



UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA
DIPARTIMENTO DI ECCELLENZA

Università di degli Studi di Salerno

Dipartimento di Informatica

Programmazione ad Oggetti

a.a. 2023-2024

Sintassi Java
Classe Stringa

Docente: Prof. Massimo Ficco

E-mail: *mficco@unisa.it*

Commenti java

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

- **// commento sulla singola riga**
- **/* commento su una o più righe */**
- **/** 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)



Identifieri java



Gli identifieri 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 identifieri 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

Abstract	else	interface
boolean	extends	long
break	false₁ ,	native
byte	final	new
case	finally	null₁
catch	float	package
char	for	private
class	goto₂	protected
const₂	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

<i>Interi</i>	
bit	tipo
8 (\pm)	byte
16 (\pm)	short
32 (\pm)	int
64 (\pm)	long

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

<i>Virgola mobile (reali)</i>	
bit	tipo
32	float
64	double

	<i>Tipo</i>
<i>Carattere</i>	char (*) (16 bit)
<i>Logico</i>	boolean

Il tipo boolean ha due valori:

- true
- false



Java: dichiarazioni

	<i>Esempi</i>
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

<i>Implicita (automatica)</i>	<i>esempio</i>
<p>A un tipo più ‘capiente’ viene assegnato un tipo meno ‘capiente’</p> <p>double ← float ← long ← int ← char ← short ← byte</p>	<code>int i = 24; long n = i;</code>

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



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

Esempio:

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

n.b.: NON C'è pericolo di confondere == e =



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 while	do { istruzione; istruzione; ...; } while (condizione);
	while (condizione) { istruzione; istruzione; ...; }



Operatori



Operatori aritmetici

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

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



Operatori: forme abbreviate

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

<i>Operazione</i>	<i>Forma abbreviata</i>	<i>equivale a</i>
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

<i>Operatore</i>	<i>Simbolo</i>
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

<i>Operatore</i>	<i>Simbolo</i>
Negazione	!
Congiunzione	&&
Disgiunzione Inclusiva	
Disgiunzione esclusiva	^



Caratteri speciali

<i>Carattere</i>	<i>Descrizione</i>
\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:

int s=new int(3);



Array



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
    }
}
```



La classe String

Stringhe e classi String

- Una stringa è una sequenza di caratteri Unicode racchiusa tra virgolette **"Massimo"**
- **""** rappresenta la stringa vuota
- Non esiste il tipo primitivo stringa, ma esiste una classe predefinita -> **le stringhe sono oggetti in Java**
- Le operazioni su stringhe sono realizzate mediante metodi della classe **String**



Le Stringhe

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

Costant stringa vengono racchiusa tra "

