Capitolo 2

Protocolli e contratti

Sommario: Protocolli e contratti

Paradigma di programmazione

Paradigma

...un insieme di teorie, standard e metodi che rappresentano un modo di organizzare la conoscenza, cioè un modo di guardare il mondo.

T. Kunh, La struttura delle rivoluzioni scientifiche, 1970

Paradigma di programmazione

Paradigma

...un insieme di teorie, standard e metodi che rappresentano un modo di organizzare la conoscenza, cioè un modo di guardare il mondo.

T. Kunh, La struttura delle rivoluzioni scientifiche, 1970

- Un paradigma di programmazione fornisce un metodo per:
  - concettualizzare il processo di computazione

Paradigma di programmazione

Paradigma

...un insieme di teorie, standard e metodi che rappresentano un modo di organizzare la conoscenza, cioè un modo di guardare il mondo.

T. Kunh, La struttura delle rivoluzioni scientifiche, 1970

- Un paradigma di programmazione fornisce un metodo per:
  - concettualizzare il processo di computazione
  - organizzare e strutturare i compiti che un calcolatore deve svolgere

Paradigma di programmazione

Paradigma

...un insieme di teorie, standard e metodi che rappresentano un modo di organizzare la conoscenza, cioè un modo di guardare il mondo.

T. Kunh, La struttura delle rivoluzioni scientifiche, 1970

- Un paradigma di programmazione fornisce un metodo per:
  - concettualizzare il processo di computazione
  - organizzare e strutturare i compiti che un calcolatore deve svolgere
- OOP è un paradigma contrapposto a quelli tradizionali:
  - imperativo (`Pascal`, `C`, ...)
  - funzionale (`LISP`, `FP`, ...)
  - logico (`Prolog`)

Esempio: `PinoPasticcino`

Vogliamo ordinare una torta per una festa di compleanno.

Come agiamo ?

Esempio: `PinoPasticcino`

Vogliamo ordinare una torta per una festa di compleanno.

Come agiamo ?

- Andiamo dal nostro pasticcere di fiducia (`PinoPasticcino`)

Esempio: `PinoPasticcino`

Vogliamo ordinare una torta per una festa di compleanno.

Come agiamo ?

- Andiamo dal nostro pasticcere di fiducia (`PinoPasticcino`)
- Gli comunichiamo il tipo della torta e il giorno in cui passeremo a ritirarla

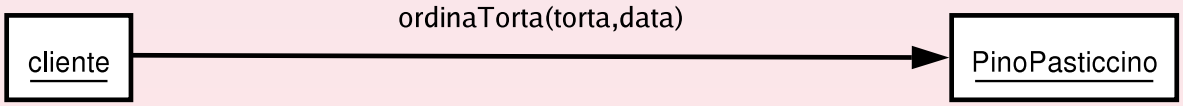
Esempio: `PinoPasticcino`

Vogliamo ordinare una torta per una festa di compleanno.

Come agiamo ?

- Andiamo dal nostro pasticcere di fiducia (`PinoPasticcino`)
- Gli comunichiamo il tipo della torta e il giorno in cui passeremo a ritirarla

Astraendo dai dettagli: inviamo un messaggio al commesso della pasticceria



Esempio: PinoPasticcino

Per risolvere il problema abbiamo:

- individuato un agente appropriato
- inviato a tale agente un messaggio contenente la nostra richiesta

Esempio: `PinoPasticcino`

Per risolvere il problema abbiamo:

- individuato un agente appropriato
- inviato a tale agente un messaggio contenente la nostra richiesta

Osserviamo che `PinoPasticcino`

- si assume la responsabilità di soddisfare la nostra richiesta

Esempio: `PinoPasticcino`

Per risolvere il problema abbiamo:

- individuato un agente appropriato
- inviato a tale agente un messaggio contenente la nostra richiesta

Osserviamo che `PinoPasticcino`

- si assume la responsabilità di soddisfare la nostra richiesta
- utilizzerà un qualche metodo (algoritmo) per soddisfarla

Oggetti

Gli oggetti sono caratterizzati da stato e comportamento.

Oggetti

Gli oggetti sono caratterizzati da stato e comportamento.

Stato

Insieme delle proprietà che caratterizzano l'oggetto in un determinato istante.

Oggetti

Gli oggetti sono caratterizzati da stato e comportamento.

Stato

Insieme delle proprietà che caratterizzano l'oggetto in un determinato istante.

- cane: nome, colore, razza, età,...
- auto: colore, potenza, livello carburante, velocità,...

