

JAVA e i linguaggi di programmazione ad oggetti

Marco Gori
Marco Maggini
Fabrizio Santini
Franco Scarselli



Programmazione ad Oggetti

Il paradigma di Programmazione orientata agli oggetti (OOP) migliora il supporto per lo sviluppo e la gestione del software.

Linguaggi ad oggetti: [JAVA](#), [C++](#), [SmallTalk](#), [Modula-2](#)

Concetti Base della OOP

- Tipo di dato astratto:
Modello + insieme di operazioni = Incapsulamento
- Classe:
 - Descrive il comportamento globale
 - Fornisce un elenco di metodi utilizzabili con il dato astratto
 - Descrive i dettagli sulla implementazione e sulla struttura del dato, separando gli elementi della classe in:
 - Pubblici**: che possono essere utilizzati anche da altre classi
 - Privati**: che possono essere utilizzati solo nella classe di appartenenza

Concetti Base della OOP

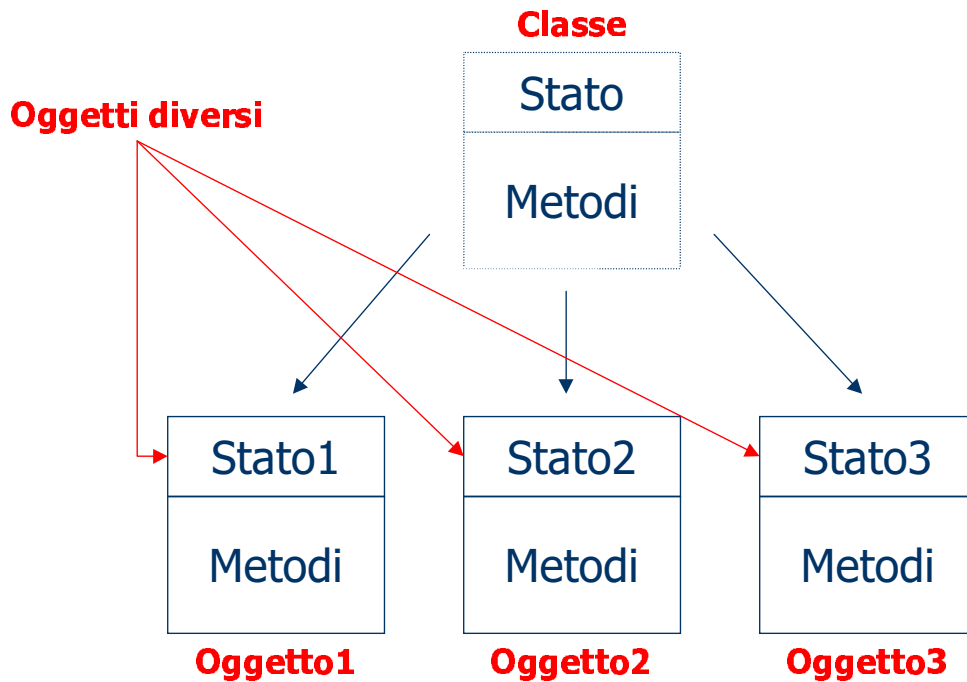
- Oggetto: **Variabile dichiarata come appartenente ad una classe**. È composta da uno **stato** e dai **metodi**.

Lo stato è rappresentato dai valori dei campi (pubblici e privati) descritti nella definizione della classe

I metodi sono procedure che possono essere applicate ad una istanza e modificare lo stato dell'oggetto stesso.

L'oggetto è una istanza fisica (che occupa memoria) di una classe. Due oggetti A e B appartenenti alla stessa classe sono due entità diverse perché hanno stati diversi.

