Programmazione ad Oggetti

Sintassi Java

A.A. 2022/2023

Docente: Prof. Salvatore D'Angelo
Email: salvatore.dangelo@unicampania.it

Università degli Studi della Campania *Luigi Vanvitelli*
Dipartimento di Ingegneria

Commenti java

I commenti possono essere inseriti nel codice Java in vari modi:

```
/* commento su una o più righe */
// commento sulla singola riga
/** commenti di documentazione */
```

Esiste la possibilità di generare documentazione in formato HTML da un codice sorgente Java con la funzionalità javadoc messa a disposizione dal Java Developer's Kit (JDK)

Identificatori java

Gli identificatori possono iniziare con una:

- lettera
- underscore (_)
- dollaro($)
- una sequenza di caratteri che può contenere dei numeri

Java fa distinzione tra maiuscole e minuscole (case sensitivity)

Le parole chiave sono identificatori che hanno un significato speciale e possono essere usate solo nel modo opportuno

Parole chiave java

Parole chiave comuni tra i linguaggi Java e C++

break	case	catch	char
class	const	continue	default
do	double	else	float
for	goto	if	int
long	new	private	protected
public	return	short	static
switch	this	throw	try
void	while	false	true
volatile			

Parole chiave del linguaggio Java

abstract	boolean	byte
byvalue	extends	final
finally	implements	import
instanceof	interface	native
null	package	super
synchronized	throws	

Parole chiave java

Abstract	**else**	**interface**
boolean	**extends**	**long**
break	**false**[1]	**native**
byte	**final**	**new**
case	**finally**	**null**[1]
catch	**float**	**package**
char	**for**	**private**
class	**goto**[2]	**protected**
const[2]	**if**	**public**
continue	**implements**	**return**
default	**import**	**short**
do	**instanceof**	**static**
double	**int**	**strictfp**

1. parole riservate; **2**. parole al momento non utilizzate

Tipi e variabili

Un tipo di dato identifica un insieme di valori e di operazioni applicabili a tali valori

Le variabili sono contenitori di valori di un qualche tipo. Ogni variabile è un nome che indica una locazione di memoria usata per contenere un valore che può cambiare nel tempo

Le costanti sono simili alle variabili ma possono contenere un unico valore per tutta la durata della loro esistenza

Java è un linguaggio fortemente tipizzato:
"è impossibile assegnare ad una variabile un valore che sia incompatibile col suo tipo"

Java: tipi e variabili

Il linguaggio Java utilizza quattro tipi di dati semplici:

- integer
- floating point
- character
- Boolean

Esempio: ***int*** *x;*

```
x = 32 * 54;
final int Posti=27; // Costante
```

Per utilizzare una variabile di un tipo predefinito occorre

- dichiararla
- inizializzarla

Java: tipi di dati

Interi	
bit	**tipo**
8 (±)	**byte**
16 (±)	**short**
32 (±)	**int**
64 (±)	**long**

Virgola mobile (reali)	
bit	**tipo**
32	**float**
64	**double**

(*) tipo **char** unicode a 16 bit compatibile con i numeri interi

	Tipo
Carattere	**char** (*) (16 bit)
Logico	**boolean**

- Il tipo boolean ha due valori:
  - true
  - false

Java: dichiarazioni

	Esempi
Variabili	**int** n; **float** x; **char** c; **boolean** enunciato; **int** v[] = **new int**[3]; String parola;
Costanti	**final float** IVA = 0,20;
Librerie	**import awt.*;**

Java: tipi e variabili

Le variabili in Java sono valide soltanto dal punto in cui vengono dichiarate fino alla fine del blocco di istruzioni che le racchiude

Non è possibile dichiarare una variabile con lo stesso nome di un' altra contenuta in un blocco più esterno

Esempio:

```
class Scope {
    public static void main (String args[]) {
      int count = 100;
      for ( int i=0; i<3; i++) {
       System.out.println("Valore : " +i );
         int count = 2; //Compile time error...
                                  } // end for
            } // end main
} // end class
```

Conversioni e casting

A volte è necessario o utile convertire un valore di un tipo nel corrispondente valore di un altro tipo

In Java le conversioni di tipo possono avvenire in due modi

- conversioni durante un assegnamento (**casting implicito**) che sono ammissibili solo per le cosiddette conversioni larghe
- **casting esplicito**: operatore java specificato da un tipo racchiuso tra parentesi che viene messo davanti al valore da convertire

Non è possibile effettuare il casting tra i tipi interi e booleani

Conversioni e casting

Implicita (automatica)	*esempio*
A un tipo più 'capiente' viene assegnato un tipo meno 'capiente' **double ← float ← long ← int ← char ← short ← byte**	**int** i = 24; **long** n = i;

