...

- OOP è un paradigma di programmazione nel quale i problemi sono modellati come in insieme di oggetti. Gli oggetti:
  - Comunicano mediante invio di messaggi
  - Hanno un loro stato interno
  - Sono caratterizzati dal loro stato, comportamento e identità
- Una classe è un insieme di oggetti con gli stessi comportamenti
- Un oggetto è una istanza di una classe
- Caratteristiche degli oggetti:
  - Incapsulamento
  - Polimorfismo
  - Ereditarietà
  - Aggregazione
  - Identità

Ricapitolando …

- **Livelli di accesso**
  - Privato
    - I campi privati possono essere acceduti solo dai campi che fanno parte della classe
  - Pubblico
    - I campi privati possono essere acceduti da qualunque parte
  - Protetto
    - Nel caso che la definizione di una classe sia derivata da un'altra classe, tutto ciò che è definito come protetto viene ereditato

Ricapitolando …

- Una classe è un insieme di oggetti con gli stessi comportamenti
- Un oggetto è una istanza di una classe
- Caratteristiche degli oggetti:
  - Incapsulamento
  - Polimorfismo
  - Ereditarietà
  - Aggregazione
  - Identità
- Terminologia:
  - Classe derivata: classe ottenuta mediante specializzazione di un'altra classe
  - Classe base: la classe dalla quale una classe è derivata
  - Ereditarietà: una classe derivata eredita da una classe base
- Ereditarietà impropria: quando la classe base ha una capacità che la classe derivata non può soddisfare (esempio: nella derivazione `struzzo`→`uccello` la classe base ha un metodo `vola()`)

Ricapitolando ...

- Terminologia:
  - Classe derivata: classe ottenuta mediante specializzazione di un'altra classe
  - Classe base: la classe dalla quale una classe è derivata
  - Ereditarietà: una classe derivata eredita da una classe base
- Ereditarietà impropria: quando la classe base ha una capacità che la classe derivata non può soddisfare (esempio: nella derivazione `struzzo:uccello` la classe base ha un metodo `vola()`)
- Costruttori/distruttori: inizializzazione var-rilascio spazio
  - Istanziazione oggetto/terminazione oggetto
  - nome uguale alla classe, senza return
  - parametri opzionali, possibili costruttori multipli
  - Costruttore di default / di copia
  - Attenzione: l'inizializzazione del costruttore segue l'ordine di definizione variabili

Ricapitolando …

- Ereditarietà
  - Relazione *has-a* (composizione)
  - Relazione *is-a* (specializzazione/derivazione)
- Esempio:
  - Se il problema si descrive con la frase: *…un veicolo contiene una o più ruote…*

→ Implementare la classe ‘ruota’ e poi la classe ‘veicolo’:

```
class veicolo{
   private:
      Ruota r1,r2,r3;
}
```

  - Se si descrive con la frase: *…un’auto è un veicolo…*

→ Implementare la classe ‘veicolo’ e poi la classe ‘auto’:

```
class auto:public veicolo{
   private:
      ...
}
auto a=new auto(); veicolo v=new veicolo();
v=a; //lecito
a=v; //errato
```

Ricapitolando ...

- Polimorfismo:
  - Capacità degli oggetti di differenti classi legate da ereditarietà di rispondere differentemente alla stessa chiamata
  - Ottenuto mediante overloading delle funzioni e funzioni virtuali
  - Abbinamento statico delle funzioni agli oggetti
  - Statico: legato al tipo del puntatore, non al tipo dell'oggetto puntato
  - 'casting' per indicare esplicitamente il tipo del puntatore
- Funzioni virtuali:
  - Abbinamento dinamico della funzione all'oggetto
  - I costruttori NON possono essere virtuali
  - I distruttori possono invece esserlo: recuperano lo spazio dell'oggetto puntato

Polimorfismo e funzioni virtuali

- Polimorfismo: capacità di rispondere in modo differenziato agli stessi comandi

→ realizzato con overloading delle funzioni e con le funzioni virtuali

- overloading delle funzioni: la scelta della funzione da attivare è effettuata esaminando una lista degli operandi o il tipo di oggetti tramite cui vengono operate le richieste alle operazioni
  - abbinamento statico(static binding): deciso alla compilazione
  - abbinamento dinamico(dynamic binding): deciso in run-time
- overloading quando gli oggetti vengono rappresentati con puntatore:
- i puntatori possono puntare a oggetti di tipo diverso!
- ma l'abbinamento statico si basa sul tipo di puntatore e non sul tipo di oggetto puntato!
- funzione virtuale: funzione il cui abbinamento con l'oggetto è fatto in run-time

Polimorfismo e funzioni virtuali

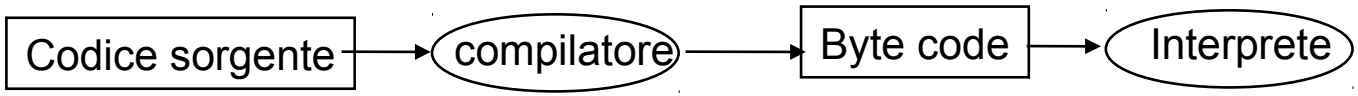
- Una funzione definita virtuale nella classe base in una gerarchia di derivazione, rende virtuali tutte le funzioni con stesso prototipo e componenti la classe derivata
- L'abbinamento dinamico oggetto-funzione con le funzioni virtuali funziona solo se gli oggetti sono gestiti con puntatore. Se l'oggetto gestito con il nome, l'associazione e' statica.
- Tre casi in cui la chiamata di una funzione virtuale è risolta staticamente:
  - quando la chiamata e' effettuata con un oggetto e non con un puntatore
  - quando si usa scope (::) alla classe nella chiamata con puntatore
  - quando una funzione virtuale è chiamata all'interno di costruttore o distruttore
- Costruttori e distruttori virtuali
  - Un costruttore non puo' essere mai dichiarato virtuale (deve essere dichiarato prima)
  - Distruttori possono essere virtuali!