Concetti Base della OOP



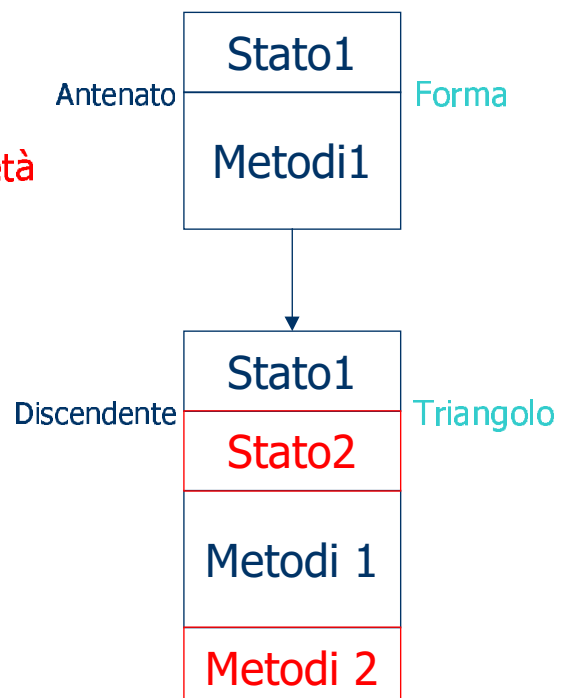
5

Concetti Base della OOP

- Ereditarietà delle classi

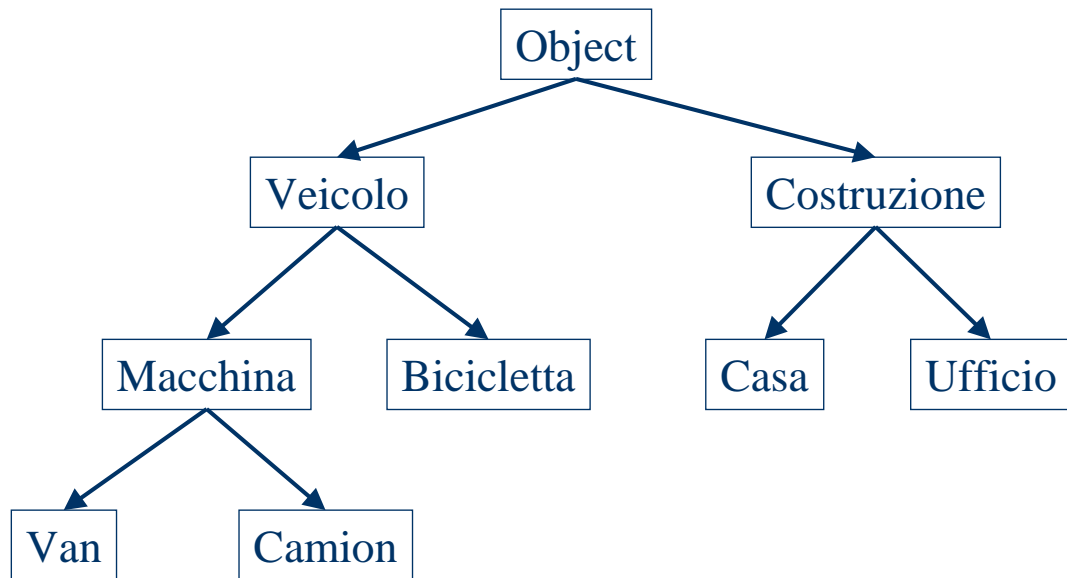
È possibile utilizzare l'**Ereditarietà** per creare nuove classi estendendo quelle già esistenti con nuove proprietà e nuovi metodi.

La ridefinizione nella classe discendente di un metodo già presente nell'antenato viene chiamato **Overriding**.

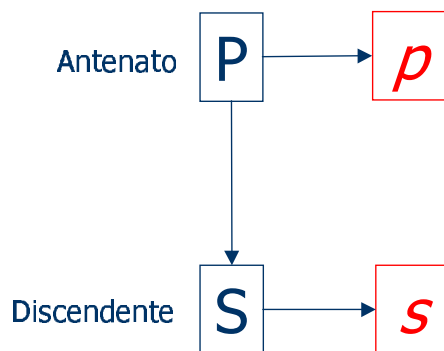


6

Ereditarietà



Ereditarietà e Polimorfismo



Ereditarietà: È possibile utilizzare *s* ovunque viene usato *p*

Polimorfismo: l'oggetto risponde a un messaggio comune a *p* ed a *s* a seconda della classe di appartenenza

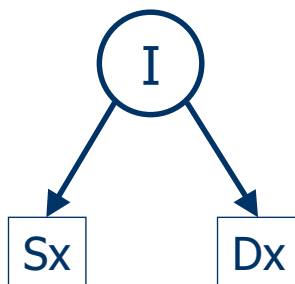
Programmazione

- Fase di **progettazione**:
 - Definizione delle classi, discendenti e loro proprietà
 - Definizione delle interfacce delle classi
- Fase di **implementazione**:

Definizione della implementazione (algoritmi) dei metodi

Vantaggi: Riutilizzabilità del codice, facilità di manutenzione e quindi riduzione dei tempi e costi di sviluppo.

Esempio di classe



```
class Nodo
  private
    oggetto Informazione
    oggetto Sottoalbero_Sx
    oggetto Sottoalbero_Dx
  public
    metodo Nuovo_Nodo
    metodo Get_Informazione
    metodo Get_Destro
    metodo Get_Sinistro
```

Riutilizzo del codice

- Aggregazione: Relazione "ha-un"

È possibile includere un oggetto come elemento del nuovo oggetto:

- Semplice
- Flessibile perché permette di aggiungere nuovi oggetti a tempo di esecuzione

- Ereditarietà: Relazione "è-un" :

È possibile ereditare tutti i metodi e le proprietà della classe antenato.

11

JAVA - la storia

- Primavera 1990 - Patrick Naughton vuole lasciare Sun, ... ma cambia idea ...
- Viene sviluppato da un piccolo team diretto da James Gosling, contemporaneamente al suo utilizzo in alcuni progetti interni alla Sun
- Originariamente si chiamava Oak (Quercia), ma prende il nome definitivo al coffee shop della Sun
- Viene lanciata FirstPerson Inc (Autunno 1992)

12

JAVA - la storia

- Primavera 1993 - nasce NCSA Mosaic
- Giugno 1994 - Bill Joy (co-fondatore Sun) intuisce il legame con il Web ... e spinge ...
- Autunno 1994 - Naughton e Payne sviluppano Webrunner (browser Sun) per girare applets
- Primavera 1995 - Sun introduce formalmente Java e HotJava (ex-Webrunner)
- Fine 1995 - Accordo con Netscape per Java-enabled browser

13

JAVA nasce per ...

