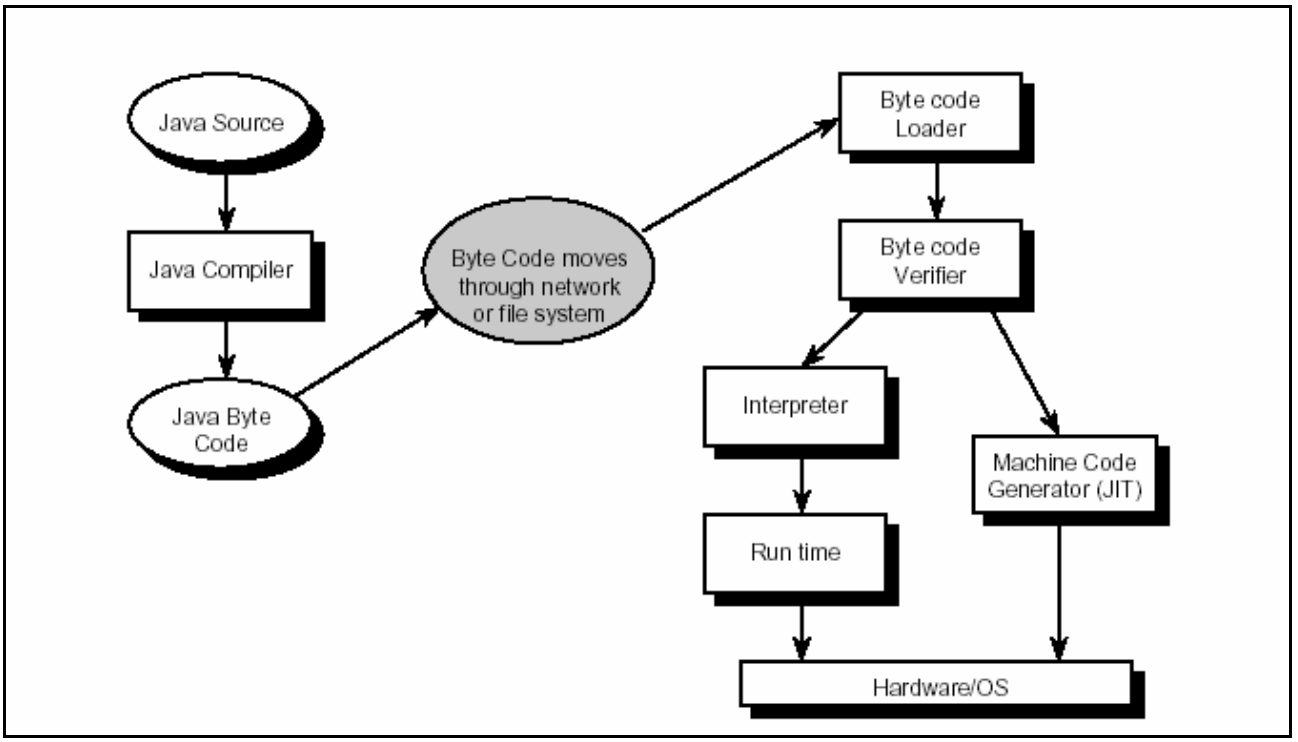
L’AMBIENTE JAVA

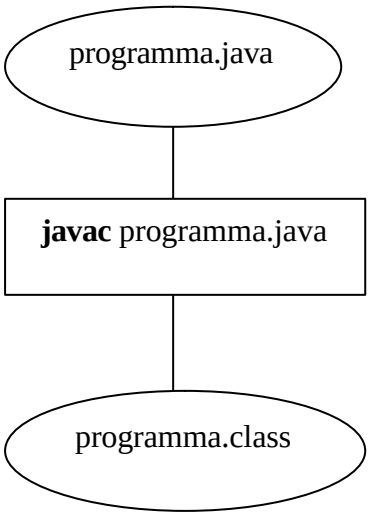


Java è in parte compilato ed in parte interpretato.

All’inizio il sorgente viene compilato del **COMPILATORE** tramite il comando:

javac programma.class,

che genera un file chiamato **programma.class** contenente il ***byte code***. E’ la traduzione del programma in un linguaggio intermedio



Il byte code può essere interpretato su qualsiasi piattaforma che supporta Java tramite il comando:

java programma

La caratteristica è più importante per sistemi distribuiti, che tipicamente hanno macchine eterogenee.

