Il linguaggio Java

(Seminario extra-programma)

Corso di ***Applicazioni Telematiche***

A.A. 2006-07 – Lezione n.8

Prof. Roberto Canonico

Università degli Studi di Napoli Federico II
Facoltà di Ingegneria

CHE COS'È JAVA

- È un linguaggio (e relativo ambiente di programmazione) definito da Sun Microsystems

- … per permettere lo sviluppo di applicazioni su *piattaforme multiple*, in *reti eterogenee* e *distribuite* (Internet)

IL LINGUAGGIO JAVA: CARATTERISTICHE

- Orientato agli oggetti
- Interpretato
- Architetturalmente neutro e portabile
- Robusto
- Distribuito
- Sicuro
- Dinamico
- Concorrente

3

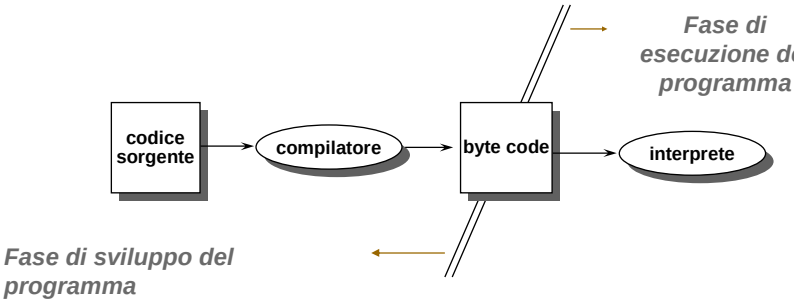
SEMPLICE E OO

- Sintassi simile a C e C++ (facile da imparare)
- Adotta il modello di programmazione Object Oriented
- Rivisita C++ in alcuni aspetti, eliminando i costrutti più "pericolosi" (allo scopo di evitare errori di programmazione difficili da individuare)
- Aggiunge *garbage collection* automatica

4

INTERPRETATO

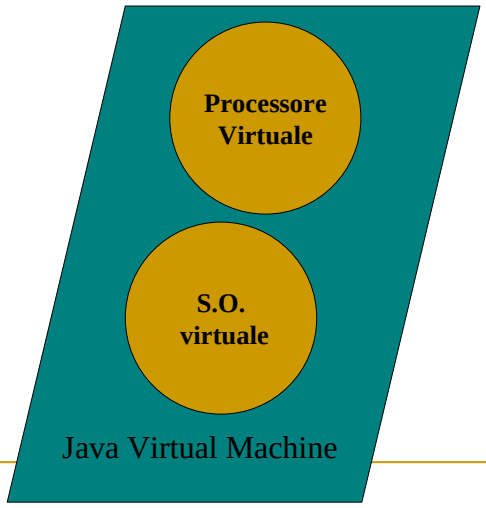
- Il compilatore produce un codice di tipo intermedio ("*byte-code*") per una "Java Virtual Machine"...
- … che viene interpretato a tempo di esecuzione



5

JAVA VIRTUAL MACHINE

- Un processore virtuale e un sistema operativo virtuale costituiscono, nel loro insieme, un ambiente virtuale entro cui gira il codice Java
- Tale ambiente è detto "Macchina Virtuale Java"



ARCHITETTURALMENTE NEUTRO

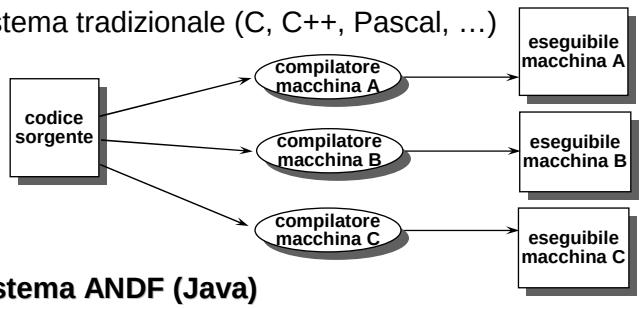
- Il byte-code è indipendente dall'architettura hardware e dal Sistema Operativo del sistema su cui dovrà essere eseguito

