

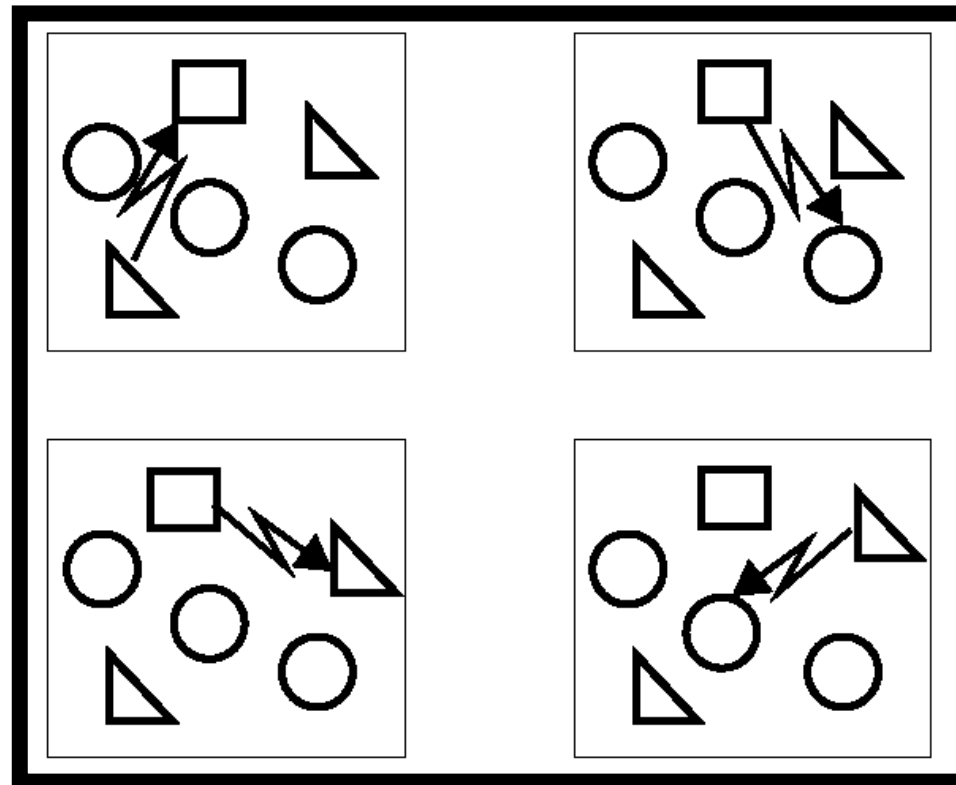


# Utilizzare Oggetti

---

# Un programma Java

- ... è un insieme di oggetti, ognuno istanza di una classe, che si inviano messaggi ...





# Percorso formativo

---

- Programmare in Java:
  - Definire classi
  - Istanziare oggetti
- Imparare ad usare oggetti e classi predefiniti
- Imparare a definire nuove classi



In questa Lezione



# Tipi e variabili

---

- Ogni valore ha un tipo
- Esempi di dichiarazioni di variabili:

```
String greeting = "Hello, World!";  
PrintStream printer = System.out;  
int luckyNumber = 13;
```

- Variabili
  - Memorizzano valori
  - Possono essere utilizzate per riferirsi ad oggetti



# Sintassi: Definizione di variabili

---

```
typeName variableName = value;
```

**oppure**

```
typeName variableName;
```

## **Esempio:**

```
String greeting = "Hello, Dave!";
```

## **Obiettivo:**

Definire una nuova variabile *variableName* di tipo *typeName* e fornire eventualmente un valore iniziale *value*



# Identificatori

---

- Nome di una variabile, un metodo o una classe
- Regole in Java:
  - Può contenere lettere, cifre e il carattere underscore (\_)
  - Non può cominciare con una cifra
  - Non può contenere altri simboli quali ad esempio ?, %, !, etc.
  - Gli spazi non sono consentiti
  - Non si possono usare parole riservate di Java
  - Maiuscolo/minuscolo sono significativi



# Convenzioni

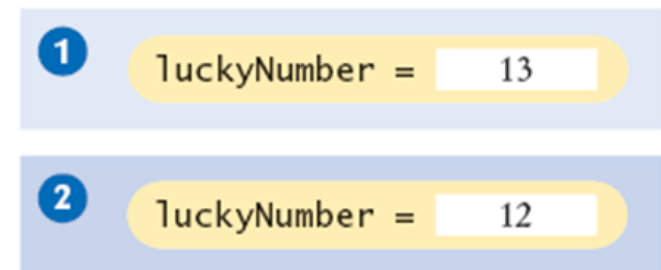
---

- Per convenzione:
  - i nomi delle variabili cominciano per lettera minuscola
  - i nomi delle classi cominciano per lettera maiuscola
  - nomi composti usano maiuscola ad ogni inizio nuova parola, es:
    - contoCorrente (variabile)
    - ContoCorrente (classe)

# Assegnamento e valori iniziali

- Operatore di assegnamento

- 1 `int luckyNumber = 13;`
- 2 `luckyNumber = 12;`



- Uso variabili non inizializzate: errore Java!

```
int luckyNumber;  
System.out.println(luckyNumber);  
// ERRORE DI COMPILAZIONE  
// variabile non inizializzata
```





# Oggetto

---

- Entità di un programma dotata di tre proprietà caratteristiche
  - stato
  - comportamento
  - identità
- Esempi:
  - casella vocale
  - conto corrente
  - stringa
  - studente
  - cliente



# Stato

---

- informazioni conservate nell'oggetto
  - Casella vocale: vuota, piena, alcuni messaggi
  - Conto corrente: saldo nullo, saldo positivo
- condiziona il comportamento dell'oggetto nel futuro
  - Casella vocale: accetta un messaggio se e solo se non piena
  - Conto corrente: consente di prelevare se e solo se saldo positivo
- può variare nel tempo per effetto di un'operazione sull'oggetto
  - Casella vocale: aggiunta/cancellazione messaggio
  - Conto corrente: versamento/prelevamento



# Comportamento

---

- definito dalle operazioni (**metodi**) che possono essere eseguite dall'oggetto
  - Casella vocale: lettura messaggio, cancellazione messaggio, etc.
  - Conto corrente: lettura saldo, versamento, prelevamento, etc.
- i metodi possono modificare lo stato dell'oggetto
  - Casella vocale: aggiunta messaggio può far cambiare lo stato da vuoto ad alcuni messaggi, o da alcuni messaggi a pieno.
  - Conto corrente: versamento può far cambiare lo stato da saldo nullo a saldo positivo