Oggetti

Gli oggetti sono caratterizzati da stato e comportamento.

Stato

Insieme delle proprietà che caratterizzano l'oggetto in un determinato istante.

- cane: nome, colore, razza, età,...
- auto: colore, potenza, livello carburante, velocità,...

Comportamento

Insieme delle azioni che l'oggetto può eseguire.

Oggetti

Gli oggetti sono caratterizzati da stato e comportamento.

Stato

Insieme delle proprietà che caratterizzano l'oggetto in un determinato istante.

- cane: nome, colore, razza, età,...
- auto: colore, potenza, livello carburante, velocità,...

Comportamento

Insieme delle azioni che l'oggetto può eseguire.

- cane: abbaiare, scodinzolare, mangiare,...
- auto: accelerare, consumare, sterzare,...

Messaggi e oggetti

Nella programmazione ad oggetti un'azione viene iniziata inviando un messaggio ad un agente (un oggetto) responsabile di svolgerla.

Messaggi e oggetti

Nella programmazione ad oggetti un'azione viene iniziata inviando un messaggio ad un agente (un oggetto) responsabile di svolgerla.

- Il messaggio codifica la richiesta di un'azione ed è corredato con l'informazione necessaria (argomenti) a soddisfarla

Messaggi e oggetti

Nella programmazione ad oggetti un'azione viene iniziata inviando un messaggio ad un agente (un oggetto) responsabile di svolgerla.

- Il messaggio codifica la richiesta di un'azione ed è corredato con l'informazione necessaria (argomenti) a soddisfarla

```
ordinaTorta(torta,data)
```

Messaggi e oggetti

Nella programmazione ad oggetti un'azione viene iniziata inviando un messaggio ad un agente (un oggetto) responsabile di svolgerla.

- Il messaggio codifica la richiesta di un'azione ed è corredato con l'informazione necessaria (argomenti) a soddisfarla

```
ordinaTorta(torta,data)
```

- Il ricevente, se accetta il messaggio, si assume la responsabilità di portare a compimento la relativa azione

Messaggi e oggetti

Nella programmazione ad oggetti un'azione viene iniziata inviando un messaggio ad un agente (un oggetto) responsabile di svolgerla.

- Il messaggio codifica la richiesta di un'azione ed è corredato con l'informazione necessaria (argomenti) a soddisfarla

```
ordinaTorta(torta,data)
```

- Il ricevente, se accetta il messaggio, si assume la responsabilità di portare a compimento la relativa azione
- In risposta al messaggio il ricevente eseguirà un metodo per soddisfare la richiesta

Protocolli e contratti

Protocollo (interfaccia)

Definisce le regole di comunicazione con l'oggetto, ovvero:

- l'insieme dei messaggi
- il formato dei messaggi

che l'oggetto può riconoscere.

Protocolli e contratti

Protocollo (interfaccia)

Definisce le regole di comunicazione con l'oggetto, ovvero:

- l'insieme dei messaggi
- il formato dei messaggi

che l'oggetto può riconoscere.

Contratto

- Associato ad ogni messaggio
- Descrive il modo in cui l'oggetto garantisce di rispondere al messaggio

PinoPasticcino è un Pasticcere

E se fossi entrato in una pasticceria a caso?

PinoPasticcino è un Pasticcere

E se fossi entrato in una pasticceria a caso?

- In realtà avrei potuto risolvere il mio problema entrando in una qualunque pasticceria

PinoPasticcino è un Pasticcere

E se fossi entrato in una pasticceria a caso?

- In realtà avrei potuto risolvere il mio problema entrando in una qualunque pasticceria
- Infatti, in base alla mia esperienza, ogni `Pasticceria` è in grado di "rispondere" al messaggio

  ```
  ordinaTorta(torta,data)
  ```

  per il solo fatto di appartenere alla categoria dei pasticceri.

PinoPasticcino è un Pasticcere

E se fossi entrato in una pasticceria a caso?

- In realtà avrei potuto risolvere il mio problema entrando in una qualunque pasticceria
- Infatti, in base alla mia esperienza, ogni `Pasticceria` è in grado di "rispondere" al messaggio

  ```
  ordinaTorta(torta,data)
  ```

  per il solo fatto di appartenere alla categoria dei pasticceri.
- Una categoria di agenti che condividono il medesimo comportamento prende il nome di classe

Classi in Java

Classe

Una classe è un modello che specifica lo stato e il comportamento di tutte le sue istanze (oggetti).

Classi in Java

Classe

Una classe è un modello che specifica lo stato e il comportamento di tutte le sue istanze (oggetti).