Esplicita (casting)	*esempio*
***Si indica di fronte alla variabile il nuovo tipo tra parentesi*: (nuovo_tipo) variabile;**	**long** n = 24; **int** i = (**int**) n;

Istruzioni di controllo flusso

- L'ordine in cui le istruzioni sono eseguite in un programma è detto **flusso di controllo** o di esecuzione
- Le istruzioni condizionali e i cicli permettono di controllare il flusso di esecuzione all'interno dei metodi
- In java appartengono a tale categoria le seguenti istruzioni:
  - **if, if-else**
  - **switch**
  - **for**
  - **while, do-while**

Costrutto If

- L'istruzione ***if*** permette al programma di decidere se eseguire o meno una determinata istruzione
- L'istruzione ***if-else*** permette al programma di eseguire una istruzione se si verifica una certa condizione e un'altra in caso contrario

if	**if** (condizione) istruzione;
if **else**	**if** (condizione) istruzione; **else** istruzione;
if **else** (blocco)	**if** (condizione) { istruzione; istruzione; …; } **else** {istruzione; istruzione; …; }

Esempio : if /else

```
int count = 5;
   if ( count < 0 ) {
    System.out.println(“Error: count value is
negative”);
   }
else {
    System.out.println(“Il valore di count è
=” + count );
   }
```

Switch

- Lo **switch** valuta una espressione per determinarne il valore e poi lo confronta con quello delle clausole **case**
- L'istruzione **break** usata alla fine di ogni case, serve per saltare alla fine dello switch stesso

switch

```
switch (i) // i variabile byte o short o int o char
{
    case 1: istruzione;
            break;
    case 2: { istruz; istruz; …; }
            break;
    case 3: { istruz; istruz; …; }
            break;
    default:
            { istruz; istruz; …; }
}
```

Cicli: for

- L' istruzione **for** è usata di solito quando un ciclo deve essere eseguito un numero prefissato di volte

for	**for** (int i=start; i<=stop; i++)
	{
	istruzione; istruzione; …;
	}

	for (int i=start; i<=stop; i--)
	{
	istruzione; istruzione; …;
	}

Nota: si possono usare le istruzioni ***break*** o ***continue*** per uscire dal ciclo o riprenderlo

Cicli condizionati: do while

- Una istruzione while permette al programma di eseguire più volte un blocco di istruzioni valutando ogni volta una condizione booleana
- Con il while il ciclo do esegue il suo corpo finchè la condizione non diventa falsa

do	**do** { istruzione; istruzione; …; } **while** (condizione);

while	**while** (condizione)
	{
	istruzione; istruzione; …;
	}

Operatori

Operatori

- Gli operatori di Java sono caratteri speciali per istruire il compilatore sull'operazione che deve compiere con alcuni operandi
- Java ha quarantaquattro operatori differenti suddivisi in quattro gruppi:
  - aritmetici
  - bitwise
  - relazionali
  - logici

Operatori aritmetici

L'operatore di incremento somma 1 a qualsiasi valore intero o in virgola mobile mentre quello di decremento sottrae

Operatore	*Simbolo*
Addizione	**+**
Sottrazione	**-**
Moltiplicazione	*****
Divisione	**/**
Modulo	**%**
Incremento	**++n; n++**
Decremento	**--n; n--**

Operatori: forme abbreviate

Java offre vari operatori che permettono di abbreviare le espressioni di assegnazione

Operazione	*Forma abbreviata*	*equivale a*
Addizione	**a += b**	**a = a + b**
Sottrazione	**a -= b**	**a = a - b**
Moltiplicazione	**a *= b**	**a = a * b**
Divisione	**a /= b**	**a = a / b**

Operatori di assegnazione

Operatore	*Simbolo*
Di assegnazione	=
Uguale	==
Diverso	!=
Maggiore	>
Minore	<
Non maggiore	<=
Non minore	>=

Operatori logici

Gli operatori logici restituiscono un valore booleano e sono spesso usati per costruire condizioni complesse

Operatore	*Simbolo*
Negazione	**!**
Congiunzione	**&&**
Disgiunzione Inclusiva	**||**
Disgiunzione esclusiva	**^**

Caratteri speciali

Carattere	*Descrizione*
\n	A capo (new line)
\t	Tabulazione
\b	Cancella a sinistra (backspace)
\r	Inizio riga (carriage return)
\f	Avanzamento pagina (form feed)
****	Barra inversa
\”	Virgolette
\’	Apice

Java: Oggetti

Le variabili non semplici (cioé strutturate) o di un tipo non predefinito sono considerate oggetti