# Classe: concetto astratto

---

- Ogni oggetto appartiene a (*è un'istanza di*) una *classe* che ne determina il tipo
- Una classe descrive un insieme di oggetti caratterizzati dallo stesso insieme di
  - possibili comportamenti (metodi)
  - possibili stati (variabili di istanza o campi)
- Es. tutte le caselle vocali di un certo tipo appartengono ad una stessa classe Mailbox



## Possibili stati: le variabili di istanza

---

- Le variabili di istanza (campi) memorizzano lo stato di un oggetto
- Ciascun oggetto di una certa classe ha la propria copia delle variabili di istanza
- Le variabili di istanza **solitamente** possono essere lette e modificate solo dai metodi della stessa classe

***(incapsulamento dei dati)***



## Possibili comportamenti: metodi

---

- parte computazionale della classe
- somigliano a funzioni dei linguaggi procedurali tipo C
- possono utilizzare altri metodi (anche della stessa classe) e manipolare/accedere il contenuto delle variabili di istanza

```
String greeting = "Hello";  
greeting.println(); // Error  
greeting.length();  // OK
```



# Messaggi e metodi

---

- Il comportamento di un oggetto è attivato dalla ricezione di un messaggio
- Le classi determinano il comportamento degli oggetti definendo quali sono i messaggi “leciti”
- Le classi determinano i messaggi leciti mediante la definizione di **metodi**:
  - Una sezione di codice all'interno di una classe che implementa un particolare comportamento
  - Sono individuati da un nome del metodo

# Forma di un messaggio

---

***nome\_del\_metodo(argomenti)***

- Un messaggio deve specificare
  - Il nome del metodo da invocare  
... il comportamento desiderato
  - Gli eventuali argomenti  
... altre informazioni

○ **System.out.println** (“Benvenuti al corso”)

Nome del metodo

Argomenti





## I metodi di PrintStream

---

- Conoscere una classe equivale a conoscerne i metodi

### La classe: PrintStream

<b><u>Nome</u></b>	<b><u>Argomenti</u></b>
println	stringa di caratteri
println	nessuno
print	stringa di caratteri



## Esempi

---

- `System.out.println("Benvenuti al corso");`
- `System.out.println();`
- `System.out.print("Questa frase va su");`
- `System.out.print(" una sola linea");`



# La segnatura di un metodo

---

- `println("salve")` e `println()` sono lo stesso metodo ?
- Due metodi differenti
  - Stesso nome
  - Argomenti diversi
  - Comportamento diverso
- La **segnatura** (*signature*) di un metodo:  
*Il nome del metodo + la descrizione degli argomenti*

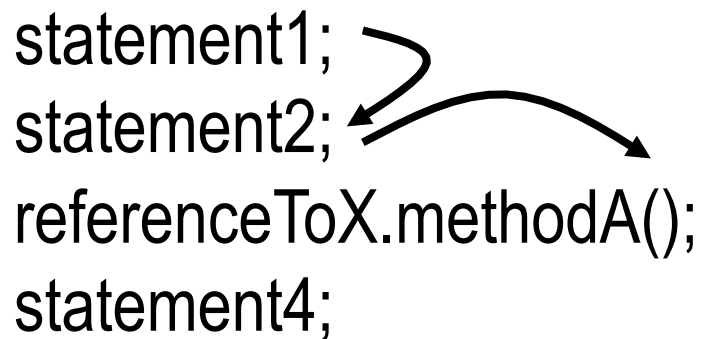


# Overloading

---

- I metodi sono individuati dalla segnatura, e non solo dal nome
- **Overloading**: la possibilità di avere una classe che definisca metodi differenti con lo stesso nome
- `println` è un metodo *overloaded* della classe `PrintStream`