- JAVA non nasce per il Web
- OAK doveva servire a realizzare una rete grande, eterogenea e distribuita con cui gli strumenti elettronici di largo consumo potessero dialogare
- Caratteristiche cruciali
 - **Indipendenza dalla piattaforma**
deve poter girare su strumenti di natura diversissima: telefoni, lavatrici, TV, PDA, PC, ...
 - **affidabilità**
evitare di dover fare il reboot della lavatrice ...
 - **sicurezza**
evitare che qualcuno possa prendere il controllo della vostra TV

14

Caratteristiche di JAVA

- JAVA è semplice:

È un linguaggio molto simile a C++, ma più semplice. Tutto quello non strettamente necessario e le caratteristiche che potevano indurre confusione sono state eliminate:

- Templates
- overloading degli operatori
- pre-processore
- gli header file
- strutture ed unioni
- Array multi-dimensionali
- Conversione implicita dei tipi
- Puntatori



15

Caratteristiche di JAVA

- JAVA è un linguaggio ad alto livello, orientato agli oggetti:

Il codice viene organizzato in classi. Java implementa il meccanismo di singola eredità, quindi ogni classe può ereditare una o più caratteristiche e comportamenti da un singolo antenato. Alla radice dell'albero di tutte le classi c'è la classe progenitrice **Object**.

La maggior parte degli elementi del linguaggio sono oggetti, anche se i tipi semplici sono implementati anche tradizionalmente.



16

Caratteristiche di JAVA

- JAVA è indipendente dalla macchina:

Un programma JAVA viene prima compilato in un formato binario chiamato **byte-codes**, e successivamente eseguito da una macchina virtuale (**JVM**).

Questa caratteristica gli conferisce una totale indipendenza dalla piattaforma di esecuzione.



17

Caratteristiche di JAVA

- JAVA è multi-threaded:

Il JAVA è naturalmente multi-threaded. Questo permette di implementare in maniera estremamente semplice, applicazioni complesse che richiedono l'uso di multi-programmazione.

- JAVA utilizza tecniche di Garbage Collection
- JAVA è robusto agli errori di programmazione



18

Caratteristiche di JAVA

- JAVA funziona bene anche su piccoli sistemi
- JAVA è intrinsecamente sicuro
- JAVA è produttivo



19

Caratteristiche di JAVA

- Heavy Compile-Time Checking
- Heavy Run-Time Checking
- Java API (e.g. Abstract Window Toolkit)

20

Caratteristiche di JAVA

- Il Pascal, il C ed altri classici linguaggi, richiedono che la deallocazione della memoria sia gestita dal programmatore
- Il JAVA utilizza tecniche per il garbage collection automatico

21

Caratteristiche di JAVA

- Java e problemi di sicurezza

Perché?

Problemi derivanti dagli applets: girano su hosts remoti.

Chi garantisce il loro comportamento?

La politica di Sun: open-source

22

Tipi diversi di programmi JAVA I

Applicazioni

- Sono normali programmi
- Per eseguirli occorre lanciare una JVM (da riga di comando)

Applets

- Appaiono all'interno di pagine HTML
- Per eseguirle occorre visualizzare la pagine HTML con un browser o lanciarle con un applet viewer
- I browser incorporano una JVM con cui eseguono l'apple
- I

Java

23

Tipi diversi di programmi JAVA II

Applicazioni

- Sono normali programmi
- Per eseguirli occorre lanciare una JVM (da riga di comando)

Applets

- Appaiono all'interno di pagine HTML
- Per eseguirle occorre visualizzare la pagine HTML con un browser o lanciarle con un applet viewer
- I browser incorporano una JVM con cui eseguono l'applet

Servlets

- Appaiono all'interno di pagine HTML
- Generano documenti Web (in genere HTML) che sono inviati al browser
- I

24

Un'applet JAVA

HTML e Java Applets

```
<html>
<head>
<title>Felix Applet</title>
</head>
<body>
<h1> Felix il gatto </h1>
Signori ... ecco Felix il gatto ...
<hr>
<applet code = Felix.class width=500 height =
300>
ouh ... se vedi questo allora forse tu non hai
un <i>browser abilitato Java! </i>
</applet>
<hr>
</body>
</html>
```

25

Un'applet JAVA

Java Applets

```
package SimpleApplet;

//la classe Felix
public class Felix extends Applet {
    JLabel jLabel1 = new JLabel();

    //Il costruttore
    public Felix() {
    }

    // questo metodo viene chiamato automaticamente dopo
    l'inizializzazione
    public void init() {
        jLabel1.setFont(new java.awt.Font("Dialog", 0, 40));
        jLabel1.setText("Miaaaoooooo...");
        this.add(jLabel1, null);
    }
}
```

26

Un'applicazione JAVA

```
/*
 * Questo programma stampa la data di esecuzione
 * del mio primo programma JAVA
 */

package MyFirstJAVA;

import java.util.*; // Importa anche la classe 'Date'

public class HelloWorld {
    public static void main(String argv[]) {

        System.out.println("Hello World!");
        System.out.println("My first JAVA program in date: ");
        System.out.println(new Date());
    }
}
```

Il modo migliore di capire la filosofia JAVA è quello di studiare subito un semplice programma.