- Tutti gli oggetti sono istanze di una classe

Classi in Java

Classe

Una classe è un modello che specifica lo stato e il comportamento di tutte le sue istanze (oggetti).

- Tutti gli oggetti sono istanze di una classe
- Il metodo utilizzato da un oggetto per rispondere a un messaggio è determinato dalla classe da cui è stato istanziato

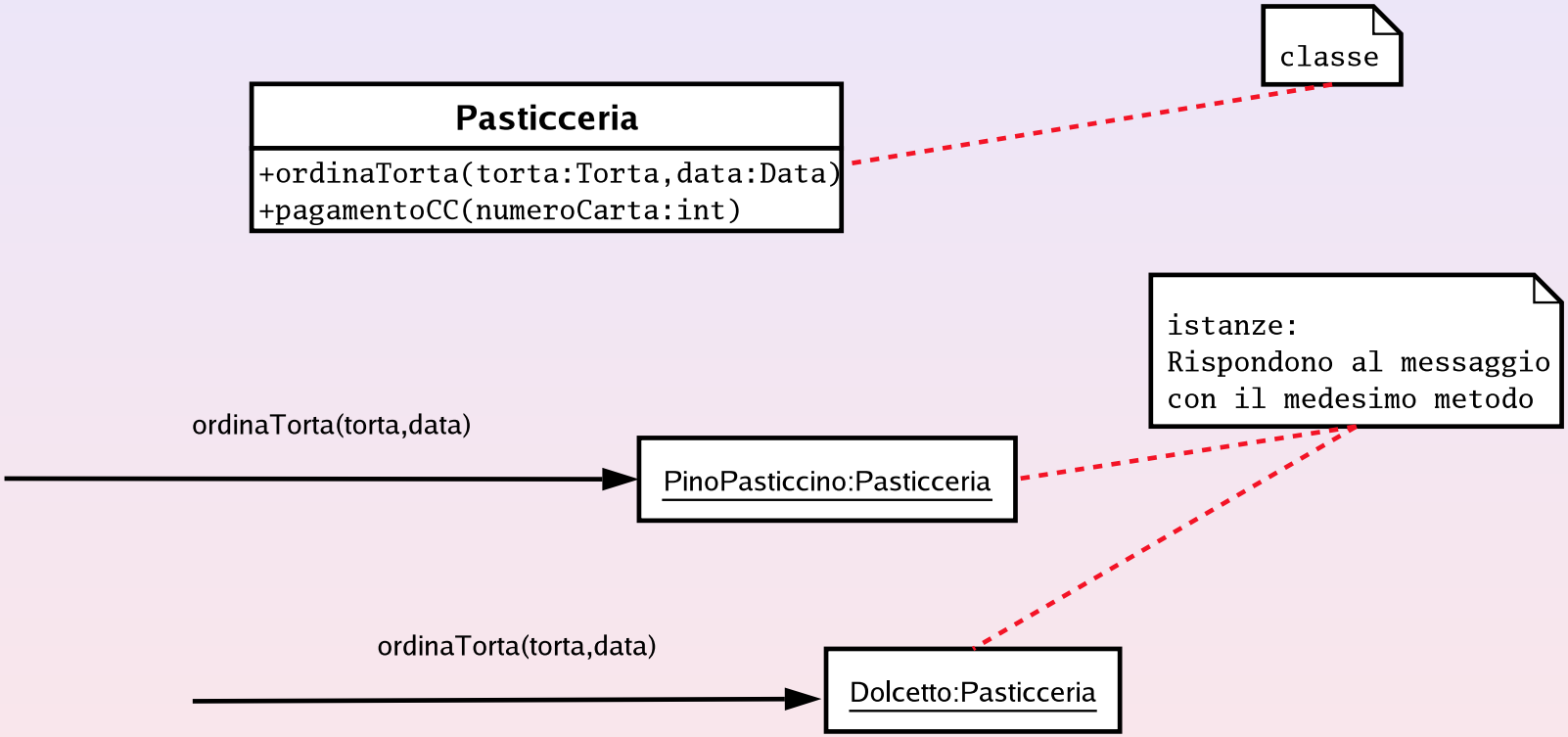
Classi in Java

Classe

Una classe è un modello che specifica lo stato e il comportamento di tutte le sue istanze (oggetti).