# Invio di un messaggio (I)



```
statement1;  
statement2;  
referenceToX.methodA();  
statement4;
```

Codice

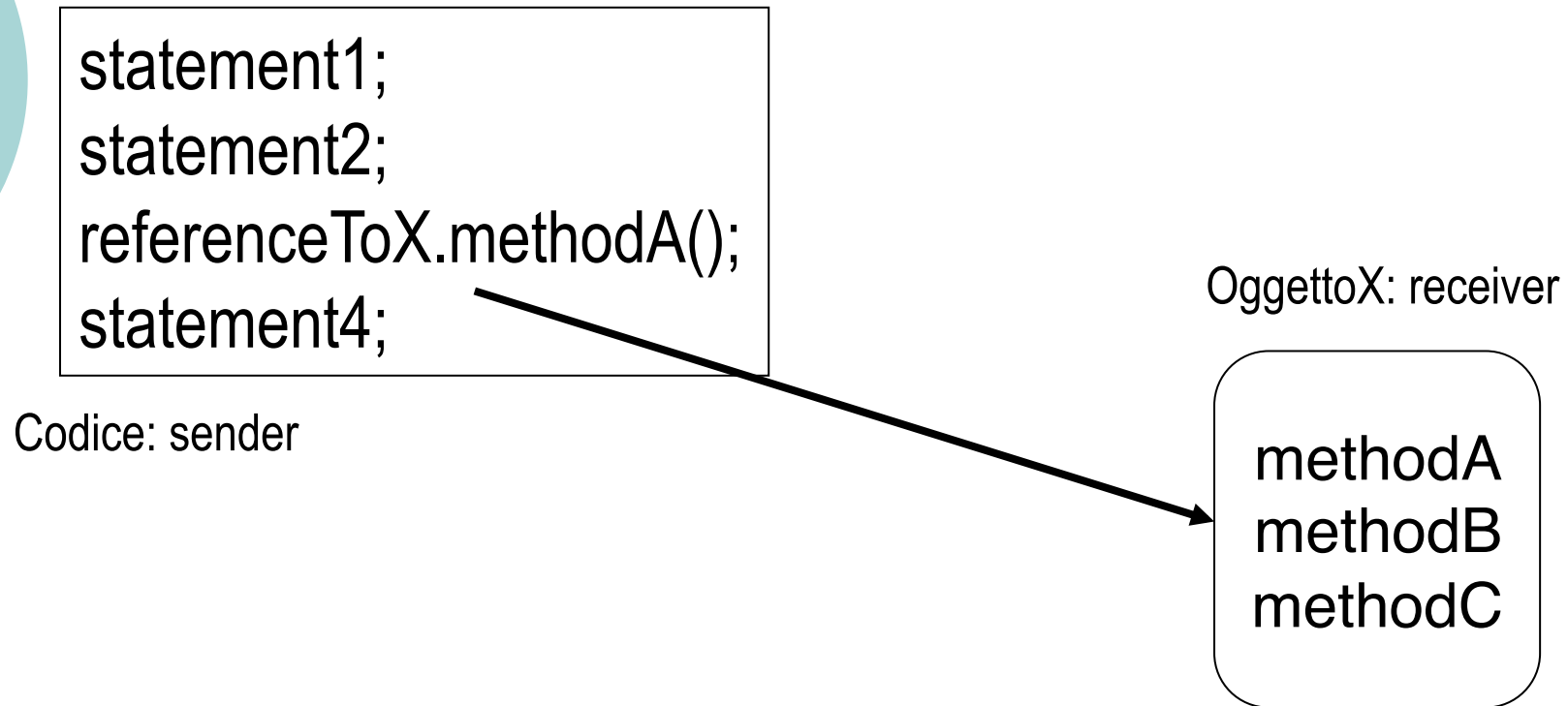
OggettoX

methodA  
methodB  
methodC

- Ordine di esecuzione sequenziale
- Fino a raggiungere una istruzione di invio di un messaggio

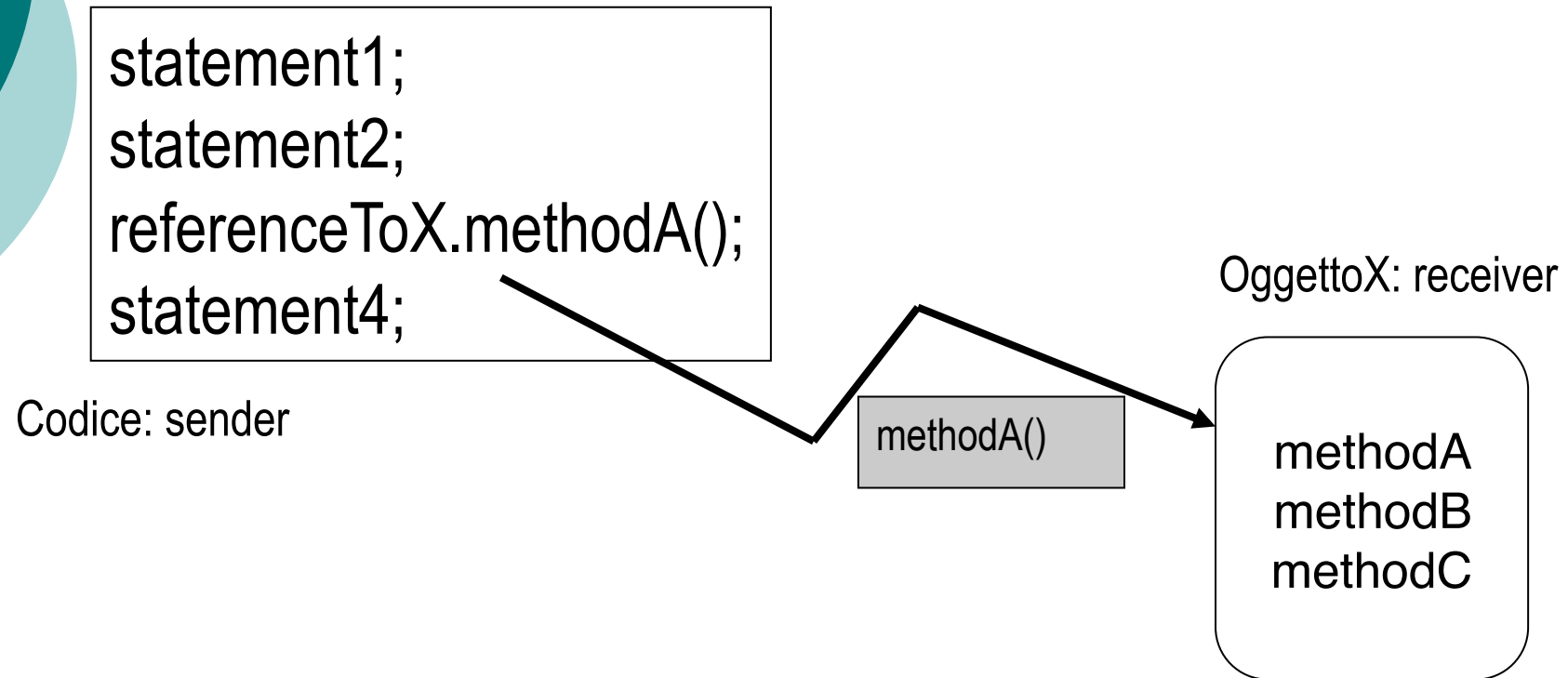
## Invio di un messaggio (II)