27

Commenti in JAVA

In Java esistono due modi per scrivere commenti nel codice:

- **Commenti su una linea**

Il compilatore ignora tutto il testo che segue i due caratteri `//` fino alla fine della riga

```
public class Prova {
    public static void main(String argv[]) {
        System.out.println("Hello!"); // stampo il messaggio
    }
}
```

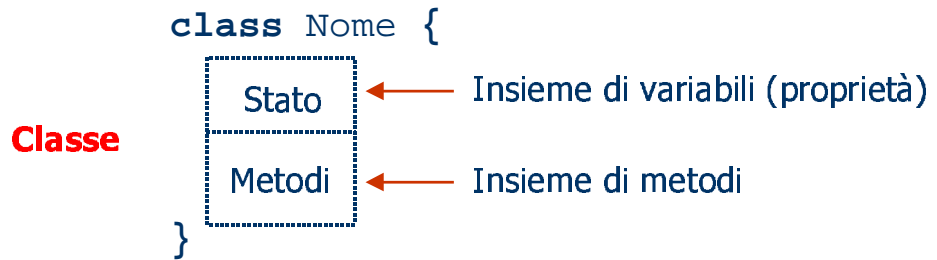
- **Commenti estesi**

Il compilatore ignora tutto il testo che è compreso fra le sequenze `/*` e `*/`

```
public class Prova {
    /* public static void main(String argv[]) {
        System.out.println("Hello!");
    } */
}
```

28

Classi in JAVA



- Le proprietà memorizzano dei dati
- I metodi definiscono delle azioni applicabili alla classe

```
public class Conto {  
    private int valuta = 0;  
  
    public void versa(int somma) {valuta += somma;}  
    public void preleva(int somma) {valuta -= somma;}  
    public int estrattoConto() { return valuta;}  
}
```

stato

metodi

29

JAVA: Access Specifiers

Esistono quattro vincoli diversi di accesso alle proprietà e metodi di una classe JAVA:

- Public → **Rende visibile all'esterno della classe**
- Private → **Nasconde all'esterno della classe**
- Protected → **Le classi discendenti possono accedere alle proprietà e metodi della classe progenitore.**
- "Frendly" → **Le classi discendenti possono accedere alle proprietà e metodi della classe progenitore.**

Tutte le classi all'interno dello stesso package possono accedere alle proprietà e metodi della classe. Questo vincolo viene considerato di default se nessun altro specificatore viene indicato.

30

JAVA: Access Specifiers II

- Per ricordarsi
 - Public: accessibile a tutti
 - Protected: alle sottoclassi
 - Friendly: a tutte le classi del package
 - Private: nessuno

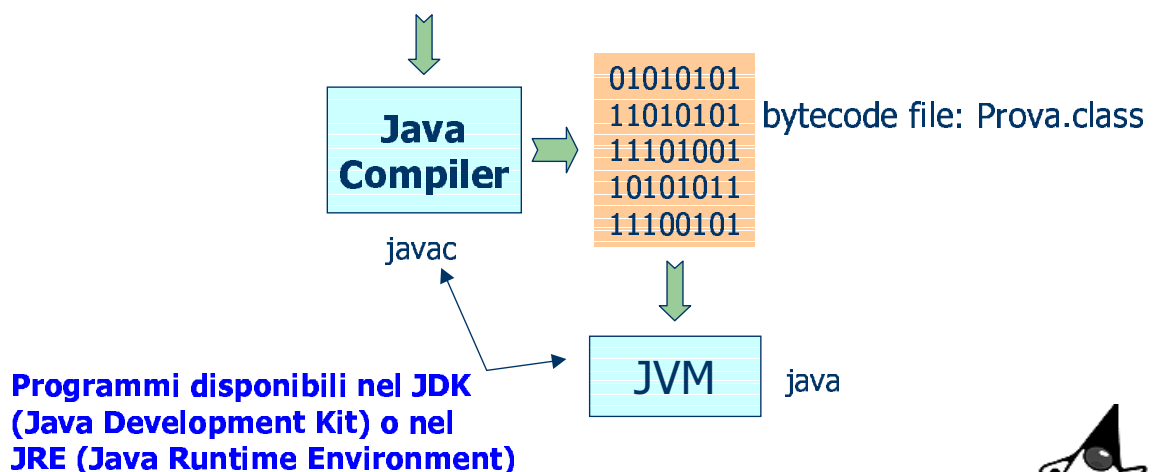
	Stesso Package		Package diversi	
	Ereditabile	Accessibile	Ereditabile	Accessibile
Public	sì	sì	sì	sì
Protected	sì	sì	sì	no
Friendly	sì	sì	no	no
Private	no	no	no	no

31

Cosa succede ad un file JAVA

File: Prova.java

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



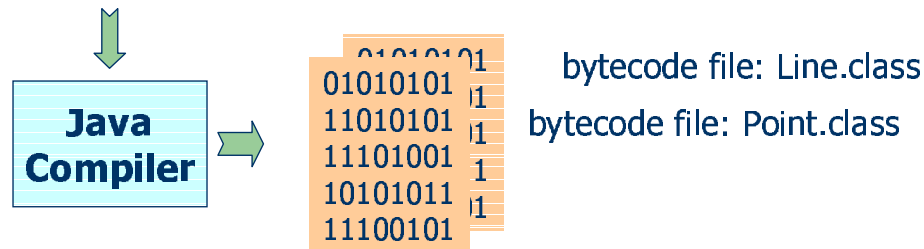
32

Il Percorso di una classe JAVA

Il risultato della compilazione è un numero di file che corrisponde al numero di classi definite

File: Prova.java

```
public class Line {  
    int x0, y0;  
    int x1, y1;  
}  
  
class Point {  
    int x, y;  
}
```



33

Librerie di classi in JAVA

Le classi omogenee possono essere raccolte all'interno di una libreria :

```
package MyLibrary;  
  
class Prova {  
    public static void main(String argv[]) {  
        System.out.println("Hello World!");  
    }  
}
```

che un altro utente può utilizzare in parte o completamente.