Gli oggetti vanno:

- dichiarati
- istanziati
- Inizializzati

Per l'istanziazione si usa la parola chiave ***new***
L'istanziazione produce l'effettiva allocazione in memoria dell'oggetto

Esempio:

String s=new String("Hello word");

Esempio Stringa 1

```
class TestStringa{
public static void main(String arg[])
    {
        String a=new String(“Nel mezzo del cammin”);
        String b=“di nostra vita”;

        a=a+b;
        System.out.println(a);
        System.out.println(a.substring(3));
        System.out.println(a.length());
    }
}
```

Output:

```
Nel mezzo del cammin di nostra vita
mezzo del cammin di nostra vita
33
```

Le Stringhe 1/3

In Java le stringhe sono oggetti appartenenti alla classe String (è un tipo particolare in java)

Costanti stringa vengono racchiusa tra ”
String s = “ciao a tutti”; *// viene creato implicitamente un oggetto di classe String*

La variabile s è un riferimento ad un oggetto di classe String e corrisponde a:
String s = new String(“ciao a tutti”);

In sostanza è il compilatore che nel primo caso istanzierà l’oggetto stringa “ciao a tutti” in memoria

Le Stringhe 2/3

La concatenazione di due stringhe in realtà genera automaticamente una nuova istanza di String il cui contenuto è costituito dai caratteri della prima e della seconda m essi insieme

String s = “ciao“+” a tutti”

- **Le due stringhe “ciao“ e ” a tutti” rappresentano due oggetti stringa in memoria.**
- **Grazie all’operatore di concatenazione viene creato un terzo oggetto stringa che verrà puntato da s.**
- **Le due stringhe “ciao“ e ” a tutti” Verranno cancellate dalla memoria.**

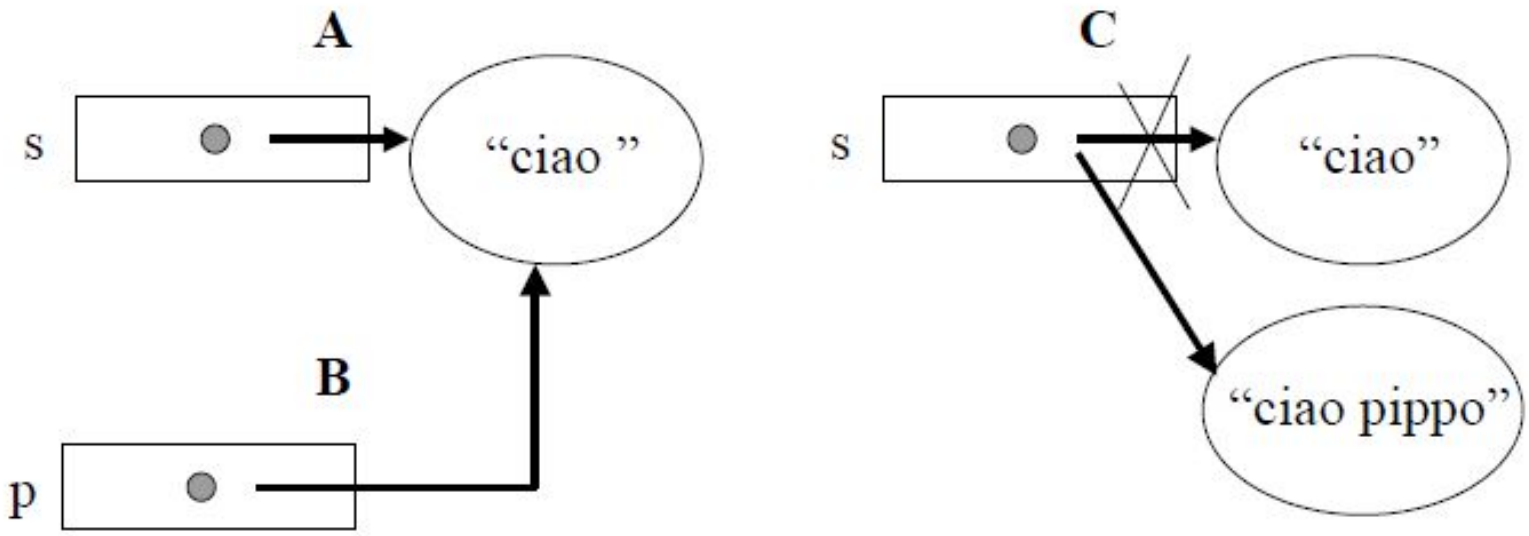
In Java le stringhe sono immutabili!