---



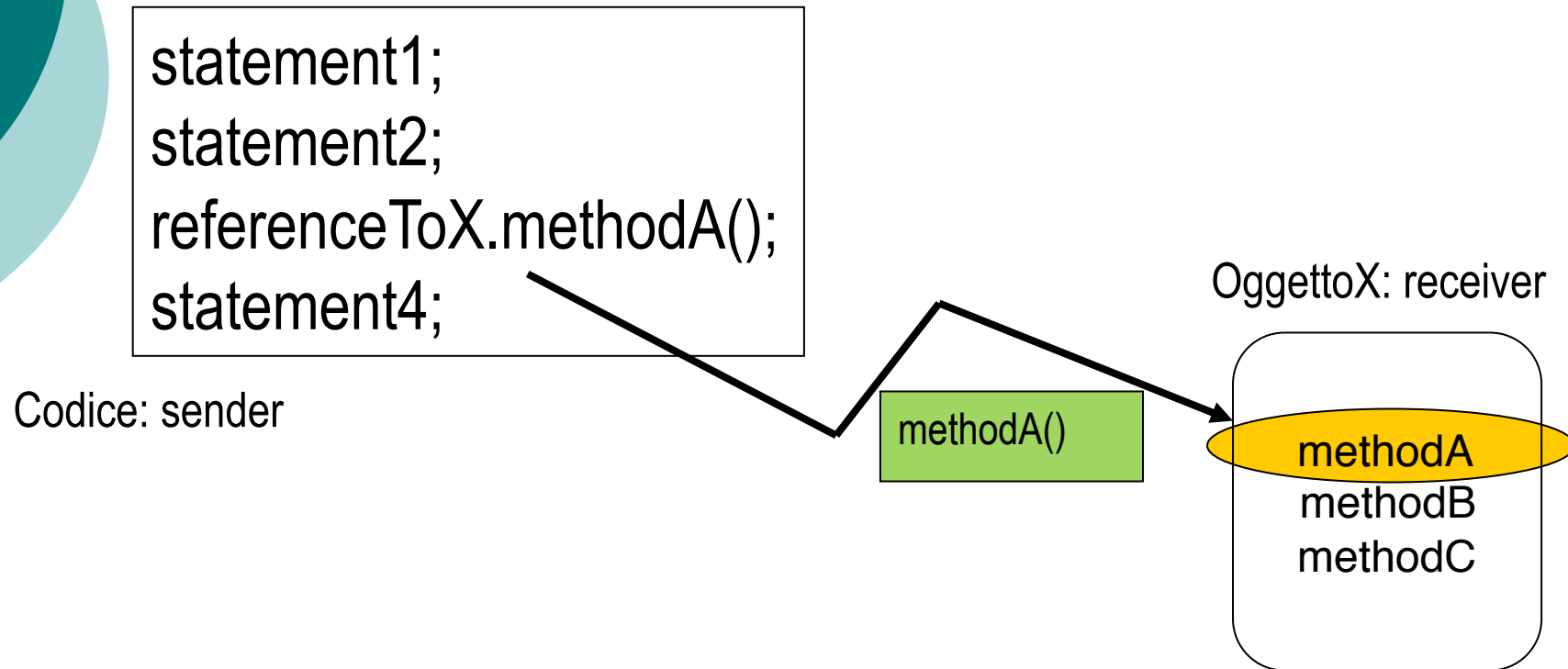
- L' esecuzione del **sender** è sospesa

## Invio di un messaggio (III)



- Il messaggio è inviato al **receiver**

## Invio di un messaggio (IV)



- L'arrivo del messaggio provoca l'invocazione di uno dei metodi del receiver

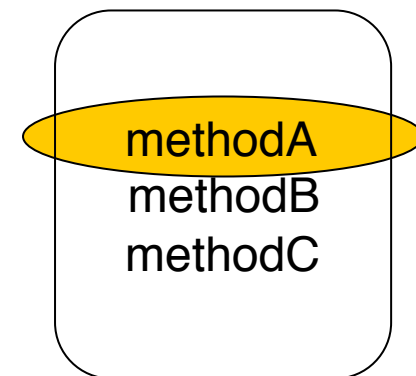


# Invio di un messaggio (V)

```
statement1;  
statement2;  
referenceToX.methodA();  
statement4;
```

Codice: sender

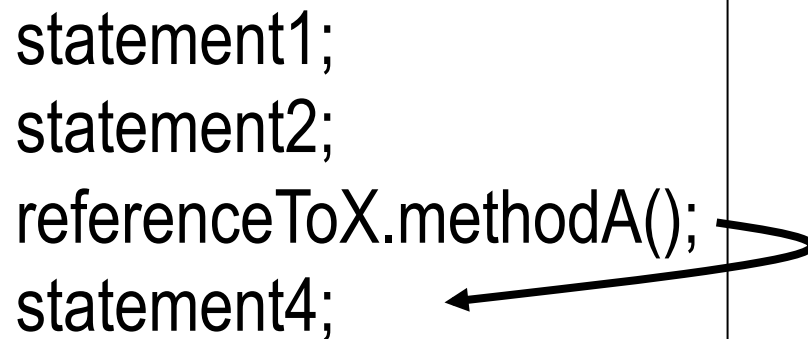
OggettoX: receiver



- Il codice relativo al metodo invocato viene eseguito
  - Questo può eventualmente provocare l'invio di altri messaggi ad altri oggetti

## Invio di un messaggio (VI)

---



```
statement1;  
statement2;  
referenceToX.methodA();  
statement4;
```

Codice: sender

OggettoX: receiver

```
methodA  
methodB  
methodC
```

- Quando l'esecuzione del metodo invocato termina
  - Il controllo (ed eventuali informazioni aggiuntive) viene restituito al sender (**return**)
  - Riprende l'ordine sequenziale



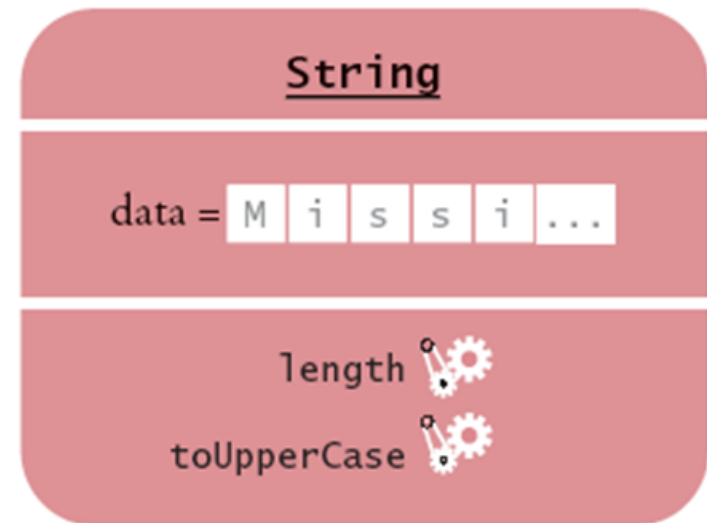
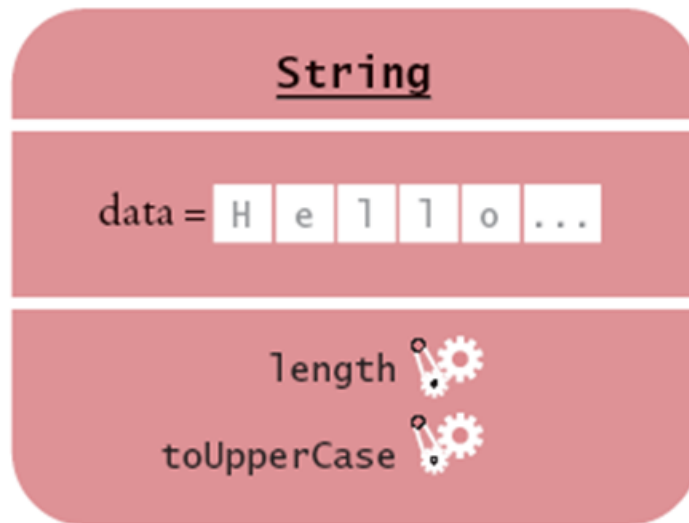
# La classe **String**