Java supporta inoltre la **mobilità del codice**, posso cioè rilocare il codice di un'applicazione mentre questa è in esecuzione.

Il byte code arriva al **CLASS LOADER** (o **code loader**) tramite la rete oppure il file system.
Il class loader si occupa di risolvere il nome di ogni classe, occorre sapere com'è fatta ogni classe per poter allocare la memoria.
Il class loader quindi reperisce il codice della classe.

> Nota:
> Il class loader di Java è riprogrammabile tramite delle API (application program interface) mediante cui ci può ridefinire il comportamento del class loader.

Di default il class loader riceve la classe da caricare. Cerca il codice sul disco fisso (o nel nodo di rete in caso di applet) in un insieme di directory definite in una variabile di ambiente (tipo il PATH di dos e windows) detta **CLASSPATH**. Se la classe viene trovata, viene caricata in memoria, altrimenti viene sollevata un'eccezione.

Il byte code viene poi verificato (controllando che sia compatibile con il sorgente) dal **BYTE CODE VERIFIER** poiché viaggiando in rete potrebbe essere manomesso.

Ci sono infine due modalità:

INTERPRETAZIONE: ogni istruzione del byte code è verificata ed eseguita

ESECUZIONE JUST IN TIME: il file .class è tradotto da codice intermedio a nativo (dipendente dalla macchina). Non è compilato tutto il programma ma ogni singola classe man mano che serve (per diluire durante l'esecuzione i costi in termini di tempo dovuti alla traduzione)

NOZIONI PRELIMINARI

IL PRIMO PROGRAMMA

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

Il main non è una funzione, è un metodo. Quando eseguo un programma java, devo lanciare il programma che contiene il metodo main.

> ***Nota:***
> *Per il debug, si consiglia di mettere un metodo main in ogni classe così da poter fare il testing delle unità.*

```
public class HelloWorld {
```

specifico il metodo di accesso alla classe

```
public static void main(String args[]){
```

può esserci o meno davanti ad un metodo a attributo, indica che sono metodi o attributi **di classe**.

Per gli attributi: ne esiste uno per ogni classe, non uno per ogni oggetto (potrebbe essere ad esempio usato per implementare le variabili globali), un esempio classico è dato dall’implementazione di un contatore di oggetti allocati.

I metodi static possono essere usati per modificare attributi static oppure anche per realizzare delle librerie di funzioni.

COMPILAZIONE ED ESECUZIONE

```
c:\classes>javac HelloWorld.java
c:\classes>java HelloWorld
HelloWorld!
c:\classes>
```

LA STRUTTURA DI UN PROGRAMMA JAVA

Un programma Java è organizzato come un insieme di classi
Classi diverse possono essere raggruppate all’interno della stessa “Compilation unit” che hanno un estensione **.java**

Il programma principale è rappresentato da un metodo speciale (main) della classe il cui nome coincide con il nome del programma

JAVA - IL PRIMO PROGRAMMA GRAFICO

```
import java.awt.*;
package examples,
class HelloWorldWindow {
public static void main(String args[]) {
Frame f=new Frame("HelloWorldWindow");
f.add(new Label("HelloWorld!",Label.CENTER),"Center");
f.pack();
f.setVisible(true);
}
}
```

import: importa caratteristiche particolari di un particolare package.

package examples: il programma è dichiarato nel package examples. Il file deve essere nella directory examples. lo compilo li e lo eseguo (oppure in una dir esterna, ma riferendomi al nome completo della classe che è: examples.HelloWorld)

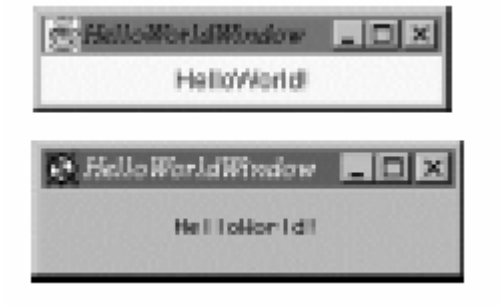
Un package definisce uno spazio dei nomi; consente di avere un meccanismo per strutturare lo spazio dei nomi (posso ad esempio definire due classi con lo stesso nome in package diversi). Vincola inoltre il modo in cui i file sono memorizzati nel file system.