```
String s = "ciao a tutti"; // viene creato implicitamente un  
                           oggetto di classe String
```

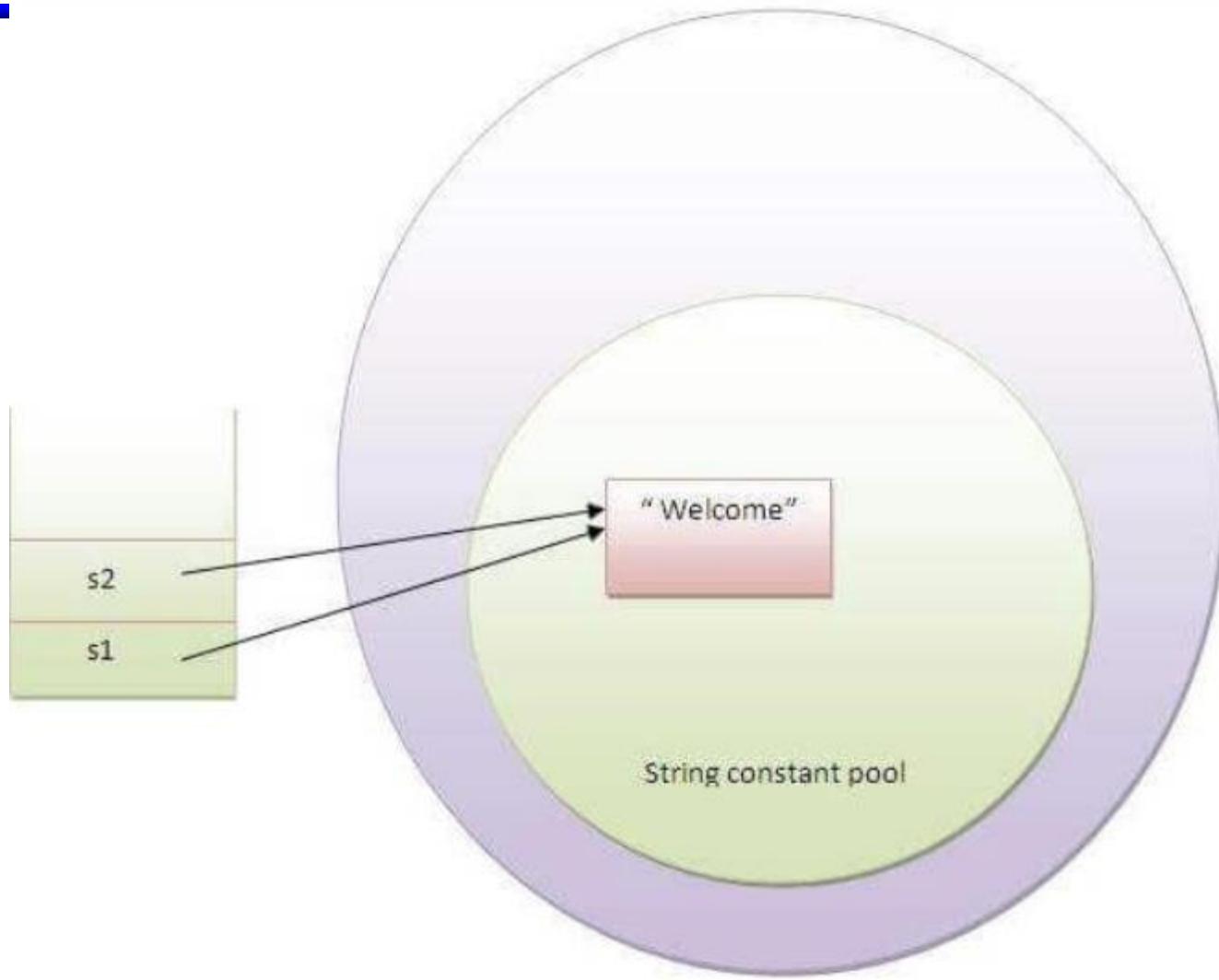
La variabile s è un riferimento ad un oggetto di classe **String** e corrisponde a:

- `s="ciao a tutti";`
- `String s = new String();`
- `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



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



Alcuni metodi della classe String

Il metodo **length()** restituisce la lunghezza della stringa:

```
String nome = "Massimo";
```

```
int i = nome.length() -> vale 7
```

```
"Massimo".length() -> vale 7
```

```
"".length() -> vale 0
```

Gli elementi della stringa vanno dalla posizione 0 a **length()-1**



Posizione nelle stringhe

Le posizioni dei caratteri in una stringa sono numerate a partire da 0

H	a	m	b	u	r	g	e	r
0	1	2	3	4	5	6	7	8



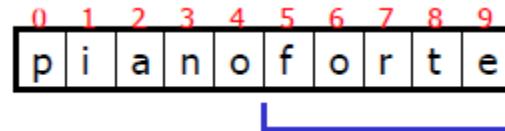
Alcuni metodi della classe String

Il metodo `String substring(int inizio)` restituisce un nuovo oggetto String che consiste una sottostringa della stringa su cui è invocato

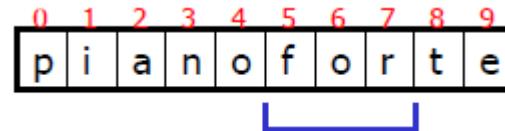
```
String s1, s2;
```