---

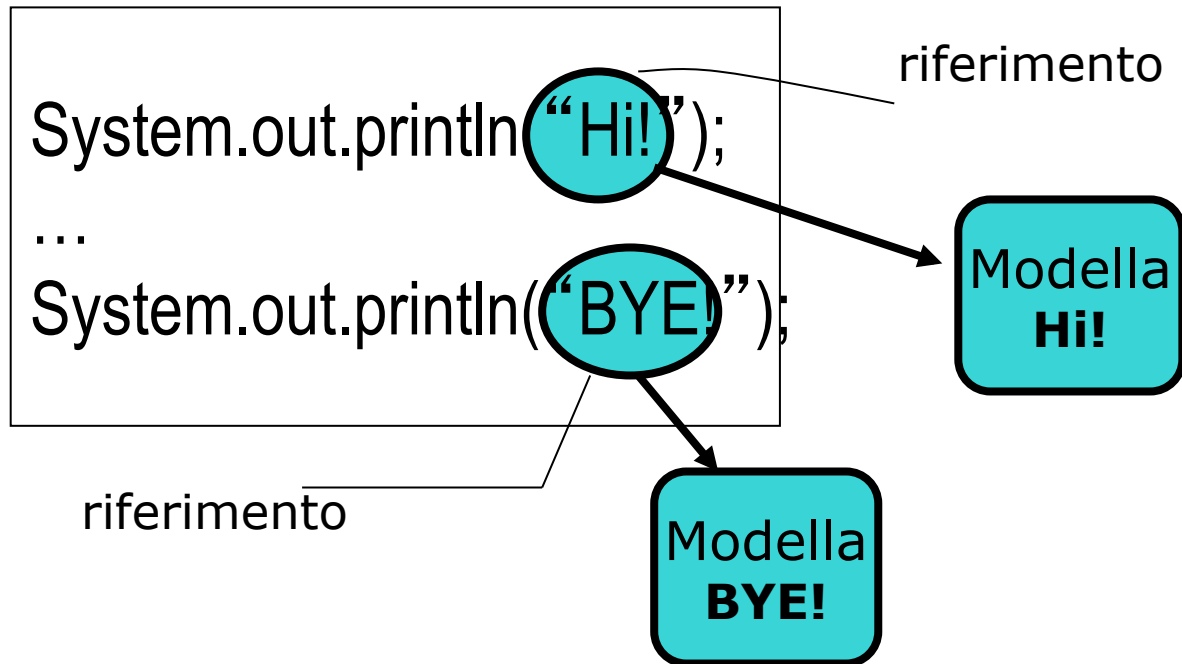
- Una classe predefinita
- Modella una qualunque sequenza di caratteri
- Referenze ad oggetti **String**
  - Sequenze di caratteri fra doppi apici
  - “Benvenuti al corso”

# Rappresentazione di due oggetti **String**

---



# String: referenze ed oggetti



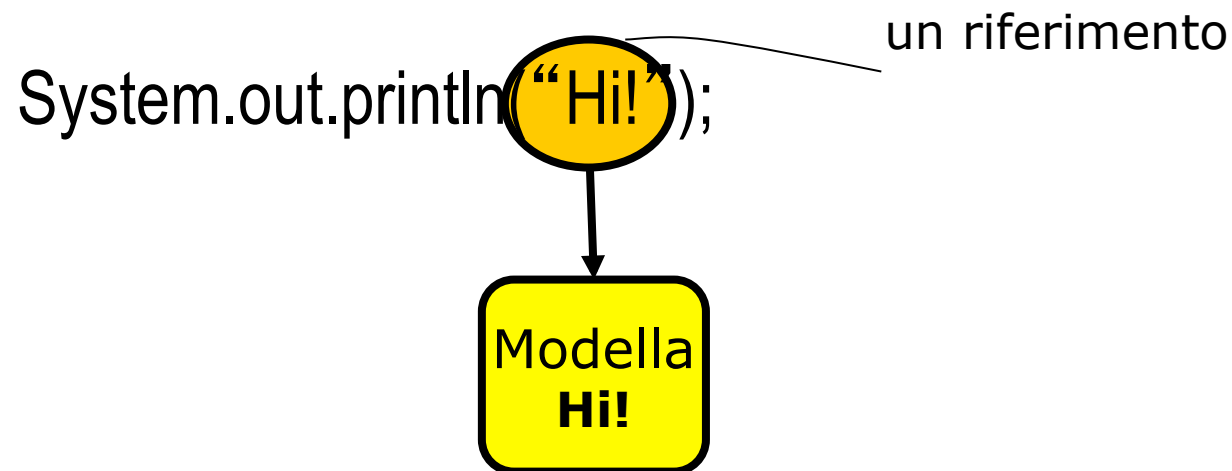
- `"Hi!"` e `"BYE!"` sono due riferimenti a oggetti **String** che modellano le sequenze di caratteri `Hi!` e `BYE!`

# Riferimenti a stringhe

## esempi di utilizzo

---

- Come argomento di un messaggio
  - Uno dei metodi **println** di **PrintStream** ha un argomento che è un riferimento ad un oggetto stringa
  - **println(riferimento-ad-un-oggetto-String)**





# Alcuni metodi di String

---

- **length()** : conta caratteri in una stringa

```
String greeting = "Hello, World!";  
int n = greeting.length(); // sets n to 13
```

- **toUpperCase()** : crea una nuova stringa che contiene i caratteri della stringa originale in maiuscolo

```
String river = "Mississippi";  
String bigRiver = river.toUpperCase();  
// sets bigRiver to "MISSISSIPPI"
```



# Invocazione di un metodo

---

- Per invocare un metodo di un certo **oggetto** bisogna specificare il nome del metodo preceduto dal nome dell'oggetto e da un punto
  - Es.: `river.length()` ;  
(Eseguiamo il metodo *length* sull'oggetto *river*)
- L'oggetto funge da *parametro implicito* nell'invocazione del metodo
  - E' come passare a *length* il parametro ***river***