I package possono essere annidati: ad esempio

myProg

- examples
- lectures

accederei ad una classe di examples con la notazione puntata: **myProg.examples.NomeClasse**



COMPILIAZIONE ED ESECUZIONE

```
c:\classes>cd examples
c:\classes\examples>javac HelloWorldWindow.java
c:\classes\examples>cd ..
c:\classes>java examples.HelloWorldWindow
c:\classes>
```

LA STRUTTURA DI UN PROGRAMMA JAVA

- **Package**
  - Ogni package contiene una o più compilation unit
  - Il package introduce un nuovo ambito di visibilità dei nomi
- **Compilation unit**
  - Ogni compilation unit contiene una o più classi o interfacce delle quali una sola pubblica
- **Classi e interfacce**
- Relazioni con il file system
  - package ⇔ directory
  - compilation unit ⇔ file

IL LINGUAGGIO JAVA IN DETTAGLIO

- Programmazione “in the small”
  - Tipi primitivi
  - Dichiarazione di variabili
  - Strutture di controllo: selezione condizionale, cicli
- sono identiche a quelle del C: if, switch, while, for...
  - Array
- Programmazione “in the large”
  - Classi
  - Interfacce
  - Packages

TIPI PRIMITIVI E VARIABILI

Tipi numerici:

- byte: 8 bit
- short: 16 bit
- int: 32 bit
- long: 64 bit
- float: 32 bit
- double: 64 bit

Altri tipi:

- boolean: true o false
- char: 16 bit, carattere Unicode

I tipi primitivi sono allocati nello stack (sempre), mentre in C++ li potevo allocare nello heap mediante i puntatori.

Per allocare i tipi primitivi nello heap si possono usare i wrapper, che mi permettono di avvolgere all'interno di una classe un tipo primitivo.

DICHIARAZIONE DI VARIABILI

```
byte un_byte;
int a, b=3, c;
char c='h', car;
boolean trovato=false;
```

I TIPI DI RIFERIMENTO

- Tipi array
- Tipi definiti dall'utente
  - Classi
  - Interfacce

I tipi oggetto sono sempre allocati nello heap (mentre in C++ se li definivo come tipi normali, venivano allocati sullo stack e venivano allocati nello heap solo con la *new*)

TIPI ARRAY

Anche gli array sono considerati tipi riferimento e sono allocati nello heap.

Dato un tipo T (predefinito o definito dall'utente) un array di T è definito come: **`T[]`**

Similmente sono dichiarati gli array multidimensionali: **`T[][] T[][][]`** …

Esempi:
int[] *//array di interi*
float[][] *//matrice di float*
Persona[] *//array di elementi riferimento*

TIPI ARRAY: DICHIARAZIONE E INIZIALIZZAZIONE

Dichiarazione:
int[] ai1, ai2;
float[] af1;
double ad[];
Persona[][] ap;

Inizializzazione:
int[] ai={1,2,3};
double[][] ad={{1.2, 2.5}, {1.0, 1.5}}

Posso anche inizializzarli al volo.

TIPI ARRAY: ALLOCAZIONE DI MEMORIA

In mancanza di inizializzazione, la dichiarazione di un array non alloca spazio per gli elementi dell'array.

L'allocazione si realizza dinamicamente tramite l'operatore:

`new <tipo> [<dimensione>]`

La dimensione dell'array deve essere nota.
Esiste una classe **Vector** che consente di avere più oggetti dello stesso tipo, senza dover dichiararne il numero.

Esempi:
int[] i=new int[10], j={10,11,12};
float[][] f=new float[10][10];
Persona[] p=new Persona[30];

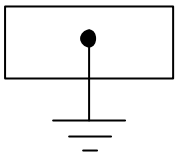
Se gli elementi non sono di un tipo primitivo, l'operatore "new" alloca solo lo spazio per i riferimenti. In C++ se dichiaro una variabile senza inizializzarla, a questa non viene assegnato nessun valore, in Java viene assegnato un valore di default che per i riferimenti è "null".