```
s1 = "pianoforte";
```

```
s2= s1.substring(5) -> restituisce la stringa "forte"
```



```
s2= s1.substring(5,8) -> restituisce la stringa "for"
```



Alcuni metodi della classe String

Il metodo `int indexOf(char c)` verifica se la stringa contiene il carattere c e restituisce la sua posizione:

```
s1 = "Massimo";
```

```
s1.indexOf('a') -> valore 1
```

```
s1.indexOf('p') -> valore -1
```

```
s1.indexOf('ssi') -> valore 2
```

Il metodo `boolean equals(string s)` permette di comparare due stringhe (non è corretto usare `==`).

```
s1 = "Massimo";
```

```
s2 = "Massimo";
```

```
System.out.println(s1.equals(s2)); -> Stampa true
```



Alcuni metodi della classe String

- ▶ public boolean equalsIgnoreCase (String str) - come equals, ignorando differenza tra maiuscole e minuscole;
- ▶ public int compareTo (String str) - confronta le stringhe rispetto all'ordinamento lessicografico



Alcuni metodi della classe String

Metodo `charAt(int p)` restituisce il carattere che occupa la posizione p.

`s1.charAt(0)` -> carattere 'M'
`s1.charAt(3)` -> carattere 's'

Il metodo `concat()`, a partire da due stringhe, restituisce una stringa il cui valore è dato dalla sequenza di caratteri della prima stringa seguito dalla sequenza di caratteri della seconda stringa → `String concat(String x)`

```
String s1, s2, s3;  
s1 = "Massimo";  
s2 = "Ficco";  
s3= s1.concat(s2) -> la stringa s3 varrà "MassimoFicco";
```



Operatore di concatenazione stringhe ‘+’

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"
```

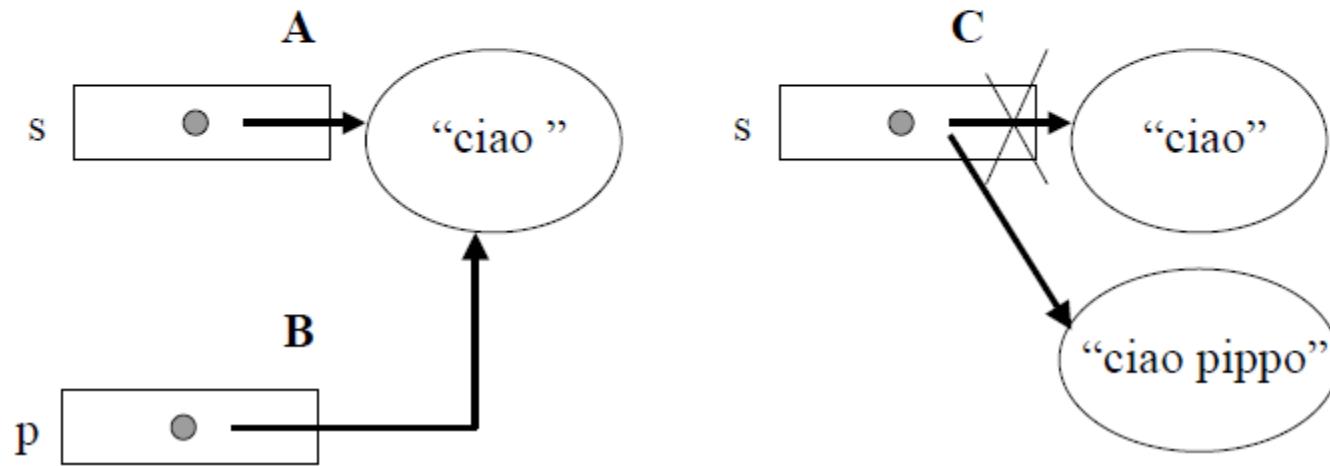
- 1) Le due stringhe “ciao“ e ” a tutti“ rappresentano due oggetti stringa in memoria.
- 2) Grazie all’operatore di concatenazione viene creato un terzo oggetto stringa che verrà puntato da s.
- 3) Le due stringhe “ciao“ e ” a tutti“ Verranno cancellate dalla memoria.