34

Librerie di Classi in JAVA

È possibile importare classi esterne semplicemente indicando il percorso completo:

```
import MyLibrary.Prova;  
  
class Prova2 {  
    public static void main(String argv[]) {  
        Prova MyProva = new Prova;  
        System.out.println("Hello World!");  
    }  
}  
  
// import MyLibrary.Prova;  
  
class Prova2 {  
    public static void main(String argv[]) {  
        MyLibrary.Prova MyProva = new MyLibrary.Prova;  
        System.out.println("Hello World!");  
    }  
}
```

35

Librerie di Classi in JAVA

È possibile importare anche tutte le classi di un pacchetto:

```
import MyLibrary.*;  
  
class Prova2 {  
    public static void main(String argv[]) {  
        Prova MyProva = new Prova;  
        System.out.println("Hello World!");  
    }  
}
```

In pratica lo spazio dei nomi delle librerie e delle classi JAVA assomigliano ai domini internet.

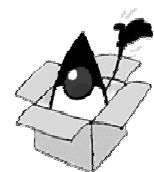
```
import Projects.Shared.MyLibrary.3D.*;
```

Il nome del pacchetto contiene anche il path relativo delle classi: `/Projects/Shared/MyLibrary/3D/`

36

Oggetti in JAVA

- Le classi sono una descrizione teorica di ciò che potrà fare l'entità che vogliamo definire
- Per poterle utilizzare bisogna creare le istanze
- Si creano oggetti a partire da una classe utilizzando l'operatore **new**



37

Oggetti in JAVA

Tutte le entità in JAVA sono oggetti

L'identificatore utilizzato per riferirsi ad un oggetto viene chiamato **handle**. Ovviamente un handle non è collegato necessariamente ad un oggetto.

Definizione:

- Solo Handles:
<nome-classe> nome-handle
- Handle + Oggetto:
<nome-classe> nome-handle = oggetto;



38

Esempio

Definizione di una stringa

Def. di un handle:

```
String s;
```

In questo caso l'oggetto non esiste, e se provo ad accedervi si genera un run-time error.

Def. di un handle + oggetto

```
String s = "Java";
```

Valido solo per le stringhe

```
String s = new String("Java");
```

In questo caso l'handle viene definito e successivamente inizializzato con l'indirizzo dell'oggetto appena creato

COSTRUTTORE DELL'OGGETTO

Questo metodo speciale indica come creare l'oggetto. Ci può essere più di un costruttore, ognuno dei quali possiede un insieme di parametri diversi (per tipo e/o numero).

39

Dove “vivono” gli oggetti?

Lo **scope** (o campo di visibilità) è la zona in cui un oggetto JAVA vive. Questa zona è delimitata dai simboli { e }.

Non permesso perché la variabile i è stata già definita nello scope 1.

```
{  
    int i = 5;  
    {  
        float w = 0.5;  
        /* int i = 3; */  
    }  
}
```

Scope 2

Scope 1


40

Quanto “vivono” gli oggetti?

La “vita” di tutti gli oggetti JAVA è gestita automaticamente: ci si preoccupa di creare un oggetto ma non di distruggerlo.

Questo è possibile perché esiste una entità, chiamata **Garbage Collector**, che provvede a liberare la memoria occupata da oggetti che non sono più utilizzati.

```
{  
    Float W = new Float(0.5);  
}
```



L'handle **W** scompare, ma non l'oggetto associato!

41

E il main?

Se un programma JAVA è fatto con aggregazione di oggetti, allora una funzione **main** non esiste?

In JAVA esiste il concetto di oggetto principale

All'interno di questo oggetto esiste un metodo che viene chiamato automaticamente, all'avvio dell'oggetto stesso.

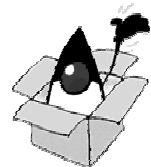
```
package MyLibrary;  
  
class Prova {  
    public static void main(String argv[]) {  
        System.out.println("Hello World!");  
    }  
}
```

42

Tipi primitivi in JAVA

Tipi di uso frequente:

- Non vengono definiti gli handle ma le variabili vere e proprie. Questo è dovuto al costo di gestione del **new** per variabili semplici
- L'occupazione di memoria di questo tipo di variabili è indipendente dalla macchina ospite
- Per tutti i tipi primitivi esistono delle classi, chiamate **wrapper**, che permettono la creazione di oggetti del tipo associato in modo dinamica



43

Tipi primitivi in JAVA

Tipo	Dimensione	Min	Max	Wrapper
boolean	1 bit	-	-	Boolean
char	16 bit	0	$2^{16}-1$	Char
byte	8 bit	-128	+127	Byte
short	16 bit	-2^{15}	$+2^{15}-1$	Short
int	32 bit	-2^{31}	$+2^{31}-1$	Integer
long	64 bit	-2^{63}	$+2^{63}-1$	Long
float	32 bit	IEEE 754		Float
double	64 bit	IEEE 754		Double
void	-	-	-	Void

44

Esempio

```
int i = 5;
```

Variabile chiamata **i** inizializzata con il valore 5

```
Integer I;
```

Handle chiamato **I** che può puntare ad un oggetto di classe Integer

```
Integer I = new Integer(5);
```

Handle chiamato **I** a cui è stato assegnato un oggetto di classe Integer inizializzato con il valore 5.