ARRAY DI OGGETTI: DEFINIZIONE

```
Person[] person;
```

crea lo spazio per il riferimento, l'array vero e proprio non esiste

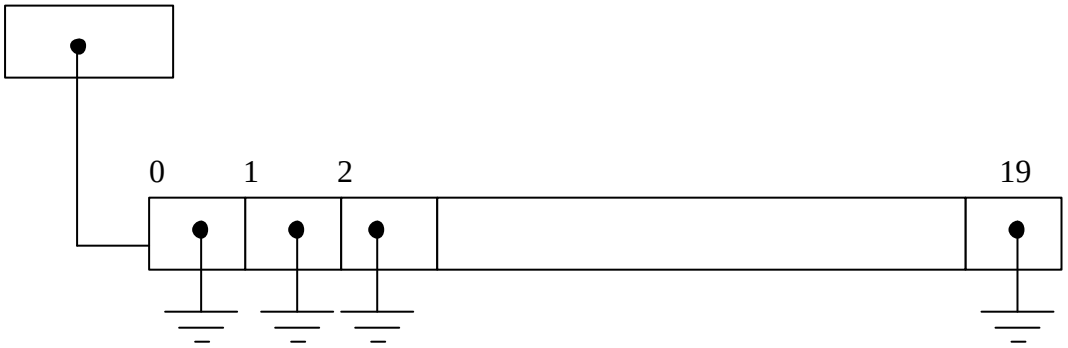
person



```
Person[] person;
person = new Person[20];
```

Ora l'array è stato creato ma i diversi oggetti di tipo Person non esistono ancora.

person

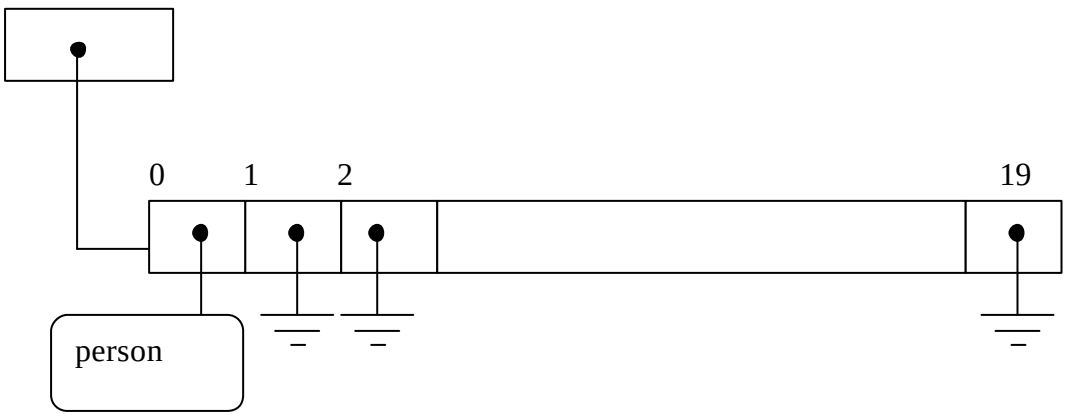


L'allocazione crea 20 celle di memoria logiche tutte contenenti un riferimento inizializzato a null.

```
Person[] person;
person = new Person[20];
person[0] = new Person()
```

Un oggetto di tipo persona è stato creato e tale oggetto è stato inserito in posizione 0.