Esempio

```
String s = "ciao ";
p=s;
s= s + "pippo";
```

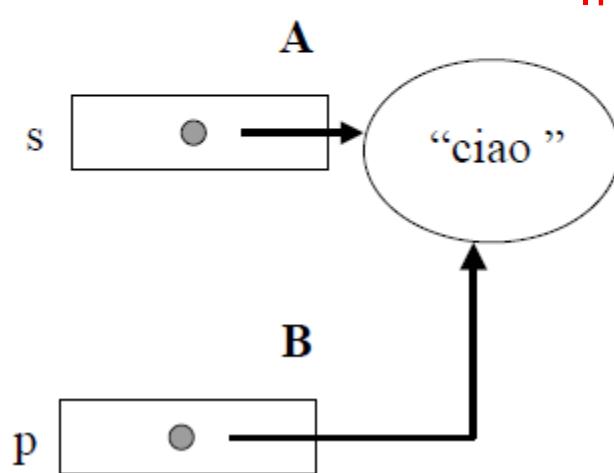
s==p????



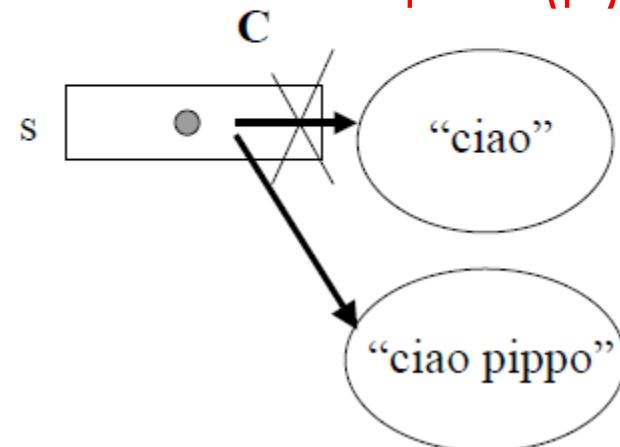
Esempio

```
String s = "ciao ";
p=s;
s= s + "pippo";
```

s==p????



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



Esempio

```
String a=new String("Nel mezzo ");
String b =new String("del cammin");
String c =a+b; //Le istanze puntate da a e b non
                vengono cancellate
String a =a+b;
```



Immutabilità delle Stringhe

In Java le stringhe sono immutabili!

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



Esempio Stringa

```
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 stringa "di nostra vita", per cui non istanzierà una nuova stringa ma semplicemente farà puntare c e d alla stessa stringa puntata da b



Output

Nel mezzo del cammin di nostra vita
mezzo del cammin di nostra vita

33



Esempio classe auto

```
class Auto{  
    public int cilindri=4;  
    public int speed=0;  
    public String targa;  
  
    public int getSpeed(){return speed;};  
    public void setSpeed(int s){speed= s;};  
    public int getCilindri(){return cilindri};  
  
    // Tre alternative ???  
    public void setTarga(){targa = "XF345PF"; }  
    public void setTarga(String s){ targa = s;}  
    public void setTarga(String s){ targa = new String(s); }  
}
```



Passaggio di parametri per Valore e Riferimento

1° Esempio- Scambio dei parametri (per valore)

```
public class P-Valore{
    static void cambiaString(String s)
    {
        s=s.concat(" casa");
        System.out.println("metodo: "+s); → metodo: ciao casa
    }

    public static void main(String args[])
    {
        String s="ciao"; → stringa costante
        System.out.println("main1: "+s); → main1: ciao
        cambiaString(s); → passaggio per valore
        System.out.println("main2: "+s); → main2: ciao
    }
}
```



2° Esempio- Scambio dei parametri (per riferimento)