Quando viene creata una stringa il suo valore non può più essere cambiato

Esempio

String s = “ciao ”;
p=s;
s= s + “pippo”;

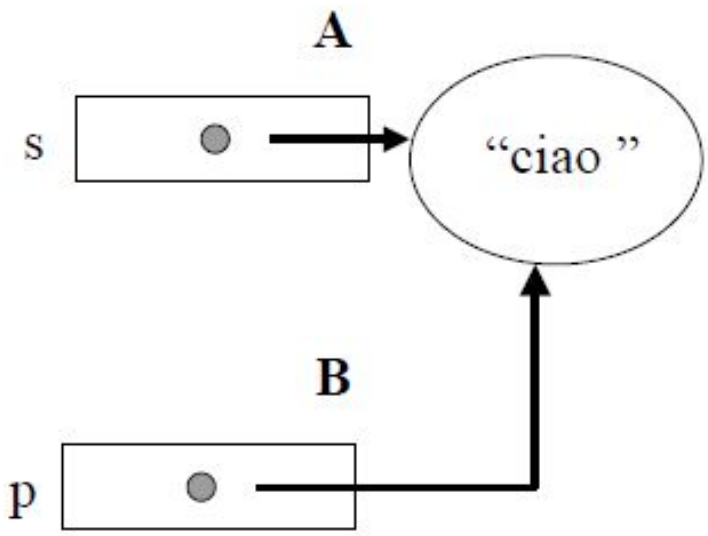
s==p????

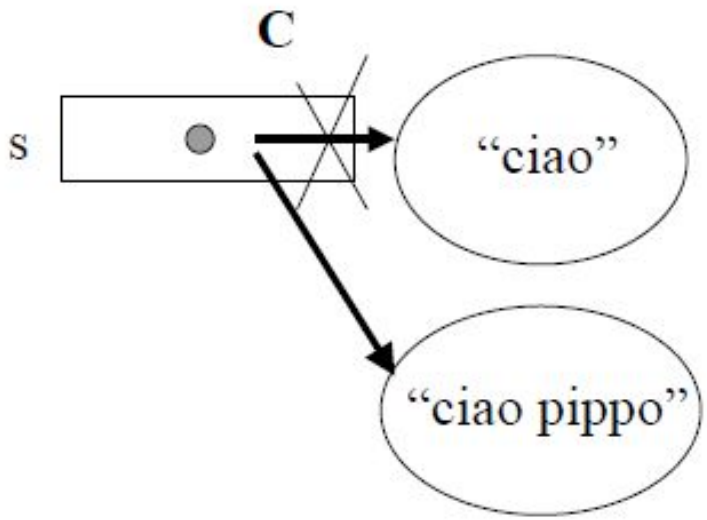


Esempio

```
String s = “ciao ”;
p=s;
s= s + “pippo”;

s==p????
```

Confronto i riferimenti
In alternativa s.equals(p);





Le Stringhe 2/3

```
String a=new String(“Nel mezzo  ”);
String b =new String(“del cammin”);
String c =a+b;
a =a+b;
```

Le istanze puntate da a e b non vengono cancellate

Le Stringhe 2/3

```
String a=new String(“Nel mezzo  ”);
String b =new String(“del cammin”);
String c =a+b;
String a =a+b;
```