person



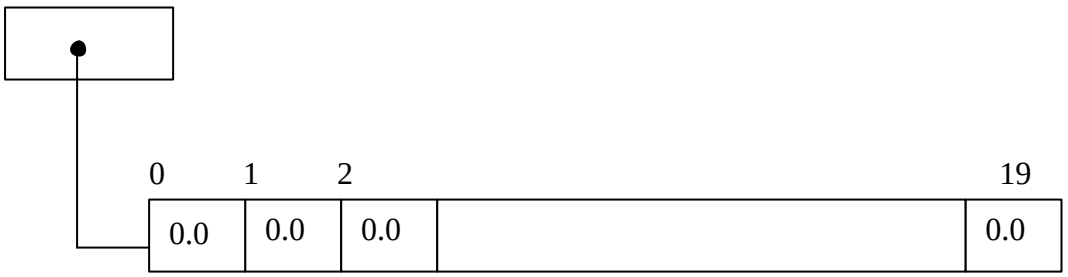
ARRAY DI OGGETTI VS. ARRAY DI TIPI BASE

L'istruzione:

```
float f[] = new float[20];
```

crea un oggetto di tipo array di float e alloca spazio per 20 float

f

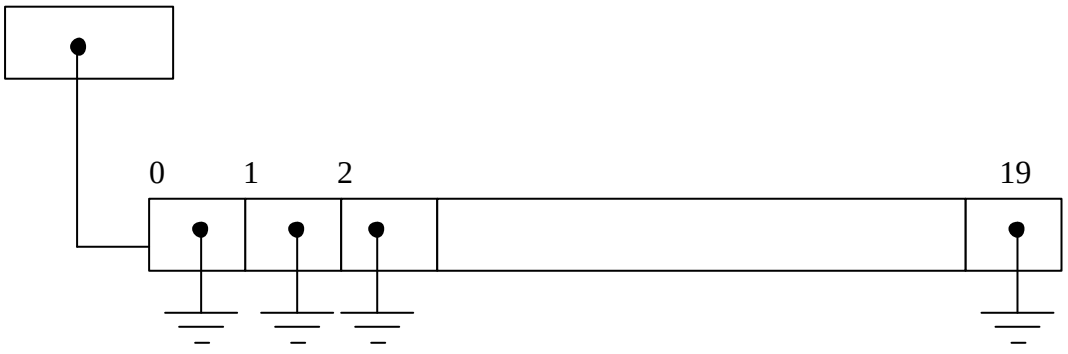


L'istruzione:

```
Person p[] = new Person[20];
```

crea un oggetto di tipo array di Person e alloca spazio per 20 riferimenti a oggetti di tipo Person

person



Non viene allocato spazio per gli oggetti veri e propri

Diverso dal C++, in cui potrei scrivere **`Person p[20];`** poiché l'array è allocato staticamente ed ogni cella di memoria contiene lo spazio per un oggetto di tipo persona. Se faccio allocazione dinamica con la new è come Java.

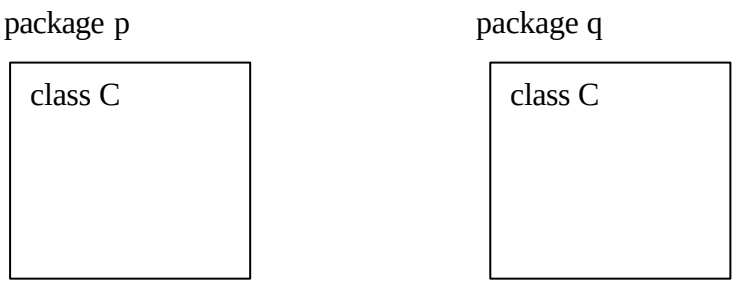
DEFINIZIONE DI UNA NUOVA CLASSE

Una nuova classe viene definita nel seguente modo:

```
class <nome classe> {
      <lista di definizioni di attributi e metodi>
}
```

Esempio:
class Automobile {...
}

CLASSI DEFINITE A LIVELLO PUBLIC O PACKAGE



Le due classi C sono distinte. Il package permette di restringere la visibilità della classe.

```
public class C
```

C è visibile a tutte le classi che hanno importato il package dove c'è C

```
class C
```

C è visibile solo dentro il package in cui è definita.

In un package ci deve essere almeno una classe public che serva da interfaccia con l'esterno.
Se non definiamo nessun package, le classi sono messe in un package di default e sono tutte tra di loro visibili.