```
class Auto{  
    public int cilindri=4;  
    public int speed=0;  
    public String targa;  
  
    public int getSpeed(){return speed;};  
    public void setSpeed(int s){speed= s;};  
    public int getCilindri(){return cilindri;};  
    public void setTarga(String s){targa = s;}  
}
```



2°Esempio- Scambio dei parametri (per riferimento)

```
public class Prova{
    static void accelera(Auto b)
    {
        b.setSpeed(b.speed+1);
        System.out.println("metodo: "+b.speed); → metodo: 1
    }

    public static void main(String args[])
    {
        Auto a =new Auto();
        System.out.println("main1: "+a.speed); → main1: 0
        accelera(a); → passaggio per riferimento
        System.out.println("main2: "+a.speed); → main2: 1
    }
}
```



3° Esempio- Scambio dei parametri di tipo String (per riferimento)

```
public class P-Riferimento{  
    static void cambiaString(String s)  
    {  
        s=s.concat(" casa");  
        System.out.println("metodo: "+s); → metodo: ciao casa  
    }  
  
    public static void main(String args[]){  
        String s= new String("ciao"); → Riferimento  
        System.out.println("main1: "+s); → main1: ciao  
        cambiaString(s); → passaggio per riferimento  
        System.out.println("main2: "+s); → main2: ciao casa  
    }  
}
```



3° Esempio- Scambio dei parametri di tipo String (per riferimento)

```
public class P-Riferimento{  
    static void cambiaString(String s)  
{  
        s=s.concat(" casa");  
        System.out.println("cambiaString: "+s);  
    }  
    public static void main(String[] args){  
        String s= new String("ciao"); → Riferimento  
        System.out.println("main1: "+s); → main1: ciao  
        cambiaString(s); → passaggio per riferimento  
        System.out.println("main2: "+s); → main2: ciao casa  
    }  
}
```

Non accade ciò che mi aspettavo, perchè le stringhe in java sono oggetti immutabili (rappresentano un'eccezione).



3° Esempio- Scambio dei parametri di tipo String (passaggio per riferimento)

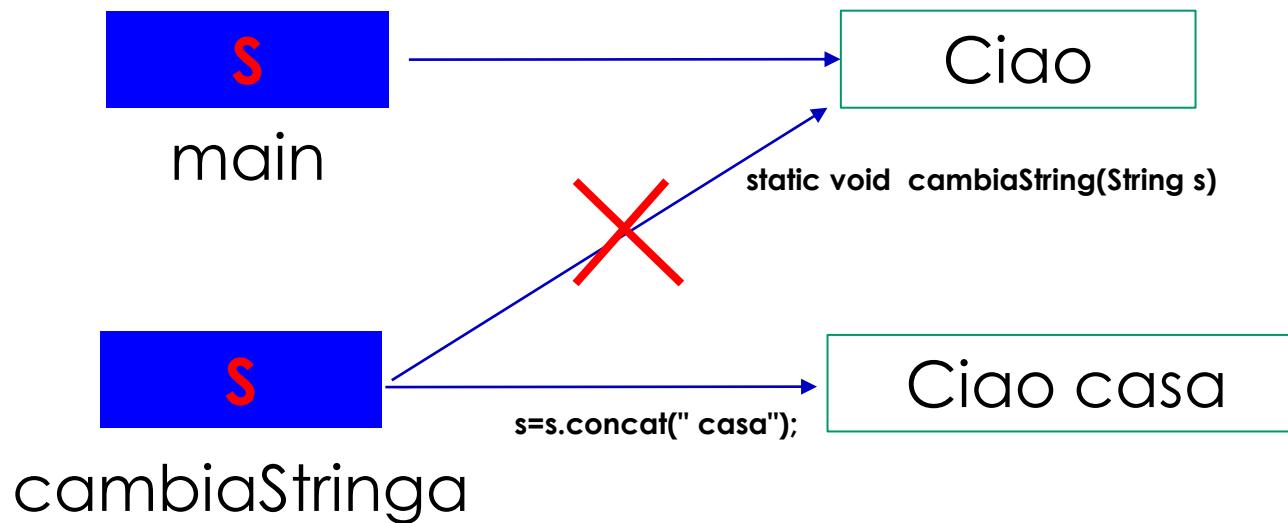
```
public class P-Riferimento{  
    static void cambiaString(String s)  
{  
    s=s.concat(" casa");  
    System.out.println("metodo: "+s); → metodo: ciao casa  
}  
}  
  