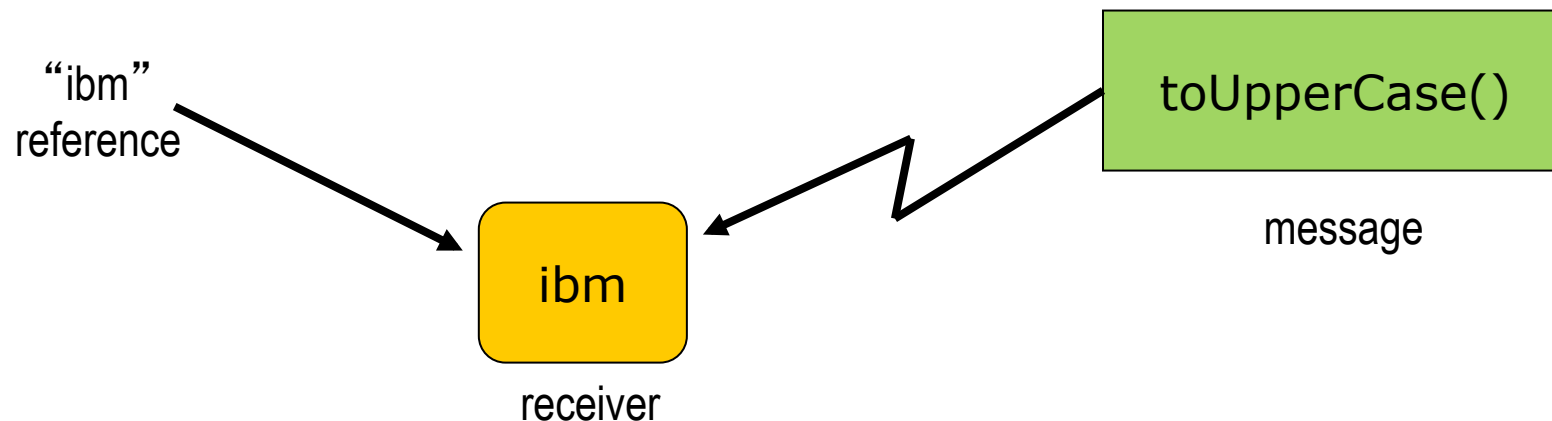
# Riferimenti a stringhe

## esempi di utilizzo

---

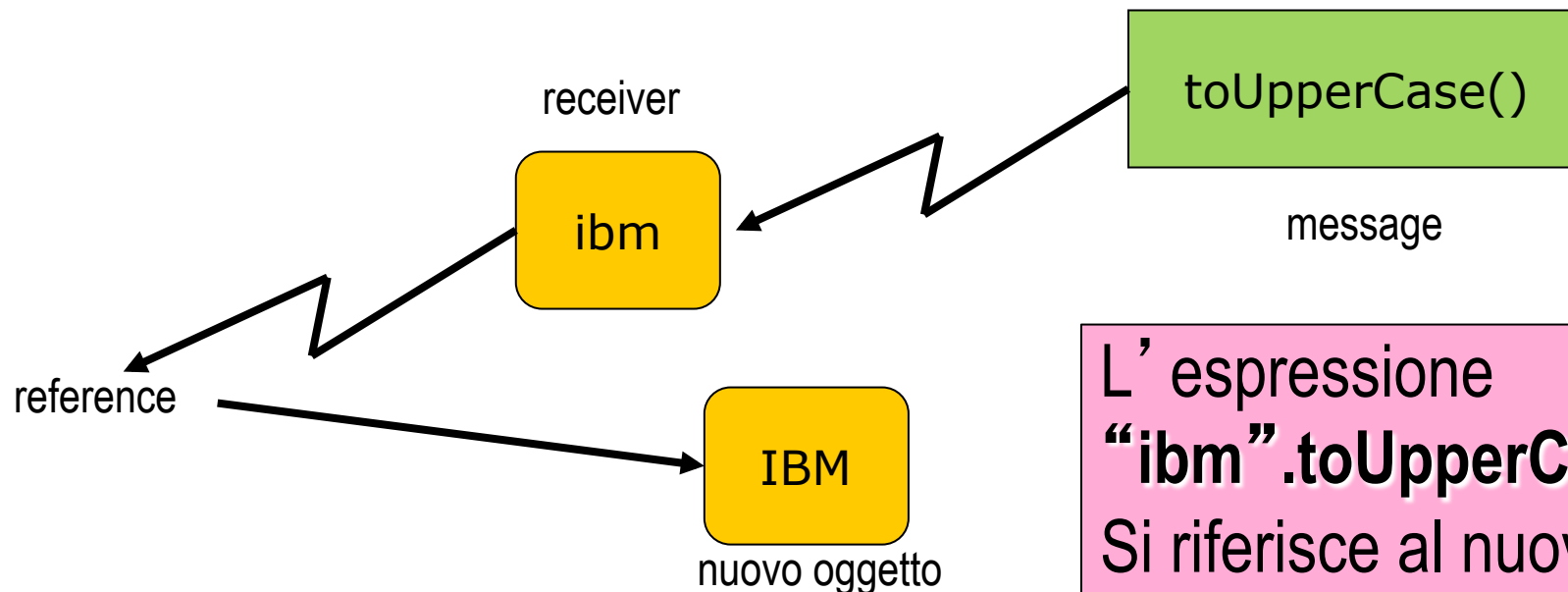
- Invio di un messaggio ad una stringa
- La classe String offre molti metodi
- Un esempio: **toUpperCase**

`"ibm".toUpperCase()`



# Il metodo toUpperCase

- Crea un nuovo oggetto `String`
- Tutti i caratteri sono in maiuscolo
- Restituisce (**returns**) un riferimento (**reference**) al nuovo oggetto



L' espressione  
"**ibm**".**toUpperCase()**  
Si riferisce al nuovo oggetto



## Self Check

---

- Come si può calcolare la lunghezza della stringa `"Mississippi"`?
- Come si può stampare la versione uppercase di `"Hello, World!"`?
- E' corretta l'invocazione `river.println()`? Perché sì o perché no?