  (ANDF: Architecture Neutral Distribution Format)
- Pertanto, un programma bytecode può essere eseguito su qualsiasi sistema su cui giri un ambiente run-time Java, senza dover ri-compilare il sorgente su ogni piattaforma

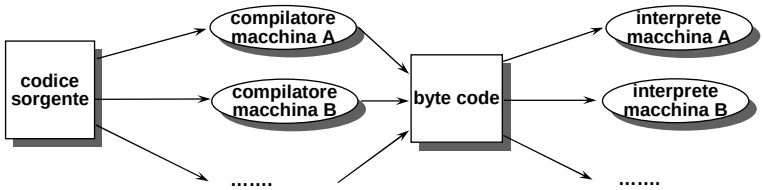
7

ESEMPIO

Sistema tradizionale (C, C++, Pascal, …)



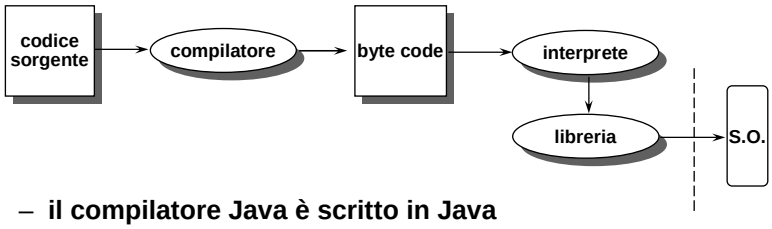
Sistema ANDF (Java)



8

PORTABILE

- Il sistema Java (compilatore + interprete + librerie run-time) è facilmente portabile su piattaforme diverse



  - **il compilatore Java è scritto in Java**
  - **l'ambiente run-time è scritto in ANSI C con interfacce standard (POSIX) verso il sistema operativo**
  - **nessuna "implementation dependency"**

9

ROBUSTO

- Controlli estensivi a compile-time e a run-time, per rilevare gli errori quanto prima possibile (es.: type checking)
- Per questo, le caratteristiche insicure di C e C++ sono rimosse:
  - Nessuna gestione esplicita dei puntatori (no aritmetica dei puntatori, no malloc e free esplicite, …)
  - Gestione della memoria con garbage collection
  - Array e stringhe "veri"
- Verifica del byte-code a load-time

10

DISTRIBUITO

- Pensato per essere eseguito in rete
- L'ambiente run-time incorpora funzioni di rete sia di basso livello (TCP/UDP)
  che di alto livello (HTTP)
- La rete è facilmente accessibile (come i file locali)

11

SICURO

- L'ambiente di esecuzione si protegge da bytecode potenzialmente "ostile"

- Esempi:
  - il bytecode viene verificato prima dell'interpretazione ("theorem prover"), in modo da essere certi di alcune sue caratteristiche
  - gli indirizzamenti alla memoria nel bytecode sono risolti sotto il controllo dell'interprete

12

DINAMICO

- Il codice è eseguibile anche in assenza di alcuni moduli:
- … le classi necessarie per la esecuzione di un programma Java possono essere caricate e collegate dinamicamente quando servono

Esempio: nuove release di moduli caricabili automaticamente dalla rete quando servono

13

CONCORRENTE

- Multithreading parte integrante del linguaggio:
  - Applicazioni interattive più facili a scriversi
  - Migliore "reattività"

Esempio: caricamento asincrono di immagini nei browser di rete riduce i tempi di attesa

14

RICCO

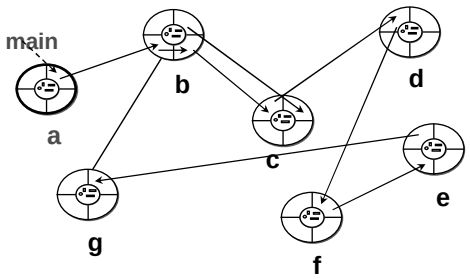
La Standard Library Java contiene una ricca collezione di classi e di metodi preconfezionati:

- Language support
- Utilities
- Input/output
- Networking
- Abstract Window Toolkit (AWT)

15

STRUTTURA DI UN'APPLICAZIONE JAVA

- Un'applicazione JAVA consiste di varie classi
- Almeno una classe deve avere un metodo statico e pubblico di nome **main**