Vantaggi del polimorfismo

- Permette di scrivere codice generico che si adatta automaticamente alle specializzazioni future
- Permette di creare nuovi metodi senza cambiare il resto del codice
- Semplice estensione del codice pre-esistente → riutilizzabilita' del codice
- Attenzione: se l'impostazione e' corretta, si puo' cambiare molto lavorando poco…
- …se si modifica troppo il codice originale, l'impostazione e' sbagliata!
- Modifiche concentrate → migliore manutenzione
- Attenzione: i vantaggi si pagano con una certa perdita di efficienza!

Cos'è Java

- Linguaggio di programmazione definito dalla Sun
- Obiettivo: sviluppo di applicazioni sicure, efficienti, robuste, su piattaforme multiple, in reti eterogenee e distribuite
- Linguaggio semplice e orientato agli oggetti
- Interpretato: produce codice intermedio ("byte-code") per una "Java Virtual machine":



- Portabile su diverse piattaforme
- Architetturalmente neutro:
  - Byte-code indipendente dalla architettura hardware
  - Il byte-code puo' eseguire su un sistema che abbia un ambiente run-time Java
- Robustezza: controlli estesi in compilazione e run-time

Il byte-code

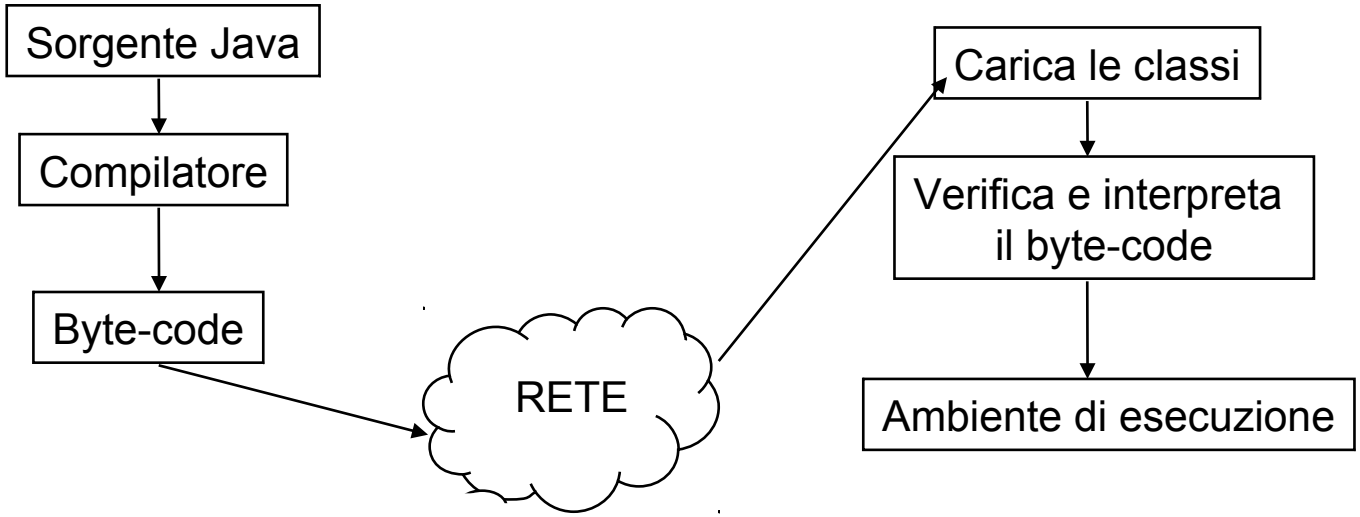
```
Sorgente in Java
{
     int i;
     int a;
     a=0;
     for(i=0;i<5;i++){   a += i; }
}
Traduzione in byte-code:

Method void main(java.lang.String[])
   0 iconst_0  // push costante 0
   1 istore_2  // memorizza in var.locale 2 (a)
   2 iconst_0  // push costante 0
   3 istore_1  // memorizza in var.locale 1 (i)
   4 goto 14        // vai alla riga 14
   7 iload_2        // prendi a (push)
   8 iload_1        // prendi i (push)
   9 iadd      // a+i
  10 istore_2  // a = a+i
  11 iinc 1 1  // i=i+1
  14 iload_1        // prendi i (push)
  15 iconst_5  // push costante 5
  16 if_icmplt 7    // salta a 7 se i<5
  19 return         // esci
```

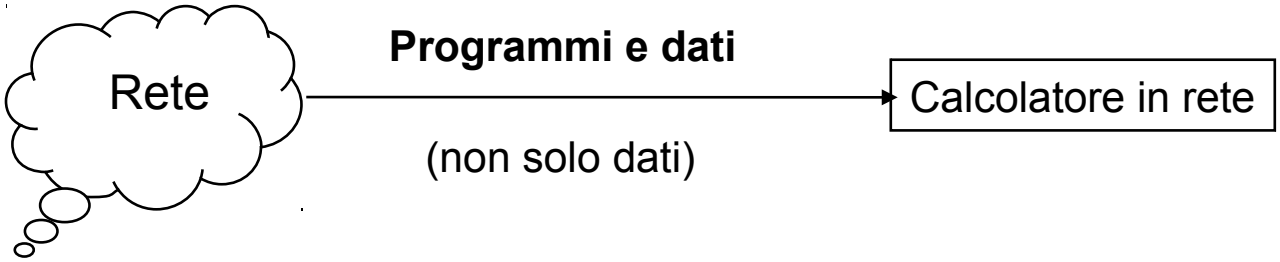
Cos'è Java (cont.)