public static void main(String args[]){  
    String s= new String("ciao"); → Riferimento  
    System.out.println("main1: "+s); → main1: ciao  
    cambiaString(s); → passaggio per riferimento  
    System.out.println("main2: "+s); → main2: ciao casa  
}
```

La reference s rappresenta un parametro locale del metodo.

Questa è una nuova stringa (locale del metodo).



3° Esempio- Scambio dei parametri di tipo String (per riferimento)



3° Esempio- Scambio dei parametri di tipo String (per riferimento)

```
public class P-Riferimento{  
    static void cambiaString(String s)  
{  
    s=s.concat(" casa");  
    System.out.println("metodo: "+s); → metodo: ciao casa  
}  
  
public static void main(String args[]){  
    String s= new String("ciao"); → Riferimento  
    System.out.println("main1: "+s); → main1: ciao  
    cambiaString(s); → passaggio per riferimento  
  
    System.out.println("main2: "+s); → main2: ciao casa  
}
```



3° Esempio- Scambio dei parametri di tipo String (per riferimento)

```
public class P-Riferimento{  
    static void cambiaString(String s)  
    {  
        s=s.concat(" casa");  
        System.out.println("metodo: "+s); → metodo: ciao casa  
    }  
  
    public static void main(String args[]){  
        String s= new String("ciao"); → Riferimento  
        System.out.println("main1: "+s); → main1: ciao  
        cambiaString(s); → passaggio per riferimento  
        s=s.concat(" casa");  
        System.out.println("main2: "+s); → main2: ciao casa  
    }  
}
```



Class String

La classe String:

```
String str = new String();
String str2 = new String("stringa_di_esempio");
String str3 = "abc"; // Stringa costante. Non può essere più cambiata
System.out.println(str2);
System.out.println(str3);
```

I metodi:

```
String str = str2.substring(1, 2);
System.out.println(str);

int i = str.length();

str.replace(char oldChar, char newChar); // Sostituisce il carattere oldChar con NewChar

int i = str.hashCode();

str.concat(String anotherString);

str.compareTo(String anotherString);

....
```



Esercizio

- ▶ Scrivere un programma che data una stringa s inserita a linea di comando, controlla se il primo carattere di s è ripetuto nella stringa, stampando la posizione in cui è ripetuto (oppure -1).



Esercizio

- ▶ Scrivere un programma che istanzi un oggetto Rectangle e ne calcoli l'area e il perimetro. Visualizzare i risultati.



L'area di un rettangolo

```
/* AreaRettangolo calcola l'area di un rettangolo */

public class AreaRettangolo {
    public static void main(String[] args) {
        int base; // base del rettangolo
        int altezza; // altezza del rettangolo
        int area; // area del rettangolo
        base = 7;
        altezza = 9;
        // int base = 7, altezza = 9, area;
        area = base * altezza;
        System.out.println();
        System.out.print("\n");
        System.out.println("\nL'area del rettangolo di base " +
            base + " e di altezza " + altezza);
        System.out.println("vale " + area + "\n\n\n");
    } // fine metodo main
} // fine classe AreaRettangolo
```



Un po' di interattività

► `import java.util.Scanner;`

```
public class AreaRettangoloInterattivo {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        System.out.println("Inserire base: ");  
        int base = sc.nextInt();  
        System.out.println("Inserire altezza: ");  
        int altezza = sc.nextInt();  
        sc.close();  
        System.out.println("L'area del rettangolo è " + (base * altezza)+" mq");  
    } // fine metodo main  
}
```