16

HELLO WORLD

Hello.java

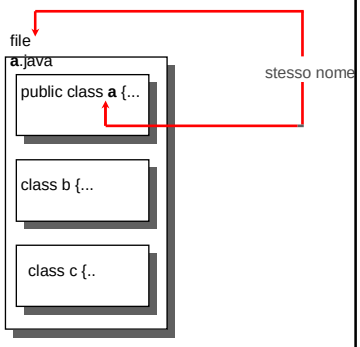
```
class Hello {
    public static void main (String args [])
    {
        System.out.println("Hello World!");
    }
}
```

obbligatorio in questa forma

17

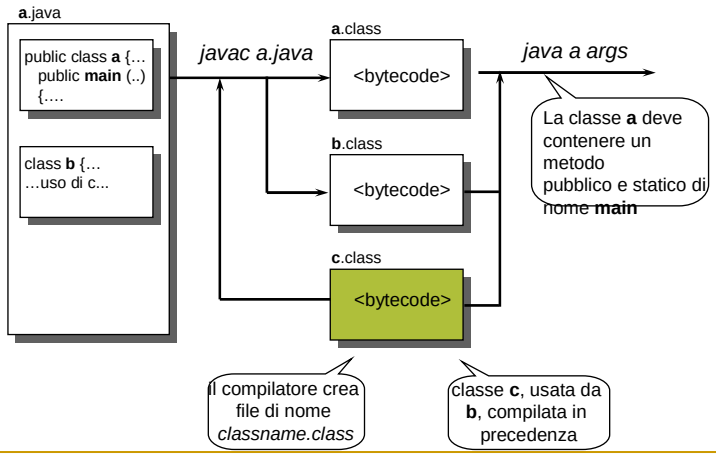
UNITA' DI COMPILAZIONE

- Il sorgente di un'applicazione consiste di uno o più file ("unità di compilazione")
- Ogni file contiene una o più dichiarazioni di classi (o di interfacce), di cui al più una dichiarata ***public***
- Il nome del file deve essere uguale a quello della sua classe ***public***, con estensione ***.java***:



18

COMPILAZIONE ED ESECUZIONE



19

Un altro esempio

```
class StringApp {
   static public void main(String args[])
   {
       // creo una stringa con valore iniziale Salve
       String s=new String("Salve");
       // creo una stringa con valore iniziale " a
   tutti"
       String s1=new String(" a tutti");
       // modifico s concatenando s1
       s=s.concat(s1);
       // trasformo s in MAISCOLO
       s=s.toUppercase();
       // Stampo a video la stringa
       System.out.println(s);
   } // chiude main
} // chiude classe
```

COMMENTI

Tre diversi stili:

- **/* Commento tradizionale, eventualmente su più linee, non nidificato */**
- **// Commento su di una sola linea**
- **/** Commento di documentazione */**

21

IDENTIFICATORI

- Sono composti da “lettere” e “cifre”, e devono iniziare con una “lettera”
- Possono essere di qualsiasi lunghezza
- Attenzione:
  - _ e *$* sono considerate lettere
  - Maiuscole e minuscole sono considerate lettere diverse

22

PAROLE CHIAVE

Le seguenti keywords non possono essere usate come identificatori:

abstract	***double***	***int***	***super***
boolean	***else***	***interface***	***switch***
break	***extends***	***long***	***synchronized***
byte	***final***	***native***	***this***
case	***finally***	***new***	***throw***
catch	***float***	***package***	***throws***
char	***for***	***private***	***transient***
class	***(goto)***	***protected***	***try***
(const)	***if***	***public***	***void***
continue	***implements***	***return***	***volatile***
default	***import***	***short***	***while***
do	***instanceof***	***static***	

Note: - *const* e *goto* sono riservate, ma non usate
- anche i letterali *null, true, false* sono riservati

23

TIPI: SINTESI

TIPI			KEYWORD	NOTE
Primitivi	boolean:		**boolean**	true, false
	numerici	interi	**byte** **short** **int** **long** **char**	8 bit interi in compl. a 2 16 bit 32 bit 64 bit 16 bit Unicode
		floating-point	**float** **double**	32 bit IEEE 754 64 bit
Reference	classi		**class**	
	interfacce		**interface**	
	array			
Null				