- Distribuito:
  - Pensato per essere eseguito in rete
  - Funzioni di rete di basso e alto livello
  - Rete accessibile come i file locali
- Sicurezza:
  - Alcune caratteristiche del byte-code sono verificate prima della interpretazione
  - salta molti controlli fatti normalmente a run-time → efficienza
  - Indirizzamenti controllati dall'interprete
- Possibilita' di caricamento dinamico delle classi dalla rete
- Concorrente (threaded)
  - Applicazioni concorrenti piu' facili da scrivere
  - Migliore interazione

Java e la rete



Network computing



Java White Paper (Sun, 1995)

- Primo scopo: eliminare la ridondanza del C e C++:
  - Caratteristiche sovrapposte, troppi modi per fare la stessa cosa
  - C++ aggiungendo classi a C, aumenta la ridondanza
- Principi guida di un buon linguaggio: semplicita', unicita', consistenza
- Unicita': fornire un buon modo per esprimere ogni operazione che interessa, evitare che ce ne siano due
- Altri linguaggi OO: Eiffel, Smalltalk, Ada
- Applicazioni o Applets?
  - Applet: codice creato per far parte di un documento
  - Applicazioni: compilatore scaricabile da rete (esempio, ftp.sun.com/pub)
- Alcuni strumenti Java:
  - javac <file.java> → compila la classe in byte code
  - java <nome_della_classe_main> → java virtual machine: interpreta byte code
  - appletviewer <url|file> → visualizza un applet
  - jdb <file[.class]> → java debugger
  - javap <file[.class]> → reverse eng., disassembla etc.
  - jar → Java Archive: jar cf *jar-file input-file(s)*

```
#include <stdio.h>     // semplice esempio di programmazione C++  bubble sort
#include <iostream.h>
#include <stdlib.h>
#include <string.h>
struct sistema {   float *aa;     int   ord; };
const int ord=10, ord1=3;  // variabili non modificabili!
int bubble(float *, int N=ord);
void scambia(float &, float &); // passaggio per riferimento
void stampa(int);  void stampa(float *a, int n=ord);
main()
{  sistema SS[ord1];        // definisce l'array SS di ord1 strutture 'sistema'
   int nl;
   for(short i=0;i<ord1;i++){    //carica le strutture
      cout<<endl<<"carica"<<i;         SS[i].ord=5+random(10);          SS[i].aa = new   float[SS[i].ord]
      //alloca l'array in memoria libera
      for(short j=0; j<SS[i].ord;j++) {SS[i].aa[j]=float(random(100)); cout<<SS[i].aa[j]<<" "; }
   }
   for(short i=0;i<ord1;i++){
      printf("\n\nArray originale %d:\n", i);  stampa(SS[i].aa,SS[i].ord);
nl=bubble(SS[i].aa,SS[i].ord);         //argomento di default
      stampa(nl);                  //overloading di funzioni e valori default
      stampa(SS[i].aa,SS[i].ord);      //overloading di funzioni e valori default
   }
}
```

```
int bubble(float *A, int N)
{
   char *flag="notsorted"; int nloop=0;
   while(!strcmp(flag, "notsorted")){
       flag="sorted"; nloop++;
       for(short i=0;i<N-1;i++)
            if(*(A+i) > *(A+i+1)){           //A[i] e' *(A+i)
                 scambia(*(A+i),*(A+i+1));   //passa per riferimento!!
                       flag="notsorted";
            }
   }
   return nloop;
}
void scambia(float &a, float &b)  // il compilatore passa l'indirizzo delle var.
{  float temp=a;  a=b; b=temp;  }

void stampa(int n)
{   printf("\nArray ordinato (nr. cicli=%d):\n", n);  }

void stampa(float *a, int n)
{
     for(short j=0;j<n;j++)  cout << a[j] << " "; cout ; // definizione short
}
```

```
#include <stdio.h>    //stesso esempio di programmazione C++ rimuovendo i puntatori espliciti
#include <iostream.h>
#include <stdlib.h>
#include <string.h>
struct sistema {     float *aa;     int  ord; };
const int ord=10, ord1=3;  // variabili non modificabili!
int bubble(float *, int N=ord);  //argomento di default
void scambia(float *, int);                  // passaggio per riferimento
void stampa(int); void stampa(float *a, int n=ord);
main()
{  sistema SS[ord1];  // definisce l'array SS di ord1 strutture 'sistema'
   int nl;
   for(short i=0;i<ord1;i++){    //carica le strutture
           cout<<endl<<"carica"<<i;         SS[i].ord=5+random(10);   SS[i].aa = new float[SS[i].ord];
           //alloca l'array nella memoria libera
           for(short j=0; j<SS[i].ord;j++) {SS[i].aa[j]=float(random(100)); cout<<SS[i].aa[j]<<" "; }
   }
   for(short i=0;i<ord1;i++){
           printf("\n\nArray originale %d:\n", i);  stampa(SS[i].aa,SS[i].ord);
nl=bubble(SS[i].aa,SS[i].ord);   //argomento di default
           stampa(nl);                                  //overloading di funzioni e valori default
           stampa(SS[i].aa,SS[i].ord);                  //overloading di funzioni e valori default
   }
}
```

```
int bubble(float A[], int N)
{
   char *flag="notsorted";
   int nloop=0;
   while(!strcmp(flag, "notsorted")){
           flag="sorted"; nloop++;
      for(short i=0;i<N-1;i++)
                if(A[i] > A[i+1]){
                scambia(A,i);
                flag="notsorted";
           }
   }
   return nloop;
}

void scambia(float a[], int i)  // il compilatore passa l'indirizzo delle var.
{  float temp=a[i];  a[i]=a[i+1]; a[i+1]=temp;  }

void stampa(int n)
{  printf("\nArray ordinato (nr. cicli=%d):\n", n);  }

void stampa(float *a, int n)
{     for(short j=0;j<n;j++)  cout << a[j] << " "; cout ; // definizione short }
```

```
//stesso programma, sort, scritto in java
import java.io.*;
import java.util.Random;
class sort{
      static int bubble(float A[], int N)
      {
            String flag="notsorted"; int nloop=0;
            while(flag!="sorted"){
             flag="sorted"; nloop++;
              for(short i=0;i<N-1;i++)
                  if(A[i] > A[i+1]){
                        scambia(A,i);
                        flag="notsorted";
                  }
         }
         return nloop;
      }

      static void scambia(float a[], int i)
      {
            float temp=a[i];
            a[i]=a[i+1]; a[i+1]=temp;
      }
```

```
        static void stampa(int n)
        {  System.out.print("Array ordinato (nr. cicli=" + n +")"); System.out.println();  }

        static void stampa(float a[], int n)
        {  for(short j=0;j<n;j++)  System.out.print(a[j] + " "); System.out.println();          }

        public static void main(String argv[])
        {
                int ord1=2;  int ord[] = new int[ord1]; float SS[][] = new float[ord1][20];
                int nl;  Random r = new Random();
                for(short i=0;i<ord1;i++){       //carica le strutture
                  System.out.println();  System.out.print("carica " + i +": ");
                  ord[i]=1+(int)(r.nextFloat()*10);
                  for(short j=0; j<ord[i];j++)
                       {SS[i][j]=(int)(r.nextFloat()*100); System.out.print(SS[i][j] + " ");}
                }
                for(short i=0;i<ord1;i++){
                  System.out.print("\nArray originale " + i +"= "); System.out.println();
                  stampa(SS[i],ord[i]);
                  nl=bubble(SS[i],ord[i]);   stampa(nl);  stampa(SS[i],ord[i]);
                }
        }  // fine del main
}  //fine della classe
```