- Tutti gli oggetti sono istanze di una classe
- Il metodo utilizzato da un oggetto per rispondere a un messaggio è determinato dalla classe da cui è stato istanziato
- Tutti gli oggetti ottenuti istanziando una medesima classe rispondono ad un certo messaggio mediante il medesimo metodo

Esempio: la classe `Pasticceria`



Sommario: Protocolli e contratti

Programmi Java

Un programma è costituito da un insieme di oggetti che cooperano per realizzare un obiettivo

Programmi Java

Un programma è costituito da un insieme di oggetti che cooperano per realizzare un obiettivo

Per realizzare un programma dovremo:

Programmi Java

Un programma è costituito da un insieme di oggetti che cooperano per realizzare un obiettivo

Per realizzare un programma dovremo:

- conoscere le classi (fondamentali) che abbiamo a disposizione

Programmi Java

Un programma è costituito da un insieme di oggetti che cooperano per realizzare un obiettivo

Per realizzare un programma dovremo:

- conoscere le classi (fondamentali) che abbiamo a disposizione
- costruire e memorizzare gli oggetti che ci servono per realizzare un compito

Programmi Java

Un programma è costituito da un insieme di oggetti che cooperano per realizzare un obiettivo

Per realizzare un programma dovremo:

- conoscere le classi (fondamentali) che abbiamo a disposizione
- costruire e memorizzare gli oggetti che ci servono per realizzare un compito
- inviare messaggi agli oggetti

ConsoleOutputManager (prog.io)

Contratto: ConsoleOutputManager

Le sue istanze realizzano canali di comunicazione con il dispositivo di output standard, cioè con il video.

ConsoleOutputManager (prog.io)

Contratto: ConsoleOutputManager

Le sue istanze realizzano canali di comunicazione con il dispositivo di output standard, cioè con il video.

- La classe mette a disposizione metodi che consentono di visualizzare a video vari tipi di dati

ConsoleOutputManager (prog.io)

Contratto: ConsoleOutputManager

Le sue istanze realizzano canali di comunicazione con il dispositivo di output standard, cioè con il video.

- La classe mette a disposizione metodi che consentono di visualizzare a video vari tipi di dati
- Ad un'istanza di questa classe possiamo inviare il messaggio

```
println("Ecco il mio primo programma")
```

che ha l'effetto di visualizzare sul video la stringa

```
"Ecco il mio primo programma"
```

Costruzione degli oggetti

Espressione di creazione

`new` *nome_del_costruttore*(*lista_argomenti*)

Costruzione degli oggetti

Espressione di creazione

`new` *nome_del_costruttore*(*lista_argomenti*)

- `new`: operatore unario prefisso

Costruzione degli oggetti

Espressione di creazione

`new` *nome_del_costruttore*(*lista_argomenti*)

- `new`: operatore unario prefisso
- nome_del_costruttore(lista_argomenti): operando

Costruzione degli oggetti

Espressione di creazione

`new` *nome_del_costruttore*(*lista_argomenti*)

- `new`: operatore unario prefisso
- nome_del_costruttore(lista_argomenti): operando
  - nome_del_costruttore: coincide con il nome della classe

Costruzione degli oggetti

Espressione di creazione

`new` *nome_del_costruttore*(*lista_argomenti*)

- `new`: operatore unario prefisso
- nome_del_costruttore(lista_argomenti): operando
  - nome_del_costruttore: coincide con il nome della classe
  - lista_argomenti: dipende dai costruttori che la classe mette a disposizione

Costruzione degli oggetti

Espressione di creazione

`new` *nome_del_costruttore*(*lista_argomenti*)

- `new`: operatore unario prefisso
- nome_del_costruttore(lista_argomenti): operando
  - nome_del_costruttore: coincide con il nome della classe
  - lista_argomenti: dipende dai costruttori che la classe mette a disposizione
- Il risultato dell'espressione è un riferimento all'oggetto costruito

Costruzione degli oggetti

Espressione di creazione

`new` *nome_del_costruttore*(*lista_argomenti*)

- `new`: operatore unario prefisso
- nome_del_costruttore(lista_argomenti): operando
  - nome_del_costruttore: coincide con il nome della classe
  - lista_argomenti: dipende dai costruttori che la classe mette a disposizione
- Il risultato dell'espressione è un riferimento all'oggetto costruito

```
new ConsoleOutputManager()
```

Espressioni

Le espressioni sono sequenze di operatori e di operandi costruite secondo le regole sintattiche del linguaggio.

Espressioni

Le espressioni sono sequenze di operatori e di operandi costruite secondo le regole sintattiche del linguaggio.

- Hanno un tipo complessivo determinato dagli operatori e dal tipo degli operandi

Espressioni

Le espressioni sono sequenze di operatori e di operandi costruite secondo le regole sintattiche del linguaggio.