24

TIPI PRIMITIVI: ESEMPI

- NON SONO CLASSI
- Dichiarazioni: ***int i, float f;***
- Inizializzazioni: ***double d = 3.14;***
- Espressioni: ***i + 5***
- Assegnamenti: ***i = j + 5;***

25

CLASSI

dichiarazione di classe:

```
class Automobile {
        <attributi>
        <metodi>
}
```

dichiarazione di oggetto:

```
Automobile a;
```

creazione di oggetto:

```
a = new Automobile();
```

uso dell'oggetto:

```
… a.attributo …     … a.metodo(…) ...
```

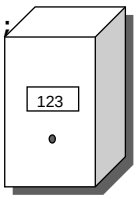
26

CLASSI: ESEMPIO

```
class Safe {
   public   int     doorNumber  = 123;
   private  boolean locked      = true;
   private  int     combination = 456;

   public boolean isLocked() {
        return (locked)
   }
   public void unLock (int thisCombination) {
        if (thisCombination == combination) unLock();
   }
   private void unLock() {
        locked = false;
   }
   private void setCombination(int setting) {
        combination = setting;
   }
   Safe () { }        /* costruttore */
   Safe (int door) {  /* altro costruttore (overloading) */
        doorNumber = door;
        setCombination(doorNumber);
   }
}
```

```
Safe sesamo;
sesamo = new Safe ();
Safe simsalabin;
simsalabin = new Safe
(1927);
...
sesamo.doorNumber=456;
sesamo.unLock(1997);
```



27

INTERFACCE

- In sostanza, sono liste di metodi di cui non è definita l'implementazione, ma solo l'interfaccia
- La loro implementazione deve essere realizzata da una classe

28

ESEMPIO

```
public interface AudioClip {
    void play ();       /* avvia l'esecuzione */
    void loop ();       /* esegui ciclicamente una audio clip */
    void stop ();       /* interrompi l'esecuzione */ }

class MyAudio implements AudioClip {
    void play () { <codice che implementa play> }
    void loop () { <codice che implementa loop> }
    void stop () { <codice che implementa stop> } }

class YourAudio implements AudioClip { <codice> }
```

I metodi possono essere *public* o *abstract*, gli attributi *public*, *static* o *final*

29

ARRAY

- In Java gli array sono oggetti
- I componenti di un array:
  - sono tutti dello stesso tipo
  - possono essere di tipo primitivo o reference (inclusi altri array)
  - sono indicizzati con ***int*** (indice primo elemento: 0), con controllo di validità degli indici a run-time
- Esempio:

```
int[ ] a;  /* dichiarazione */
           /* anche int a[ ]; */
a = new int[3]; /* creazione */
a[0] = 0;  /* uso */
```

30

ARRAY (segue)

- La lunghezza di un array è fissata al momento della sua creazione, e non può essere cambiata…
- ... ma si può assegnare un nuovo array di diversa lunghezza all'array reference:

int[ ] a = new int[3];

a = new int[5];

31

ARRAY DI ARRAY

```
short [ ] [ ] a;          /* array di array di short */
short a[ ] [ ];           /* equivalente */

a = new short [3][2];

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

32

ARRAY DI OGGETTI

```
class Automobile {
   public String targa;
   public int velocità;
   ....
}
....
Automobile[ ]  a=new Automobile[3];
```

33

STRUTTURE DI CONTROLLO: SINTESI

Sequenza		
Selezione	***if*** ***switch***	
Iterazione	***for*** ***while*** ***do-while***	
Salto	***break*** ***continue*** ***return***	**uscita da un blocco** **continua un loop** ***uscita*** **da un metodo**
Gestione eccezioni	***try-catch*** ***finally-throw***	

34

SINTASSI

```
if ( condition )
    statement;
[else
    statement; ]
```

```
while ( condition )
    statement;
```

```
do
    statement;
while ( condition );
```

```
switch ( intexpr ) {
      case intexpr : statement;
     [case intexpr : statement;
            ...
      default : statement; ]
}
```