Osservazioni

- Per gran parte e' codice C++
- Non ci sono variabili globali
- Case sensitive
- Un programma java e' sempre un insieme di classi: nel programma, la classe e'**sort** i cui *metodi* sono **bubble, scambia, stampa, main**
- Il main e' chiamato differentemente dal C++:

C++	**Java**
void main(int argc, char * argv[])	public static main(String argv[])
Esempio:	
prog 5 → argc=2, argv[0]="prog", argv[1]=5	argv[0]=5, argv.lenght=1

- L'input/optput e' diverso dal C++: System.out(xx) = cout << xx
- #include e' sostituito da import: **java.util.random** e' la classe Random dal package util
- Cosi', Random r=new Random() crea una istanza della classe Random e r.nextFloat() genera un numero random tra 0 e 1 attivando il metodo nextFloat
- Assenza di *struct* e *union* in Java: in C++ e' una parte dipendente dalla macchina (allineamenti e dimensioni)

Costanti, variabili, identificatori

- Non esiste ne' #define ne' const
- Definizione di una costante:

```
final    tipo_costante    nome_costante = valore;
```

- Definizione di una variabile:

  tipo_variabile nome_variabile [=valore][, nome_variabile [=valore] …]

- Convenzioni per i nomi identificatori:
  - Case misto per i nomi delle classi (es.: MiaClasse)
  - I nomi delle costanti sono in lettere capitali (es.: PI_GRECO)
  - Altri nomi (funzioni, variabili, parole riservate) sono in minuscolo o con case misto, partendo pero' con una lettera minuscola
- Visibilita' delle variabili: all'interno di un blocco { }. Se i blocchi sono annidati, non e' possibile usare lo stesso nome

Tipi di dati

- Tipi semplici: Interi, virgola mobile, caratteri, logici
- Interi: tutti con segno (non esiste l'*unsigned*)
  - **Byte** → 8 bit con segno
  - **Short** → 16 bit con segno, big endian, high byte first
  - **Int** → 32 bit con segno
  - **Long** → 64 bit con segno
- Virgola mobile: formato IEEE-754
  - **Float** → 32 bit
  - **Double** → 64 bit
- Caratteri:
  - **Char** → 16 bit senza segno, codifica Unicode
- Logici:
  - **Boolean** → due valori possibili, *true* e *false*
- Casting fra tipi primitivi, promozione automatica

Casting

- Casting di tipi primitivi:
  - Perdita di precisione
  - Boolean non puo' essere convertito in nessun tipo primitivo
- Casting di oggetti:
  - La classe di partenza e di arrivo devono essere in relazione ereditaria
  - Casting in superclasse fa' perdere i dati della sottoclasse
- Casting tra tipi primitivi e oggetti:
  - Il package *java.lang* comprende classi speciali: **Integer, Float, Boolean, …**

per rappresentare oggetti equivalenti ai tipi primitivi

```
Integer o = new Iteger(22);      // tratta un valore come un oggetto
```

- **Casting tra oggetti a dati primitivi: mediante metodi speciali**

```
Int n = o.intValue(); //restituisce 22
```