45

Costanti in JAVA

- Una costante è di un tipo specifico e può essere usata per inizializzare una variabile dello stesso tipo:

```
long    l = 34L;  
short   s = 0xCAFE;  
boolean b = true;  
float   f = 0.34e23F;  
double  d = 34.45D;
```

- Le costanti carattere sono rappresentate dal carattere fra apici semplici

```
char c = 'A';
```

- I caratteri UNICODE sono rappresentati col loro codice in esadecimale con il prefisso \u (es. \u 2122)
- Sono definite delle sequenze per rappresentare caratteri speciali (es. \n è l'avanzamento di riga, \b il backspace, \\ la barra \)

46

Inizializzazione nelle classi

- I tipi primitivi vengono sempre inizializzati ad un valore di default (zero)
- Gli handle ad oggetti vengono inizializzati a NULL

```
class Segmento {  
  
    int x1, y1;        // Vengono inizializzati a zero  
    int x2, y2;        // idem  
  
    Integer Modulo;    // Viene inizializzato a NULL  
}
```

Anche in questo caso l'oggetto non esiste, e se provo ad accedervi si genera un run-time error.

47

Inizializzazioni nelle classi

```
public class Conto {  
  
    private int valuta = 0;  
  
    public void versa(int somma) {valuta += somma;}  
  
    public void preleva(int somma)  
    {  
        int Spese;  
        /* int Spese = 0; Con questa modifica è meglio */  
        valuta -= (somma + Spese);  
    }  
  
    public int estrattoConto() { return valuta;}  
}
```

Se si tenta di accedere alle variabili locali dei metodi nelle classi senza una preventiva inizializzazione, viene generato un errore di compilazione

48

Accesso alle proprietà

Si accede alle proprietà degli oggetti specificando: `<nome-classe>.nome-proprietà`

```
Segmento NuovoSegmento = new Segmento;
```

```
NuovoSegmento.x1 = 5;
```

```
NuovoSegmento.y1 = 3;
```

```
class DueSegmenti {  
    Segmento S1 = new Segmento;  
    Segmento S2 = new Segmento;  
}
```

```
DueSegmenti Spezzata = new DueSegmenti;
```

```
Spezzata.S1.x1 = 5;
```

```
Spezzata.S1.y1 = 3;
```

Definizione dei metodi

I metodi indicano quali azioni è possibile compiere sugli oggetti, e sono definiti con:

- Nome
- Lista di argomenti
- Tipo del valore risultante
- Corpo

```
tipo-risultante nome_metodo (lista argomenti) {  
    /* corpo del metodo */  
}
```

- I metodi non esistono al di fuori delle classi di appartenenza
- Non si può più parlare di "parametri formali", perché chiamare un metodo significa mandare un messaggio all'oggetto in questione

Esempio di metodo

Tipo del valore di ritorno

```
import java.lang.Math.*;
```

```
class Segmento {
```

```
    float x1, y1;
```

```
    float x2, y2;
```

```
    float Modulo(float Scala) {
```

```
        float l1 = x2 - x1;
```

```
        float l2 = y2 - y1;
```

```
        return Math.sqrt(l1 * l1 + l2 * l2) * Scala;
```

```
    }
```

Istruzione di ritorno del metodo

Corpo del metodo

Nome

Lista argomenti

51

Argomenti dei metodi

La lista di argomenti specifica le informazioni da trasmettere nel messaggio all'oggetto. Possono essere di due tipi:

- Tipi primitivi
- Handle di oggetti

Non restituisce nessun valore

Questo è un handle

```
void Connetti(Segmento S) {
```

```
    x2 = S.x2;
```

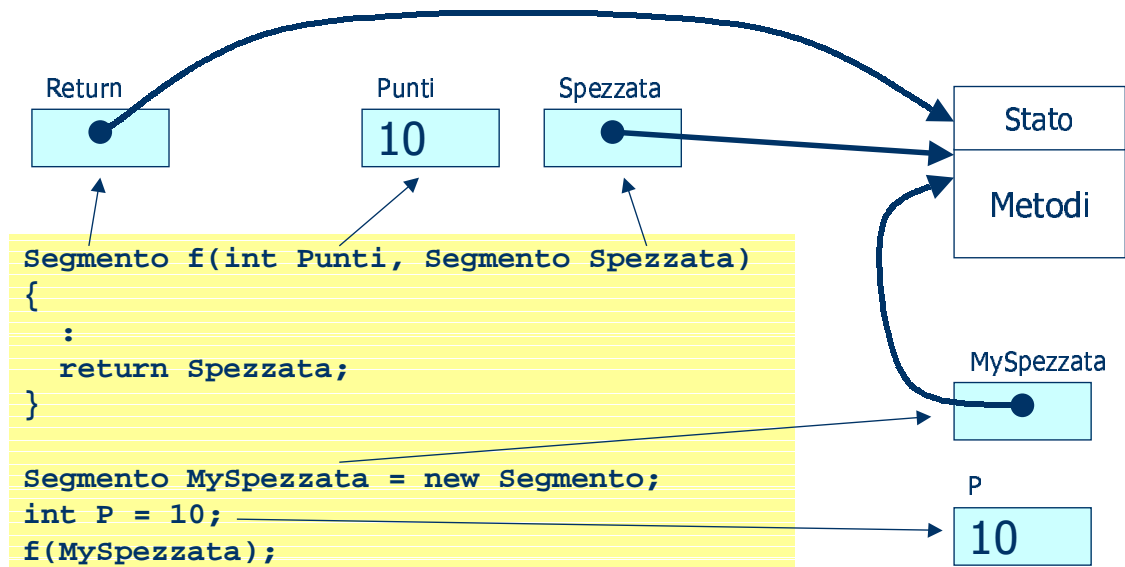
```
    y2 = S.y2;
```

```
}
```

52

Passaggio e ritorno di oggetti

Il passaggio di variabili di tipo primitivo avviene **per copia**, mentre gli oggetti vengono passati **per riferimento**



53

Primo operatore: Assegnazione

- L'assegnazione permette di definire il valore memorizzato in una variabile
- Il valore assegnato deve essere compatibile col tipo della variabile, altrimenti si genera un errore

```
int a, b;
float v1, v2;
char c = 'M';

a = 4;
b = a;
v1 = 3.2;
v2 = a;
b = v1;
```

variabile ← espressione

b assume il valore di a (4)

ammesso: v2 assume il valore di a (4) convertito in float

non ammesso

- Le conversioni ammesse in modo implicito fra tipi diversi sono:

byte → short → int → long → float → double



54

Operatori comuni

- Operatori unari

Permettono di effettuare operazioni di inversione di segno e autoincremento:

- Positivo +, Negativo -
- Pre-incremento ++a, Pre-decremento --a
- Post-incremento a++, Post-decremento a--

```
class Prova {  
  
    public static void main(String[] args) {  
  
        int A = 10;  
        System.out.println(A);           // Stampa 10  
        System.out.println(A++);         // Stampa 10  
        System.out.println(++A);         // Stampa 12  
    }  
}
```



55

Operatori comuni

- Operatori aritmetici

- Moltiplicazione *, Divisione /
- Modulo %
- Addizione +, Sottrazione -

- Operatori relazionali

Permettono di costruire espressioni logiche

- Uguale ==
- Diverso !=
- Maggiore >
- Maggiore e Uguale >=
- Minore <
- Minore e Uguale <=

```
(c != 'n')
```

```
(b == 's')
```

È uguale a true se
c **non** è il carattere 'n'

È uguale a true se b è il carattere 's'



56

Operatori comuni

- Operatori booleani

Permettono di costruire espressioni logiche:

- And &&
- Or ||
- Not !

```
( !(c == 'n') || (c == 'N')) )
```

c non è il carattere 'n' minuscolo o maiuscolo

```
( (b == 's') || (b == 'S')) )
```

b è il carattere 's' minuscolo o maiuscolo

```
( !(a < 0.5) && (a >= 0)) )
```

a è compreso tra 0 (incluso) e 0.5 (escluso)

Non vengono mai valutate tutte i termini della condizione, ma solo fino a quando il risultato è univocamente determinato.

```
T      T      F      T  
v1 && v2 && v3 && v4
```

la condizione v4 non viene valutata perché il risultato finale non può più cambiare



57

Operatori comuni

- Operatori bit a bit

Permettono di costruire espressioni logiche:

- And &
- Or |
- Xor ^
- Not ~

```
class ProvaXor {  
  
    public static void main(String[] args) {  
  
        boolean A = 0x80; // (1000 0000)2  
        boolean B = 0xAB; // (1010 1011)2  
  
        System.out.println(A ^ B); // Stampa 0x2B  
                                   // (0010 1011)2  
    }  
}
```

58

Operatori “meno comuni”

- Operatori di shift

Permettono di traslare a destra o a sinistra sequenze di bit:

- Shift a destra >>
- Shift a sinistra <<

```
1001001011101101 >> 2 → 0010010010111011
```

```
1001001011101101 << 3 → 0010010111011000
```

- Operatore ?

Permette di introdurre condizioni di assegnazione particolari:

```
espressione-boolean ? valore0 : valore1
```

```
double A = Math.Random();
```

```
int B = (A > 0.5)? 10: 20;
```

assume il **valore0** se la condizione è true, altrimenti **valore1**

Varianti di operatori

Per quasi tutti gli operatori è possibile sostituire una espressione del tipo:

```
variabile = variabile <operatore> valore
```

con una scrittura abbreviata:

```
variabile <operatore>= valore
```

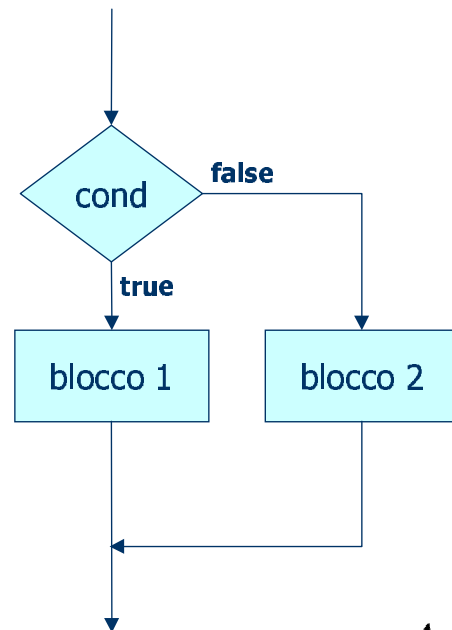
esempio:

```
double A = 0.0D;  
  
// Invece di scrivere A = A + 2.0D  
// abbiamo scritto A += 2.0D  
System.out.println(A += 2.0D); // Stampa 2.0  
  
System.out.println(A /= 2.0D); // Stampa 1.0
```