```
for ( init; condition; increment )
      statement;
```

35

GESTIONE ECCEZIONI

- È una struttura di controllo che permette la gestione di condizioni di errore senza complicare la struttura del codice
- Quando si rileva una condizione anomale (eccezione), essa viene segnalata (***throw***), e il controllo viene automaticamente trasferito al gestore della eccezione, che la tratta

36

DICHIARAZIONE DI CLASSI

[Modifiers] *class* ***ClassName***
[*extends* ***SuperClassName]***
[*implements* ***InterfaceName [, InterfaceName] ...]***
{ ***ClassBody*** *}*

Modifiers	*abstract, final, public,* ***...***
extends	**la classe è sottoclasse di un'altra**
implements	**la classe realizza una o più interfacce**
ClassBody	**attributi e metodi della classe**

37

ESEMPIO

Dichiarazione di una classe MyClass

```
public class MyClass {
    int i;                              /* attributo */
   public void Add_to_i (int j) {       /* metodo */
        i = i+j;
    }
    public MyClass () {          /* metodo costruttore: ha lo
        i = 10;                     stesso nome della classe */
    }
}
```

Creazione e uso di una istanza di *MyClass*

```
MyClass mc;              /* dichiarazione, non creazione */
mc = new MyClass();      /* creazione: l'attributo i vale 10 */
mc.i++;                  /* ora i vale 11 */
mc.Add_to_i(10);         /* e ora i vale 21 */
```

38

DICHIARAZIONE DI ATTRIBUTI

[Modifiers] Type VariableDeclarator [,VariableDeclarator]… ;

Esempio: **static int i = 5, j = 7, a[];**

Modifiers	***static, public, protected,private, final, ...***
Type	tipo (primitivo o reference)
VariableDeclarator	identificatore (anche di array), con eventuale inizializzazione

39

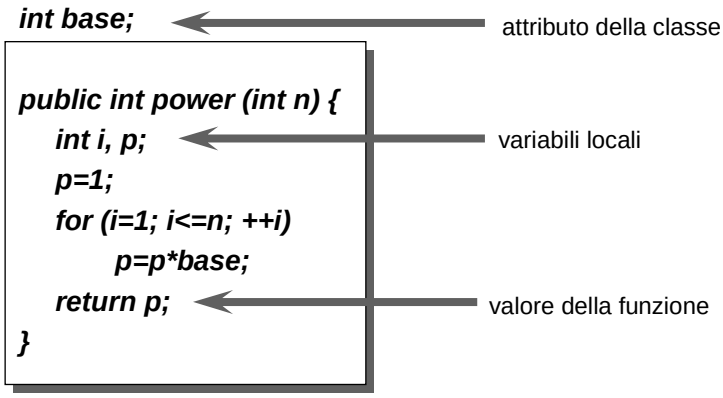
DICHIARAZIONE DI METODI

```
[Modifiers] ReturnType MethodName  (ParameterList)
        [throws ClassType [,ClassType] … ]
{
        MethodBody
}
```

Modifiers	**static, public,protected,private, abstract, final,native, ...**
ReturnType	**può essere un tipo primitivo o reference, o *void* Deve essere sempre specificato ma non per i costruttori !!!**
ParameterList	**un metodo ha sempre un numero fisso di argomenti**

40

ESEMPIO



41

IL MODIFICATORE static

- **Attributi e metodi dichiarati *static*,** sono associati alla classe e non a una particolare istanza

```
class MyClass {
        static int a;
        …
        static void MyMethod() {
                a = a+1;
        }
}
```

- **Pertanto: esiste una sola copia di un attributo statico, condiviso da tutte le istanze della classe; non occorre istanziare un oggetto per usare un attributo statico; metodi statici possono accedere solo ad attributi statici della classe**
- **Sono qualificati con il nome della classe, e non della istanza**