Array

- Array o matrici: sono oggetti! Una variabile array punta a tutto l'oggetto. Campo dell'oggetto: length
- Array monodimensionali:
  - int buf[]; → esempio di dichiarazione di array di interi
  - buf = new int[10]; → esempio di allocazione
  - String nome[]; → esempio di dichiarazione di stringa

```
class  buffer{
  public static void main(String args[]) {
        int buf[] = new int[10];//buf e' un puntatore ad un oggetto array
        float arr[] = new float[10];
        buf[0]=1;   int buf[] = new int[5]; // il vecchio array e' eliminato (garbage)
        arr[5] = 3.14; float[] new; //altro modo per dichiarare un array
       new=arr; //arr e new puntano allo stesso oggetto array
       System.out.println("Valore di buf[0] = " + buf[0]+"nr.elementi "+buf.length);
  }
}
```

- Array multidimensionali:
  - int buf[][] = new int[10][5]; //esempio di matrice di interi

Array (cont.)

- I componenti di un array;
  - Sono tutti dello stesso tipo
  - Possono essere di tipo primitivo o riferimenti
  - Sono indicizzati con *int (controllo a run-time)*
- Dichiarazione:

`int[] a;` oppure `int a[];`

- Creazione:

```
a=new int[3];
```

- Uso:

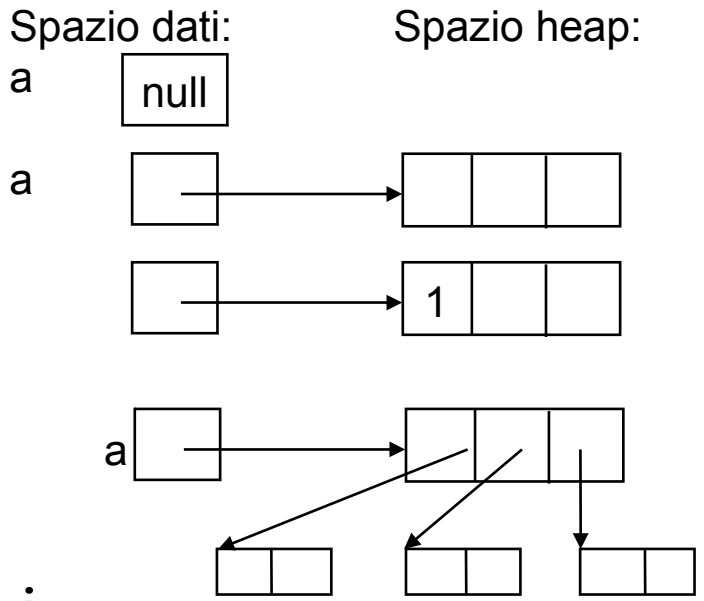
```
a[0]=1;
```

- Array di array:

```
int a[][];
a=new int[3][2];
```

- Array di oggetti:

```
class B{…} → B a[];     a=new B[3];
```



Packages

- Definizione di classi: simile a C++. Esistono classi predefinite, raggruppate in *packages*
- Ogni classe va' in un file separato: il nome del file sorgente deve essere esattamente uguale al nome della classe, con estensione ".java"
- La *import* (es. import java.io.*) importa tutte le classi di un *package*
- Ogni programma inizia con *import java.lang.*;* (inserimento predefinito)
- La *import* puo' essere omessa, indicando esplicitamente quale classe si vuole utilizzare

esempio: import java.util.Random; → Random r = new Random();

oppure: java.util.Random r; → r = new java.util.Random();

Packages

- Attenzione: la import non importa nulla. E' solo una abbreviazione!
- Il compilatore javac importa la classe quando richiesta, eventualmente compilando il file *.java*.

```
class Messaggi {
     void msg1(){ System.out.println("Primo messaggio");
     void msg2(){System.out.println("Secondo messaggio");
}
class Test {
     public static void main(String[] args){
          Messaggi m1 = new Messaggi;    Messaggi m2 = new Messaggi;
          m1.msg1(); m2.msg2()
 }
```

- Le due classi vanno in due file, chiamati Messaggi.java e Test.java. Basta scrivere: javac test.java

Puntatori

- Gli oggetti in Java possono solo essere acceduti tramite puntatori
- Una variabile puo' contenere valori primitivi o riferimento a oggetti
- Una variabile non puo' contenere un oggetto
- Non esistono gli operatori "*****", "**&**", "**->**" ma solo l'operatore punto "**.**"
- I membri della classe sono per default *friend* di altre classi del package

Puntatori

Esempio C++

```
#include <iostream.h>
#include <math.h>
class punto {
          friend class stampa;
          float x, y;   //private
          public:
               punto(float a, float b){x=a; y=b;};
              float distanza(punto *a){
                    float t=pow(x-a->x,2)+pow(y-a->y,2);
                   return sqrt(t);
               }
};
class stampa{
          public:
              void out(float a, float b, float c, float d)
              {
                     punto *p,*q=new punto(a,b), *r;
                     cout << q->x << " " << q->y << endl;
                  p->x=c; p->y=d;
                     cout << p->x << " " << p->y << endl;
                cout << q->distanza(p);
      }
};
main()
{          stampa    s;
           s.out(1,2,5,5);
}
```

Esempio Java

```
class punto {                                   //file punto.java
          float x, y;
          punto(float a, float b){x=a; y=b;};
          float distanza(punto a){
             float t=(float)(Math.pow(x-a.x,2)+Math.pow(y-a.y,2));
                return (float)Math.sqrt(t);
           }
};

import java.io.*;
class dots{             //file dots.java
          static void out(float a, float b, float c, float d)
          {
               punto p=new punto(0,0),q=new punto(a,b), r;
       System.out.println(q.x +" "+ q.y);
       p.x=c; p.y=d;
                                        System.out.println(p.x + " " +
p.y);
               System.out.println("distanza="+ q.distanza(p));
           }

           public static void main(String argv[])
          {
               out(1,2,5,5);
              System.exit(0);
           }
}
```