Controllo di flusso: If ... else ...

```
if (boolean-cond) {  
    blocco 1  
}  
else {  
    blocco 2  
}
```

```
if (vendite > media) {  
    bonus = 0.1;  
    guadagno = quota * bonus;  
}  
else {  
    guadagno = 0;  
}
```



61

Controllo di flusso: switch

```
switch (variabile) {  
    case Valore1: blocco1  
        break;  
    case Valore2: {  
        blocco2;  
        break;  
    }  
    default: blocco default;  
}
```

```
char C = 'n';  
switch (variabile) {  
    case 'n': System.out.println("Sconto");  
        break;  
    case 's': System.out.println("Sconto no");  
        break;  
    default: System.out.println("Boh!");  
}
```

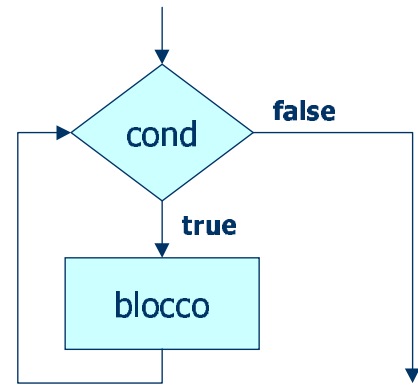
62

Cicli indeterminati

- Cicli While

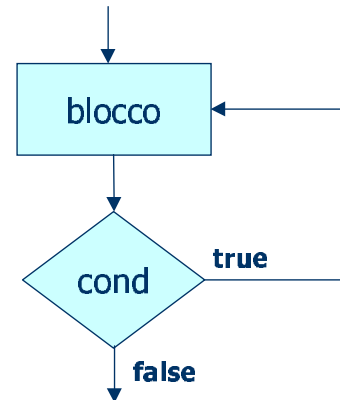
```
while (boolean-cond) {  
    blocco  
}
```

```
while (true) {  
    System.out.println("Infinito");  
}
```



- Cicli Do ... While

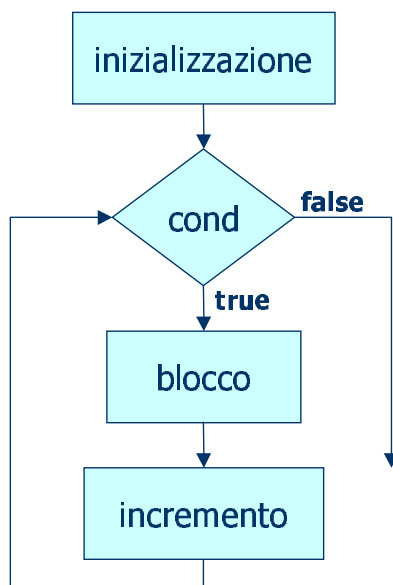
```
do {  
    blocco  
} while (boolean-cond)
```



63

Cicli determinati

```
for (inizializzazione; boolean-cond; incremento) {  
    blocco  
}
```



```
for (int i = 0; i < 5; i++) {  
    System.out.println(i);  
}
```

```
for (;;) {  
    System.out.println("Infinito");  
}
```

- Operatore Comma

```
for (int i = 0, j = 5; i < 5; i++, j--)  
{  
    System.out.println(i);  
    System.out.println(j);  
}
```

64

Break e Continue

- L'istruzione Break interrompe l'attuale istruzione di iterazione (while, do while, for).
- L'istruzione Continue interrompe la corrente iterazione e ritorna all'inizio del ciclo, iniziandone un'altra.

```
for (int i = 0, j = 10; i < 15; i++, j--) {  
    System.out.println(i);  
    if (i % 2 == 0)  
        continue;  
    if (j < 0)  
        break;  
    System.out.println(j);  
}
```



65

Break e Continue

- Le istruzioni Break e Continue permettono di interrompere anche cicli diversi da quelli in esecuzione.

```
CycleI:  
    // Possono esserci anche commenti  
    // purché non ci siano altre istruzioni  
for (int i = 0; i < 15; i++) {  
    CycleJ:  
    for (int j = 0; j < 15; j++) {  
        System.out.println("i=" + i + "j=" + j);  
        if (j > 4) continue CycleI;  
        if (i > 10) break CycleJ;  
    }  
}
```

In questi casi le etichette devono essere indicate subito prima del ciclo relativo.

Output:

```
i=0 j=0  
i=0 j=1  
i=0 j=2  
i=0 j=3  
i=0 j=4  
i=0 j=5  
i=1 j=0  
i=1 j=1  
i=1 j=2  
i=1 j=3  
:  
:  
i=10 j=4  
i=10 j=5  
i=11 j=0
```

66