```
MyClass.a =MyClass.a + 1;
```

42

ESEMPIO

```
class Safe {
    static int LastSerialNumber;
    public int SerialNumber;
    int SetCombination;
    ....
    Safe(int Combination) {
        SerialNumber = LastSerialNumber++;
        SetCombination = Combination;
    }
    public static StartNewSeriesFrom (int Start) {
        LastSerialNumber = Start;
    }
}
....
Safe.LastSerialNumber(100000);    /* inizializza il numero di serie */
Safe MySafe = new Safe(23456);    /* crea una Safe di SerialNumber = 100001
                                     e combinazione 23456 */
    ... MySafe.SerialNumber ....
```

Il costruttore Safe fa sì che ogni istanza di Safe abbia un SerialNumber unico

43

VARIABILI LOCALI

- **Le variabili locali a un metodo:**
  - **sono visibili solo dal corpo del metodo**
  - **vengono allocate (nello stack di run-time) alla chiamata e deallocate all'uscita del metodo**
  - **non vengono inizializzate automaticamente (diversamente dai campi di una classe)**
- **Non si può accedere a una variabile a cui non si sia prima assegnato un valore (e viene segnalato in compilazione !)**

Esempio:

```
int  i;
if ( cond ) { i = 55; ... }
i++;     /* compile-time error */
```

44

COSTRUTTORI

- **Sono metodi che vengono chiamati quando si crea una istanza di una classe; non ritornano alcun valore**
- **Se non definisco alcun costruttore, per default viene fornito il costruttore vuoto:**

```
class MyClass {
        ...
        MyClass() { }
}
```

- **Possono essere overloaded**
- **non ha senso che siano *static***

ESEMPIO: OVERLOADING

Dichiarazione di una classe MyNewClass

```
public class MyNewClass {
    int i;
    public MyNewClass () {
        i = 10;
    }
    public MyNewClass (int j) {
        i = j;

    }   .....
}
```

Overloading: *metodi omonimi devono essere distinguibili per la quantità e/o per il tipo degli argomenti*

Creazione di una istanza di *MyNewClass*

```
MyNewClass mc0, mc1;          /* dichiarazione, non creazione */
mc0 = new MyNewClass();       /* creazione: l'attributo i vale 10 */
mc0.i++;                      /* ora i vale 11 */
mc1= new MyNewClass(20);      /* creazione: l'attributo i vale 20 */
mc1.i++;                      /* ora i vale 21 */
```

DISTRUZIONE DI OGGETTI

- **Java non supporta distruttori di oggetti: gli oggetti non possono essere distrutti esplicitamente**
- **Un oggetto privo di riferimenti incidenti non è più accessibile, e la memoria che esso occupa può essere "riciclata" dal** garbage collector**, che opera in un thread indipendente a bassa priorità**

```
String s;                  /* dichiarazione, non creazione */
s = new String ("abc");    /* creazione: s punta a "abc" */
s = "def";                 /* "abc" non è più puntata da s */
```

47

SOTTOCLASSI

- **In Java, una classe può estendere una sola altra classe (**ereditarietà singola**)**
- **Una sottoclasse eredita gli attributi ed i metodi della sua superclasse che non siano dichiarati *private* e che non siano costruttori, e li può usare come se fossero dichiarati al suo interno**
- **Un oggetto di una sottoclasse può essere usato ovunque può essere usato un oggetto della sua superclasse**

48

ESEMPIO

```
class Animale {
    float Peso;
    …
    void Mangia () { … }
    …
}
class Mammifero extends Animale {
    int BattitoCardiaco;
    // eredita Peso
    …
    void Respira() { … }
    // eredita mangia
    …
}
class Gatto extends Mammifero {
    // eredita BattitoCardiaco, Peso, Mangia, Respira
    void FaLeFusa() { … }
}
```

```
Gatto Fufi = new Gatto();
Animale creatura = Fufi;
```

lecito, perchè "gatto" è una sottoclasse di "animale"

49

"COME" ESTENDERE UNA CLASSE

- **Attenzione alla differenza fra la relazione "**è un**" e la relazione "**ha un**" : solo la prima può definire una sottoclasse**

Esempio:

Un cerchio può essere definito mediante il suo centro e il suo raggio, ma sarebbe scorretto definire una classe cerchio come sottoclasse di punto:

```
class punto {
   double x; int y;
   …
}
```

```
class cerchio extends punto {
   double raggio;
   …
}
```

50

SHADOWING E OVERRIDING

- shadowing — **un attributo di una sottoclasse può "nascondere" un attributo omonimo di una sua superclasse**
- overriding — **un metodo di una sottoclasse può "sovrascrivere" un metodo di ugual "*signature*" ed ugual ReturnType di una sua superclasse**

Esempio:

```
class SuperClass {
        int i;
        void m(int k) {…}
        …       }
class Subclass extends Superclass {
        long i;
        void m(int n) {…}
        …       }
```

long i nasconde int i

sovrascrive

POLIMORFISMO

```
class Animale {
        …
        void Mangia () { … }
        …
}

class Mammifero extends Animale {
        …
}
class Gatto extends Mammifero {
        …
        void Mangia() { … }
        …
}
```

```
Gatto Fufi = new Gatto();
Animale creatura = Fufi;
creatura.Mangia();
```

viene eseguito il metodo Mangia della classe Gatto!

(**binding dinamico**: il metodo da chiamare viene selezionato a run-time)

LA KEYWORD super

- **per denotare un attributo nascosto (shadowed) o un metodo overridden (non *static*), si può usare la keyword *super***

Esempio:

```
class SuperClass {
        int i;
        …
}

class Subclass extends SuperClass {
        long i;
        …
        i = super.i +1;
        …
}
```

53

COSTRUTTORI NELLE SOTTOCLASSI

Nel costruttore di una sottoclasse, è corretto, come prima cosa, chiamare il costruttore della superclasse

Esempio:

```
class SubClass extends SuperClass {
        SubClass() {
                super();
                <altre operazioni>
        }
}
```

54

IL MODIFICATORE abstract

- **Una classe è dichiarata *abstract* quando contiene almeno un metodo *abstract* (cioè senza body)**
- **Una classe *abstract* non può essere istanziata: occorre sovrascrivere tutti i metodi *abstract* in una sottoclasse, e istanziare la sottoclasse**

Esempio:

```
abstract class a {
        ...
        abstract int m(int k);
}
class b extends a {
        ...
        int m(int n) { … }
}
```

sovrascrive, fornendo la implementazione del metodo

55

IL MODIFICATORE final

final ha tre significati diversi:

- campo ***final*** **non può essere modificato: è un campo costante (deve essere inizializzato)**
  Esempio: *final int i = 5;*
- metodo ***final*** **non può essere sovrascritto**
- classe ***final*** **non può avere sottoclassi (quindi i suoi metodi sono implicitamente *final*)**

56

METODI final: ESEMPIO

```
class Password {
   private int passwd;
   final boolean validatePassword(String s) {
   .....
   }
...
}
```

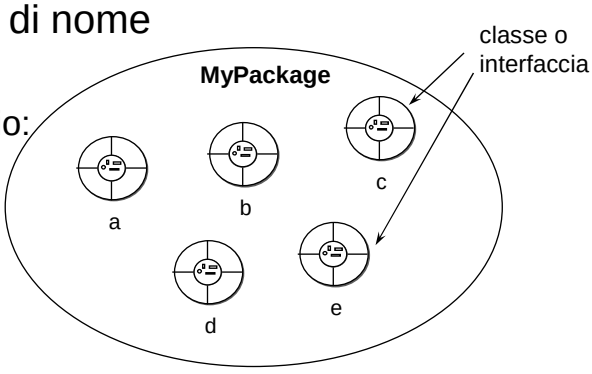
se non fosse *final* ,
potrebbe essere sovrascritto !

57

PACKAGES

- Più classi o interfacce interrelate possono essere riunite in un ***package,*** dotato di nome

Esempio:



58

NOMI DI CLASSI

- Il nome di una classe (o di una interfaccia) va sempre qualificato con il nome del package a cui appartiene, tranne quando viene usato all'interno dello stesso package

Esempio:

aaa.bbb.ccc.MyClass

all'interno di aaa.bbb.ccc basta questo

59

NOMI DI PACKAGE

- Il nome di un package può essere composto da più identificatori separati da ".":

utilities.internet

- Per evitare che package di produttori diversi abbiano lo stesso nome, si suggerisce di far iniziare il nome del package con il dominio Internet del produttore (invertito, e con il nome di più alto livello tutto maiuscolo):

it.unina.dis.utilities.internet

nome di dominio

60

PACKAGES STANDARD

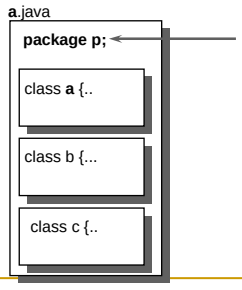
java.lang	classi base del linguaggio (*Object, Thread, Throwable, System, String, Math*, wrapper classes, …)
java.io	classi di I/O (*FileInputStream, FileOutputStream*,)
java.util	classi di utilità (*Date, Random*, …)
java.net	classi di supporto alle applicazioni di rete (socket, URL, …)
java.applet	classe Applet, …
java.awt	Abstract Windowing Toolkit

61

LO STATEMENT package

- Specifica che le classi che seguono apparten-gono a un certo package
- Deve apparire (una sola volta) all'inizio di una unità di compilazione

Esempio:



le classi a,b,c appartengono al package p

Se lo statement è omesso, la classe appartiene al package anonimo

62

LO STATEMENT import

- Per evitare di usare sempre nomi completi di classi, si può usare lo statement ***import***

Esempio:

```
class MyClass  {
   java.util.Date today;
   …
}
```



```
import java.util.Date;
class MyClass {
   Date today;
   …
}
```

Note:

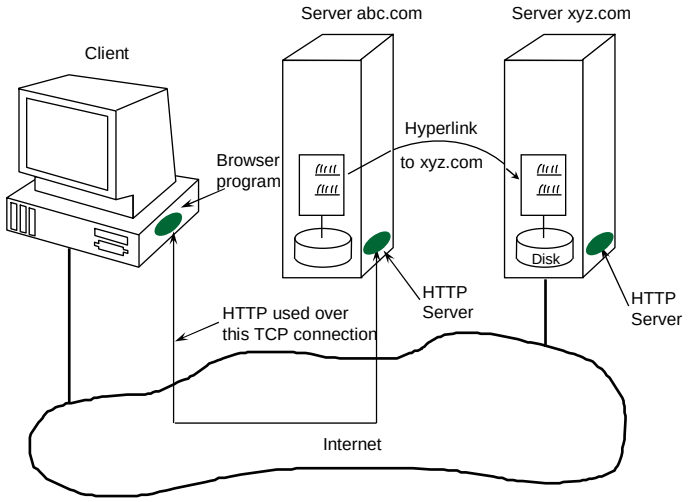
- ***import java.util.**** **importa tutte le classi del package *java.util***
- ***java.lang.**** **è sempre importato implicitamente**

63

JAVA E INTERNET

64

WEB BROWSER E SERVER



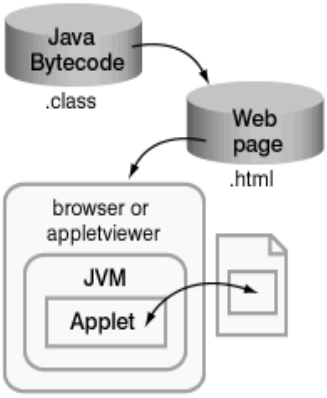
65

APPLET JAVA

- Sono programmi Java riferiti da link in pagine HTML
- Vengono caricati e interpretati dal browser (se questo supporta Java)
- Permettono di “animare” le pagine web

66

APPLET JAVA (2)



- Una applet Java compilata può essere incorporata come oggetto in una pagina HTML
- Il codice della applet (file .class) è riferito all'interno del codice HTML mediante il tag APPLET
- L'esecuzione della applet è affidata ad una Java Virtual Machine implementata nel browser
- Il testing delle applet può essere condotto mediante l'utility *appletviewer*

HTML con APPLET: ESEMPIO

```
<HTML>
<HEAD>
   <TITLE> Esempio </TITLE>
</HEAD>
<BODY>
   <P>
   Testo
   <APPLET CODE="HelloWorld.class WIDTH=300 HEIGHT=80>
   Qui c'è un applet che ti saluta
   </APPLET>
</BODY>
</HTML>
```

testo visualizzato se il browser non supporta le applet Java

68

Domande?