## Risposte

---

- `river.length() or "Mississippi".length()`
- `System.out.println(greeting.toUpperCase());`
- Non è corretto. La variabile `river` è di tipo `String`.  
Il metodo `println` non è un metodo della classe `String`.



# Parametri impliciti ed espliciti

---

- Parametri (espliciti): dati in ingresso ad un metodo. Non tutti i metodi hanno parametri espliciti

```
System.out.println(greeting);  
greeting.length(); // senza parametri espliciti
```

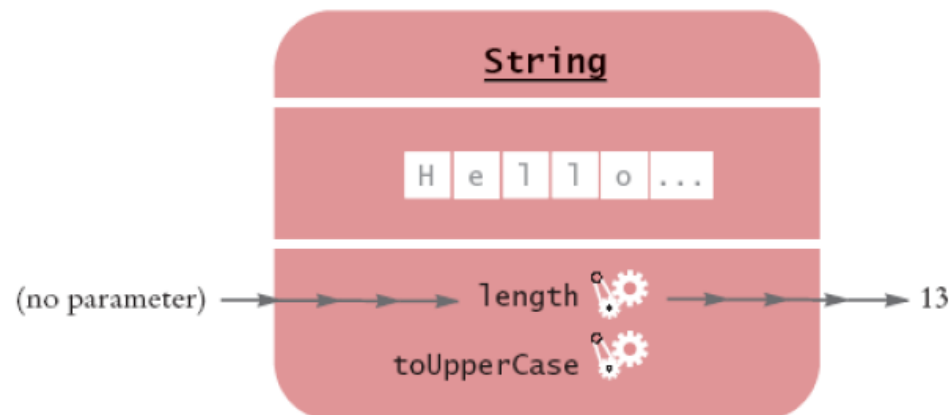
- Parametro implicito: Oggetto su cui è invocato il metodo

```
System.out.println(greeting);
```

# Valore restituito

- Un risultato che il metodo ha calcolato e che viene passato al metodo chiamante per essere utilizzato nella computazione di quest'ultimo

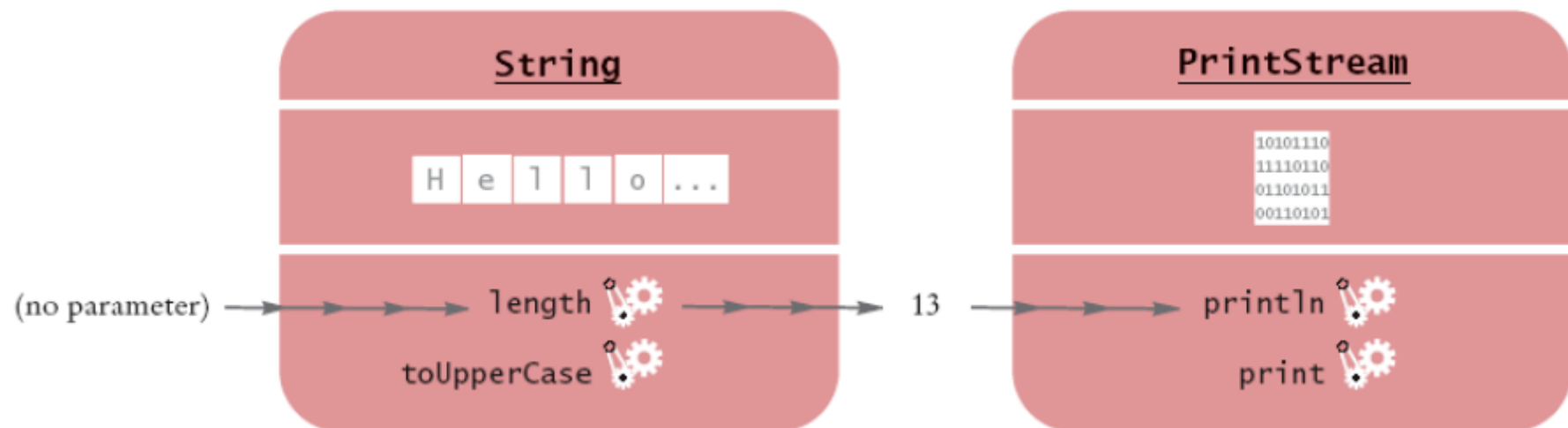
```
int n = greeting.length(); // n contiene valore restituito
```



# Valore di restituzione

- Può essere utilizzato come parametro in un messaggio

```
System.out.println(greeting.length());
```

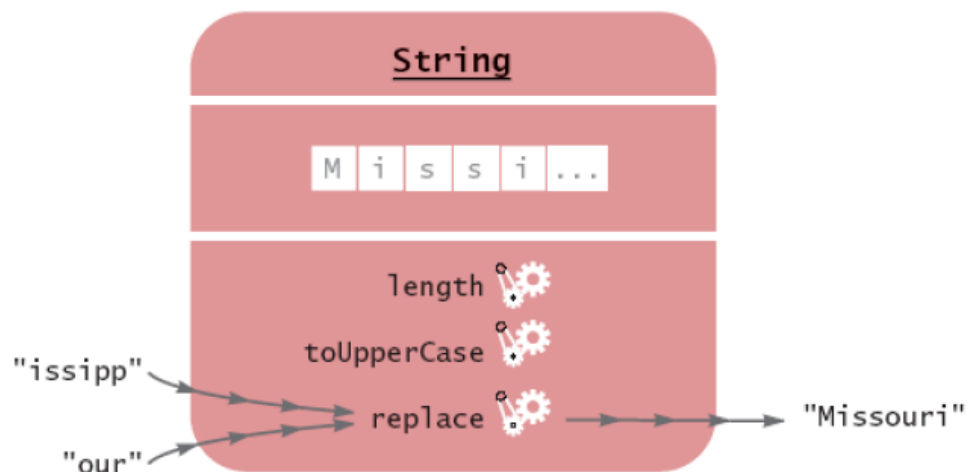


# Un esempio complesso

- Il metodo **replace** esegue una operazione di ricerca e sostituzione in una stringa

```
river.replace("issipp", "our");  
// costruisce una nuova stringa ("Missouri")
```

- Questo metodo ha:
  - 1 parametro implicito: la stringa **"Mississippi"**
  - 2 parametri espliciti: le stringhe **"issipp"** e **"our"**
  - 1 valore restituito: la stringa **"Missouri"**