Le istanze puntate da a e b non vengono cancellate

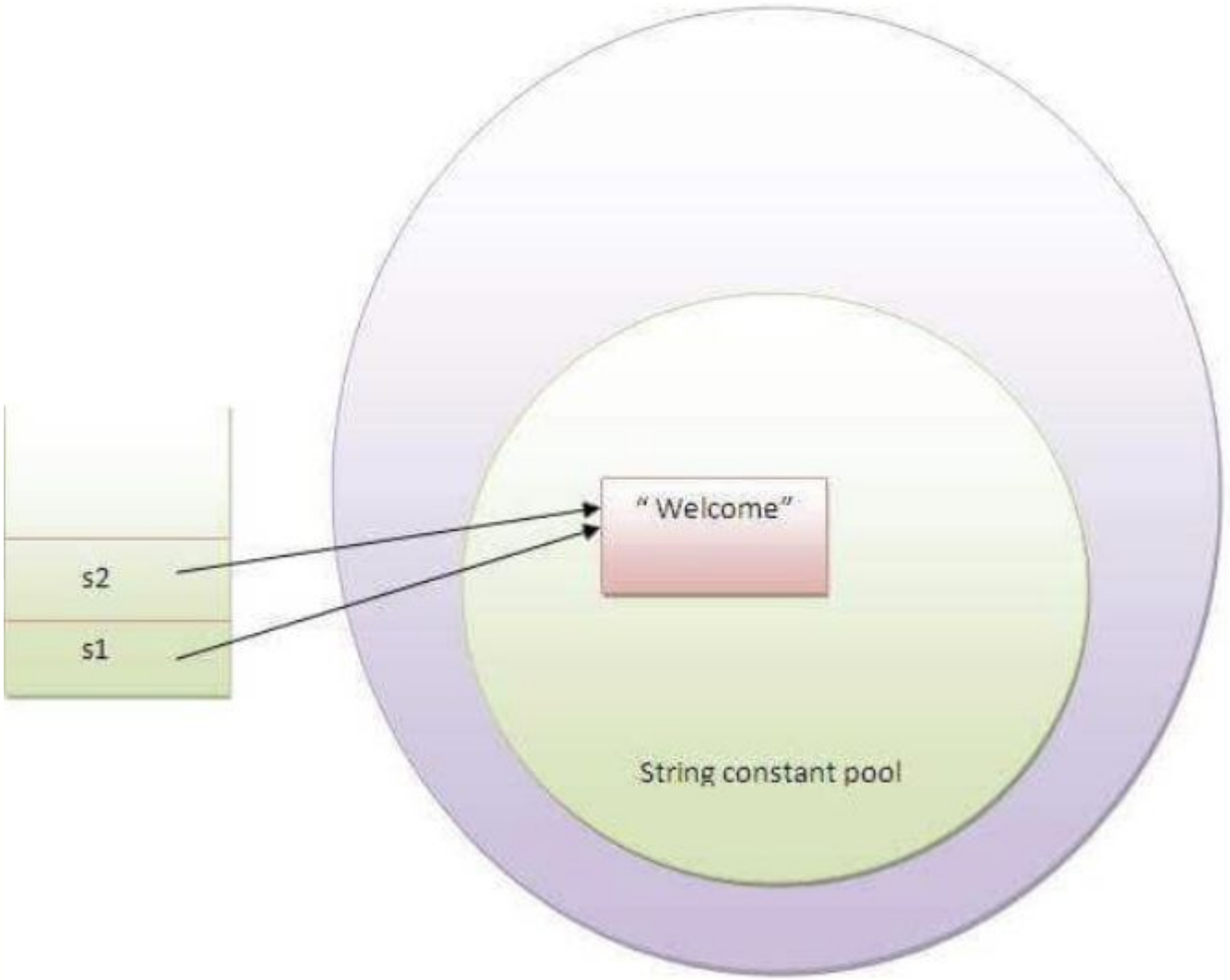
Esempio Stringa 2

```
class TestStringa{
public static void main(String arg[])
   {
      String a=new String("Nel mezzo del cammin");
      String e=new String("Nel mezzo del cammin");

       // b, c, d puntano fisicamente alla stessa stringa
      String b="di nostra vita";
      String c="di nostra vita";
      String d="di nostra vita";

      a=a+b;
      b=a+b;
      System.out.println(a);
      System.out.println(a.substring(3));
      System.out.println(a.length());
   }
}
```

Due stringhe puntate da due riferimenti differenti

Il compilatore si accorge che esiste già la stinga "di nostra vita", per cui non istanzierà una nuova stringa ma semplicemente farà puntare c e d alla stessa stringa puntata da b

```
String s1="Welcome";
String s2="Welcome";//It doesn't create a new instance
```



*Prova interi

```
class TestStringa{
public static void main(String arg[])
    {
        int a=6;
        int b=5;
        a=b;
        b=10;
        System.out.println(a);
        System.out.println(b);
}
}
```

*Riferimenti

- La dichiarazione di un oggetto equivale a dichiarare una reference
- La creazione dell'oggetto rende valida la reference e alloca memoria per l'oggetto
- Tutti gli oggetti devono essere creati

Array Java

In Java un *array* è un oggetto
Si possono dichiarare *array* di qualsiasi tipo
Esempio:

- *char s[ ]; // dichiarazione di array*
- *int [ ] array;*
- *int [ ] x, y[ ];*
- *int x[ ], y[ ];*

Gli *array* sono creati mediante la parola chiave ***new***, occorre indicare un tipo e un numero intero, non negativo, di elementi da allocare
Si possono creare array di *array*
Java controlla in modo rigoroso che il programmatore non tenti accidentalmente di memorizzare o far riferimento ad elementi dell'*array* tramite un indice non valido

Esempio Array

```
public class Prova{
   public static void main(String args[])
   {
      System.out.println("ciao");

      int v[] = new int[10];
                        v[1]=2;
                        …
      v[10]=9;  //Java effettua un controllo
sulle dimensioni e da un eccezione
      }
   }
```