Garbage collection

- Come in C++, un oggetto e' creato con *new*
- Delete in C++ puo' introdurre errori: cancellazione prematura, cancellazione tardiva

- In Java non esiste l'operatore *delete*
- Cancellazione automatica di oggetti quando non ci sono piu' riferimenti ad essi

Incapsulamento

- Membri privati: accesso consentito solo dall'interno dei metodi della classe
- Membri protetti: accesso consentito alla classe e alle classi derivate
- Membri pubblici: accesso consentitoa qualsiasi funzione
- Sintassi C++ Sintassi Java

```
class Mia{
   Private:
        int i;
        double d;
   Public:
        int j;
        void funzione() { … }
}
```

```
class Mia{
        private     int i;
        public    int j;
        private      double d;
        public     void funzione() {…}
}
```

- Di default, i membri sono visibili pubblicamente
- Membri statici: esiste solo una copia condivisa da tutte le instanze

```
//file Veicolo.java
class Veicolo
{
        private int VelocitaMassima;
        private int NumeroPosti;
        public Veicolo(int VM, int NP) // costruttore
        { VelocitaMassima = VM; NumeroPosti = NP; }
}
```

```
//file mioveicolo.java
public class mioveicolo
{
        public static void main(String args[])
        {
                Veicolo MiaMacchina= new Veicolo(150, 5);
                System.out.println("Creato un oggetto di classe Veicolo");
        }
}
```

```
//file Veicolo.java
class Veicolo
{
     private int VelocitaMassima; // variabili private
     private int NumeroPosti; // semantica per valore
     public Veicolo(int VM, int NP) // costruttore
     { VelocitaMassima = VM; NumeroPosti = NP; }
     public int getVelocitaMax() // metodi pubblici
     { return VelocitaMassima; }
     public int getNumeroPosti()
     { return NumeroPosti; }
}
```

```
//file mioveicolo2.java
public class mioveicolo2
{
     public static void main(String args[])
     {
          int Intero;
          Veicolo MiaMacchina = new Veicolo(150, 5);
          System.out.print("La mia macchina ha ");
          System.out.print(MiaMacchina.getNumeroPosti()+" posti");
          System.out.print(" e raggiunge la velocita' di ");
          System.out.println(MiaMacchina.getVelocitaMax() + " km/h.");
          // Intero = MiaMacchina.NumeroPosti;
          // il compilatore da' errore
     }
}
```

Stringhe

- Il tipo **String** crea un array di char
- L'operatore + e' sovrapposto per introdurre la concatenazione
- La concatenazione puoì essere fatta con qualsiasi cosa: conversione automatica in stringa
- Operazioni piu' comuni (String s="blabla")
  - s.charAt(n) → ritorna il carattere alla posizione n della stringa s
  - s.substring(n) → ritorna la sottostringa di s dalla posizione n alla fine
  - s.compareTo(str) → <, >, == 0 se "s" precede, segue, e' uguale a "str"
  - s.indexOf('c') → ritorna il primo indice del carattere 'c' in "s"
  - s.lastIndexOf('c') → ritorna l'ultimo indice del carattere 'c' in "s"
  - s.endsWith("str") → ritorna vero o falso

Costruttori e sovrapposizione

- I costruttori in Java hanno lo stesso significato del C++
- Differenza: () anche se non ho argomenti
- Differenza: i costruttori devono essere scritti in linea
- Sovrapposizione (overloading): piu' costruttori, con diversi argomenti

```
class punto {           //file punto.java
              float x, y;
              punto(float a, float b){ x=a; y=b; //stesso che this.x=a; this.y=b;};
              punto(float a){ x=a; y=0; };
              punto(){ x=0; y=0;  };
};

import java.io.*;
class dots{ //file dots.java
            public static void main(String argv[]) {
            punto p1=new punto(1,2);
            punto p2=new punto; //errore
            punto p3=new punto();
            System.exit(0);
           }
}
```

Ereditarieta’

- Due tipi di ereditarieta’: di **metodo** e di **interfaccia**
- **Ereditarieta’ di metodo**:
  - eredità singola, parola chiave: *extends*
  - L’istruzione **super** fa’ riferimento alla classe del padre)
  - La classe figlia eredita:
    - variabili e metodi della classe padre definiti *public*
    - variabili e metodi della classe padre definiti *protected*
    - variabili e metodi della classe padre senza attributo se appartiene allo stesso package
    - variabili e metodi che la classe padre ha ereditato dagli «avi» della gerarchia
  - Riutilizzare componenti già definiti, specializzandoli

```
class Base{          // superclasse
   void fz1(){…}
   void fz2(){…}
};
Class Derivata extends Base{      //sottoclasse di Base
      void fz2(){…} //sostituisce fx2() di Base (overriding)
      void fz3(){             //la classe Derivata ha tre metodi, fz1, fz2, f3
            super.fz2();    //si riferisce al padre!
            ...
      }
};
```