## Nota sui parametri impliciti

---

- All'interno di un metodo per riferirsi esplicitamente al parametro implicito si può usare la parola chiave `this`
- Il nome di una variabile di istanza o di un metodo all'interno di un metodo di una classe si riferisce alla variabile di istanza o al metodo del parametro implicito
- A volte per chiarezza si usa `this.nomeMetodo` o `this.nomeVariabile`
  - ad es., in un metodo che esegue un'operazione con un altro oggetto della stessa classe

```
this.x = paramRect.getX();
```

(`paramRect` è un oggetto della classe `Rectangle` passato come parametro al metodo)



# Definizione di un metodo

---

- Specifica il tipo dei parametri espliciti e il valore di restituzione
- Tipo del parametro implicito = la classe corrente; non è scritto nella definizione del metodo
- Esempio nella classe **String**

```
public int length()  
// return type: int  
// no explicit parameter  
public String replace(String target, String replacement)  
// return type: String;  
// two explicit parameters of type String
```



# Definizione di un metodo

---

- **void** è usato per indicare che il metodo non restituisce alcun valore

```
public void println(String output) // in class PrintStream
```

- Il nome di un metodo è sovraccaricato (overloaded) se ci sono più metodi con lo stesso nome nella classe (con parametri differenti)

```
public void println(String output)  
public void println(int output)
```



## Self Check

---

- Quali sono i parametri impliciti, i parametri espliciti, e il valore di ritorno nella chiamata a metodo `river.length()`?

Ricorda che `String river= "Mississippi";`

- Qual'è il risultato della chiamata `river.replace("p", "s")`?

- Qual'è il risultato della chiamata `greeting.replace("World", "Dave").length()`?

Ricorda che `String greeting = "Hello, World!";`

- Com'è definito il metodo `toUpperCase` nella classe `String`?



## Risposte

---

- Il parametro implicito è `river`. Non ci sono parametri espliciti. Il valore di ritorno è `11`
- `"Missississi"`
- `12`
- Come `public String toUpperCase()`, con nessun parametro esplicito e tipo di ritorno `String`.



# Variabili di riferimento

---

- Variabile: un identificatore a cui si può attribuire un valore
  - “si supponga che x valga 5”
  - “posto y pari al valore della temperatura esterna ...”
  - Radice: variabilità nel tempo
- **Variabile di riferimento (reference variable)** = Una variabile il cui valore è un riferimento ad un oggetto



# Dichiarazione

---

- Le variabili di riferimento devono essere dichiarate

```
String greeting;
```

```
PrintStream output;
```

- In generale:

```
classe identificatore;
```



# Assegnamento

---

- E' necessario assegnare un valore ad una variabile di riferimento prima di poterla utilizzare
- Il tipo del valore deve combaciare con il tipo con cui si è dichiarata una variabile (**type matching**)

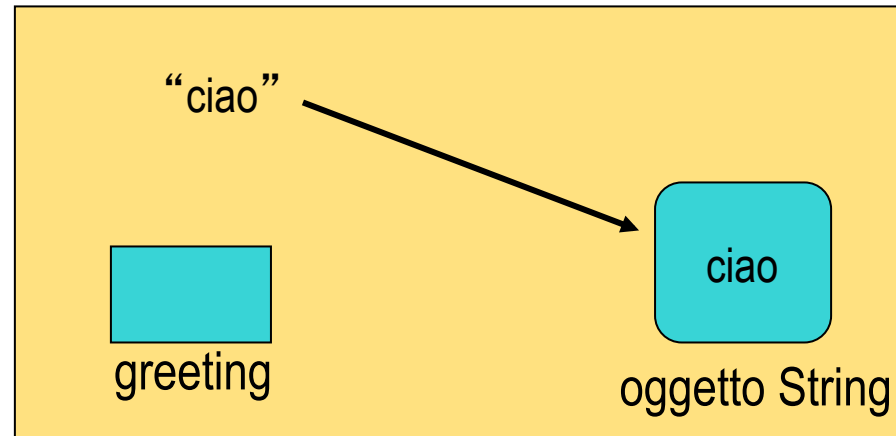
```
greeting = "Ciao";  
greeting = System.out;
```

- In generale:  
**variabile = valore;**
- Il valore è copiato nella variabile

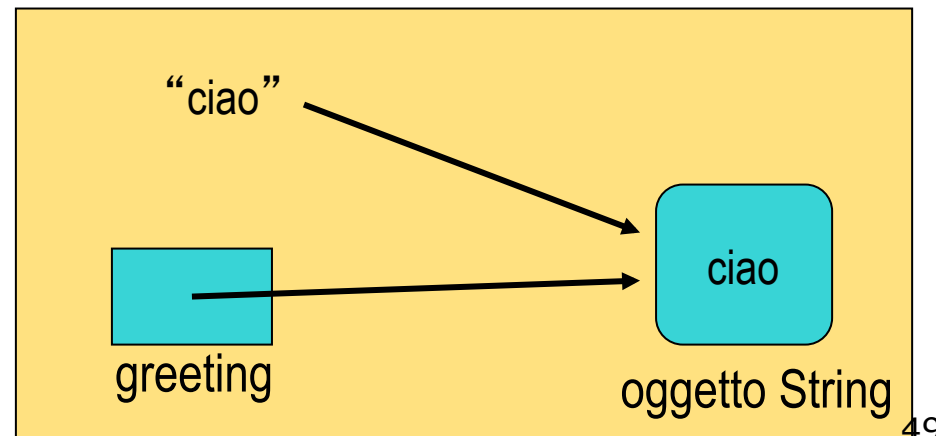


# Dichiarazione ed assegnamento

**String greeting;**  
greeting = "ciao";



String greeting;  
**greeting = "ciao";**





## Esempio (I)

---

```
String s1, s2;  
PrintStream ps1, ps2;  
s1 = "hello";  
s2 = "goodbye";  
s1 = s2;  
ps2 = System.out;  
ps1 = ps2;  
ps1.println(s1); // cosa succede ?
```



## Esempio (II)

---

```
String greeting;  
greeting = "hey!";  
String bigGreeting;  
bigGreeting = greeting.toUpperCase();  
System.out.println(bigGreeting);  
System.out.println(bigGreeting);  
System.out.println(bigGreeting);  
... al posto di ...  
System.out.println(greeting.toUpperCase());  
System.out.println(greeting.toUpperCase());  
System.out.println(greeting.toUpperCase());
```