DEFINIZIONE DI ATTRIBUTI

Gli attributi costituiscono lo "stato" degli oggetti appartenenti ad una classe
Gli attributi si definiscono usando la stessa sintassi della dichiarazione di variabili

Esempio

```
class Automobile {
      String colore, marca, modello;
      int cilindrata, numPorte;
      boolean accesa;
}
```

DEFINIZIONE DI METODI

I metodi definiscono il comportamento degli oggetti appartenenti ad una classe

Ogni metodo è definito come segue:

```
<tipo val. rit.> <nome.>([<dic. par. formali>]){
     <corpo>
     }
```

Il tipo void viene utilizzato per metodi che non ritornano alcun valore

Esempio:

```
class Automobile {
     String colore, marca, modello;
     int cilindrata, numPorte;
     boolean accesa;
     void accendi() {accesa=true;}
     boolean puoPartire() {return accesa;}
     void dipingi(String col) {colore=col;}
     void trasforma(String ma, String mo) {
          marca=ma;
          modello=mo;
     }
}
```

CLASSI E OGGETTI

Le **istanze** di una classe si chiamano **oggetti**

Ogni variabile il cui tipo sia una classe (o un interfaccia) contiene un **riferimento** ad un oggetto

Ad ogni variabile di tipo riferimento può essere assegnato il riferimento **null:** Automobile a=null;

Regola per il passaggio parametri:

- I parametri il cui tipo sia uno dei **tipi semplici** sono passati per **copia**
- I parametri il cui tipo sia un tipo **riferimento** (classi, interfacce e array) sono passati per **riferimento** (ovvero per copia del riferimento)

- Quindi, gli oggetti sono sempre passati per riferimento

ESEMPIO

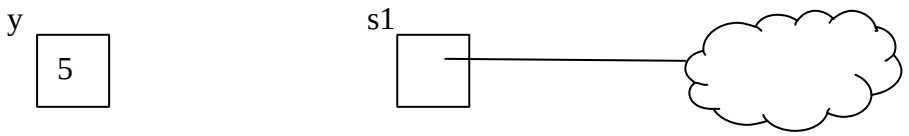
Abbiamo una classe che definisce un metodo

```
class NomeClasse{
      …
      m (int x, Stack s) {
            …
      }
      …
}
```

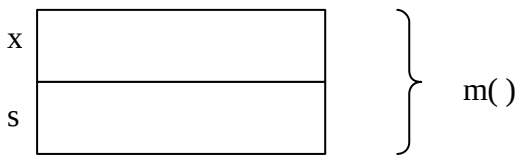
Invochiamo:

```
NomeClasse o = new NomeClasse();
int 5;
Stack s1 = new Stack(20);
o.m(y, s1);
```

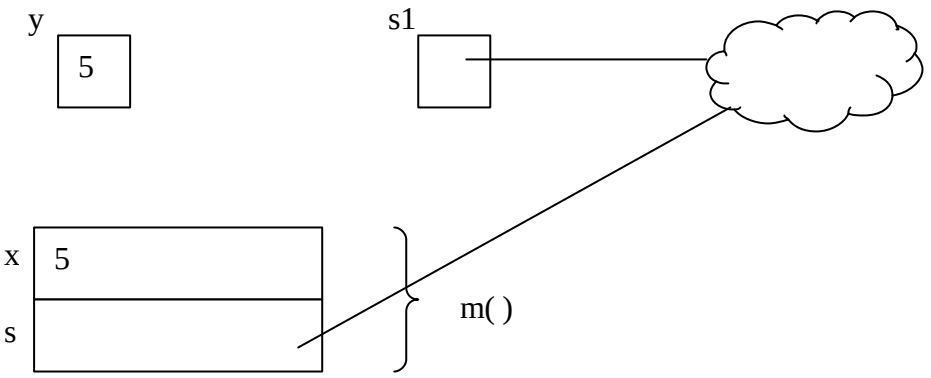
Vediamo cosa succede



Nell'AR di m, davo avere due celle, una per x ed una per s



Il 5 viene copiato nel parametro formale x, mentre per s copiamo il valore del puntatore.



s viene passato per riferimento.

ACCESSO AD ATTRIBUTI E METODI DI UN OGGETTO

L'accesso ad attributi e metodi di un oggetto per il quale si abbia un riferimento si effettua tramite la "notazione punto" (inglese: **dot-notation**)

Esempio:

```
Automobile a;
....
if(a!=null){
      a.accendi();
      a.dipingi("Blu");
}
```

ACCESSO AD ATTRIBUTI E METODI LOCALI

Nella definizione di un metodo ci si riferisce ad attributi e metodi dell'oggetto sul quale il metodo sia stato invocato direttamente (senza notazione punto)

Esempio

```
class Automobile {
      String colore;
      void dipingi(String col) {colore=col;}
      ...
}
```