- **Ereditarieta’ di interfaccia**
  - eredità multipla, parola chiave: *implements*

```
//file Veicolo.java
class Veicolo           //uguale a quello precedente!
{ … }
 //file VeicoloTerrestre.java
 class VeicoloTerrestre extends Veicolo
 {     private int NumeroRuote;
       public VeicoloTerrestre(int VM, int NP, int NR) // costruttore
       {     super(VM,NP); // chiama il costruttore del padre
             NumeroRuote = NR;
       }
       public int getNumeroRuote()
       { return NumeroRuote; }
 }

//file VeicoloMarino.java
class VeicoloMarino extends Veicolo
{     private long Stazza;
      public VeicoloMarino(int VM, int NP, long S)
      {     super(VM,NP);  Stazza = S;  }
      public long getStazza()
      { return Stazza; }
}

//file mioveicolo3.java
public class mioveicolo3
{
      public static void main(String args[])
      {
            VeicoloTerrestre MiaMacchina = new VeicoloTerrestre(100, 5, 4);
            VeicoloMarino MiaNavenew VeicoloMarino(5, 10, 10);
            System.out.print("La mia macchina ha ");
                        System.out.print(MiaMacchina.getNumeroPosti() + " posti, ");
            System.out.println(MiaMacchina.getNumeroRuote() + " ruote");
                        System.out.println(" e una velocita' di "+MiaMacchina.getVelocitaMax()+"KM/h");
                        System.out.print("La mia nave ha ");
            System.out.print(MiaNave.getNumeroPosti() + " posti, ");
            System.out.println("una stazza di " + MiaNave.getStazza());
                        System.out.println(" e una velocita' di "+MiaNave.getVelocitaMax()+"nodi/h");
      }
}
```

Interfacce

- Interfaccia: struttura sintattica con nome
- Specifica i nomi, gli argomenti e i tipi di ritorno dei metodi di una classe non ancora implementata

```
interface Punto {
        punto(float a, float b);
        float distanza(punto a);
}
```

- Una interfaccia puo' essere implementata da una o piu' classi

```
class MioPunto implements Punto {
        float x, y;
        punto(float a, float b) ){x=a; y=b;};
        float distanza(punto a) { float t=(float)(Math.pow(x-a.x,2)+Math.pow(y-a.y,2));
                return (float)Math.sqrt(t);
};
class TuoPunto implements Punto {
        float x, y;
        punto(float a, float b) ){x=2*a; y=2*b;};
        float distanza(punto a) { float t=(float)(Math.abs(x-a.x)+Math.abs(y-a.y));
                return (float)Math.sqrt(t)}
};
```

- Una classe puo' implementare piu' di una interfaccia

  → ereditarieta' multipla

```
class MiaClasse implements A,B{…}
```

Costruttori e sovrapposizione

- I costruttori in Java hanno lo stesso significato del C++
- Differenza: () anche se non ho argomenti
- Differenza: i costruttori devono essere scritti in linea
- Sovrapposizione (overloading): piu' costruttori, con diversi argomenti

```
class punto {                    //file punto.java
        float x, y;
        punto(float a, float b){ x=a; y=b; //stesso che this.x=a; this.y=b;};
        punto(float a){ x=a; y=0; };
        punto(){ x=0; y=0;  };
};

import java.io.*;
class dots{              //file dots.java
      public static void main(String argv[]) {
            punto p1=new punto(1,2);
            punto p2=new punto; //errore
            punto p3=new punto();
            System.exit(0);
          }
}
```

Eccezioni

- Java non crea *core files*
- Errori di run-time fanno scattare una eccezione
- Di default, una eccezione causa la terminazione di un programma
- Le eccezioni possono essere catturate con le istruzioni *catch-try*

```
try{
...//istruzioni, chiamate a funzioni etc. da osservare
}
catch(TipoDiEccezione e){  //cattura l'eccezione
...//descrive cosa fare quando
}
```

- Eccezioni built-in: ArithmeticException, NullPointerException, ClassCastException, IOException,

  ArrayIndexOutOfBoundsException, NegativeArraySizeException, OutOfMemoryException, …
- Una eccezione e' un oggetto, che viene ereditato da altri oggetti
- L'istruzione **catch** cattura una eccezione e le sue derivate
- Le eccezioni possono essere definite dall'utente con *throw* (lancia eccezioni)
  - Definizione della classe NuovaEccezione
  - throw new NuovaEccezione();
  - try {…} catch (NuovaEccezione e) { … }

Il tipo “Vector”

- Un Vector e’ un array dinamico
- Gli elementi di un Vectorsono oggetti, non valori
- La classe Vector e’ realizzata come un array ordinario

```
import java.util.*;
import java.io.*;
class vect{
            public static void main(String argv[])
            {
                int n;   Integer o;
                Vector v=new Vector();
                System.out.println("carico il Vector di interi");
                for(int i=0; i<10; i++) v.addElement(new Integer(i*2));
                System.out.println("visualizzo Vector:");
                for(int i=0; i<10; i++) {
                        o=(Integer) v.elementAt(i);
                        n=(int)o.intValue();
                        System.out.println("elemento "+i+"="+n);
                }
                System.exit(0);
            }
}
```

Costruzione di liste

```
public class ListNode {
          public Object   element ;
          public ListNode next ;
          /** Costruttore nodo isolato       */
          public ListNode ( Object element ) {
           this( element, null ) ;
          }

           /** Costruttore nodo per liste unidirezionali, dove element e' non  null    */
          public ListNode ( Object element, ListNode next ) {
               if ( element == null )
                      throw new IllegalArgumentException ( ) ;
               this.element  = element ;
               this.next     = next ;
          }
}
```

Polimorfismo

- Gli oggetti della classe derivata hanno la stessa interfaccia della classe base, o un suo sovrainsieme
- Gli oggetti della classe derivata possono essere visti come oggetti ‘estesi’ della classe base
- Variabili del tipo della classe base possono contenere riferimenti a oggetti della classe derivata
- Quando viene inviato un messaggio a un oggetto, la scelta dell’implementazione del metodo è effettuata dinamicamente sulla base della classe effettiva, non del tipo della variabile (polimorfismo)

```
//file Animale.java
public class Animale
  {   public void verso()
                {  System.out.println("Che animale sono?");  }
  }

//file Cane.java
public class Cane extends Animale
 {    public void verso()
                {  System.out.println("Sono un cane: Bau bau!");       }
  }

//file Gatto.java
public class Gatto extends Animale
 {    public void verso()
                {  System.out.println("Sono un gatto: Miao!");      }
  }

 //file VersiAnimali.java
 public class VersiAnimali
   {  public static void main(String args[])
         {  Animale t;     Animale a=new Animale();  Animale b=new Cane();  Animale c=new Gatto();
            t=a;    t.verso();
            t=b;    t.verso();
            t=c;    t.verso();
         }
   }
```

```
//file PoliVeicolo.java
class PoliVeicolo
{     private int VelocitaMassima; private int NumeroPosti;
            public PoliVeicolo(int VM, int NP) { VelocitaMassima = VM; NumeroPosti = NP; }
      public int getVelocitaMax() { return VelocitaMassima; }
      public int getNumeroPosti() { return NumeroPosti; }
            public String stampa() {return "Veicolo con "+NumeroPosti+"posti e velocita' massima di“
      +VelocitaMassima+"Km/h"; };
}

//file Ferrari.java
class Ferrari extends PoliVeicolo
{     public Ferrari(int VM, int NP) // costruttore
      { super(VM, NP); }
      public String stampa() //ridefinisce il metodo
      { return "Sono una Ferrai, ho " + getNumeroPosti() + " posti"+
            "e vado a” + getVelocitaMax()+"Km/h"; }
}

//file Fiat500.java
class Fiat500 extends PoliVeicolo
{     public Fiat500(int VM,int NP) // costruttore
      { super(VM,NP); }
      public String stampa() //ridefinisce il metodo
            { return "Sono una Fiat500, ho " + getNumeroPosti() + " posti" + "e posso andare a”+
            getVelocitaMax()+"Km/h"; }
}


//file Veicoli.java
public class Veicoli
{     public static void main(String args[])
      {                PoliVeicolo v=new PoliVeicolo(0,0); // istanza della classe padre
                        Ferrari v1 = new Ferrari(300,1);  Fiat500 v2 = new Fiat500(100,4);
            System.out.println(v.stampa());
            v = v1; System.out.println(v.stampa()); v = v2; System.out.println(v.stampa());
      }
}
```

Input/output in Java

- Flusso di dati sorgente-destinazione
- Potenza della astrazione: sorgente/destinazione possono essere qualsiasi (file, socket, tastiera, monitor etc)
- In Java si legge e si scrive su un oggetto **Stream**
- Classi del package java.io:
  - Metodi di lettura: specificano come argomento la sorgente
  - Metodi di lettura: specificano come argomento la destinazione
  - Metodi di elaborazione: leggono i dati dal flusso indicato nel 1o argomento e scrivono i dati elaborati sul 2o argomento
- I metodi di elaborazione non conoscono la sorgente o la distinazione: dettagli nascosti
- Gerarchia di classi:
  - Reader: input caratteri
  - Writer: output caratteri
  - InputStream: input byte
  - OutputStream: output byte
  - Buffered: associano un buffer allo stream di I-O. Il buffer:
    - Permette di trasferire piu' di un byte alla volta
    - Permette di inserire metodi di *skip, mark* e *reset*

Input/output in Java

- Legge dallo standard input e scrive sullo standard output

```
import java.io.*;  //in questo caso considero le eccezioni
public class Copia1 {
    public static void main(String[] args)  {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        for(;;){
            try{
                String line=in.readLine();
                System.out.println(line);
             } catch(IOException e){ System.out.println("errore di IO : " + e);}
        }
    }
}
```

- Legge dallo standard input e scrive sullo standard output

```
import java.io.*; // in questo caso trascuro le eccezioni
public class Copia {
    public static void main(String[] args) throws IOException {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        for(;;){
                String line=in.readLine();
                System.out.println(line);
        }
    }
}
```

Input/output in Java

- Copia il file README nel file READout per interi

```
import java.io.*;

public class prova {
   public static void main(String[] args) throws IOException {

      FileInputStream f = new FileInputStream("README");
      FileOutputStream f1 = new FileOutputStream("READout");

       while(f.available()!=0){
            int line=f.read();
                 f1.write(line);
       }
   }
}
```

Input/output in Java

- La classe FILE: descrive le proprieta' del file. Alcuni metodi compresi:

```
public java.lang.String getName();
public java.lang.String getPath();
public java.lang.String getAbsolutePath();
public java.lang.String getCanonicalPath();
public java.lang.String getParent();
public boolean exists();
public boolean canWrite();
public boolean canRead();
public boolean isFile();
public boolean isDirectory();
public long lastModified();
public long length();
public boolean mkdir();
public boolean renameTo(java.io.File);
public boolean mkdirs();
public java.lang.String list()[];
public boolean delete();
```

Input/output in Java

- Esempio: lista di alcune caratteristiche del file README

```
import java.io.*;

public class prova1 {
   public static void main(String[] args) throws IOException {

      File f = new File("README");

       System.out.println("Nome:" + f.getName());
       System.out.println("Path:" + f.getPath());
       System.out.println("Dimensione:" + f.length());
   }
}
```