- Hanno un tipo complessivo determinato dagli operatori e dal tipo degli operandi
- Danno luogo, in fase di esecuzione, a un valore

Espressione di creazione

Espressione di creazione

`new` *nome_del_costruttore*(*lista_argomenti*)

Tipo la classe dell'oggetto costruito

Espressione di creazione

Espressione di creazione

`new` *nome_del_costruttore*(*lista_argomenti*)

Tipo la classe dell'oggetto costruito

Valore il riferimento a un oggetto della classe specificata dal costruttore

Espressione di creazione

Espressione di creazione

`new` *nome_del_costruttore*(*lista_argomenti*)

Tipo la classe dell'oggetto costruito

Valore il riferimento a un oggetto della classe specificata dal costruttore

Esempio

```
new ConsoleOutputManager()
```

Tipo `ConsoleOutputManager`

Espressione di creazione

Espressione di creazione

`new` *nome_del_costruttore*(*lista_argomenti*)

Tipo la classe dell'oggetto costruito

Valore il riferimento a un oggetto della classe specificata dal costruttore

Esempio

```
new ConsoleOutputManager()
```

Tipo `ConsoleOutputManager`

Valore riferimento ad un oggetto di tipo `ConsoleOutputManager`

Memorizzazione del riferimento

Dobbiamo:

- dichiarare una variabile del tipo opportuno

Memorizzazione del riferimento

Dobbiamo:

- dichiarare una variabile del tipo opportuno
- assegnarle il valore restituito dall'espressione di creazione

Memorizzazione del riferimento

Dobbiamo:

- dichiarare una variabile del tipo opportuno
- assegnarle il valore restituito dall'espressione di creazione

```
ConsoleOutputManager video;
video = new ConsoleOutputManager();
```

Memorizzazione del riferimento

Dobbiamo:

- dichiarare una variabile del tipo opportuno
- assegnarle il valore restituito dall'espressione di creazione

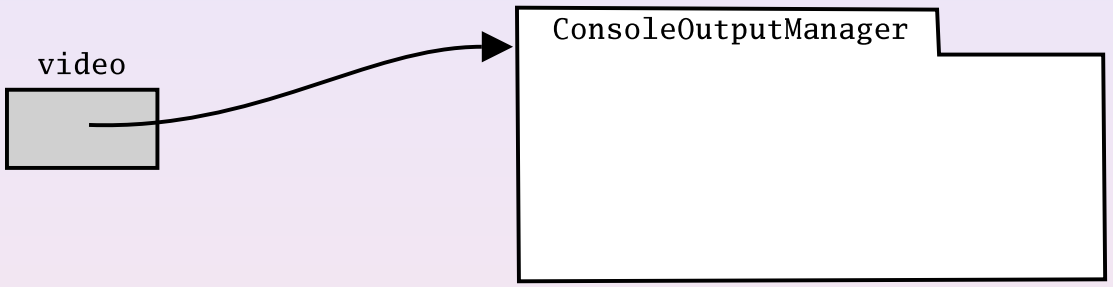
```
ConsoleOutputManager video;
video = new ConsoleOutputManager();
```

Oppure

```
ConsoleOutputManager video = new ConsoleOutputManager();
```

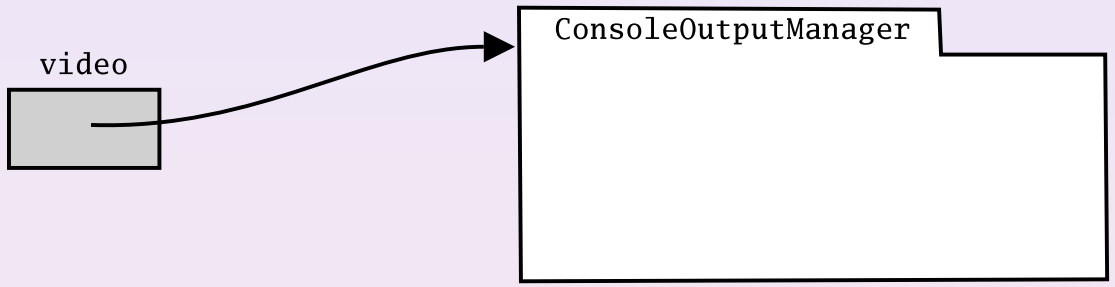
Situazione in memoria

```
ConsoleOutputManager video = new ConsoleOutputManager();
```



Situazione in memoria

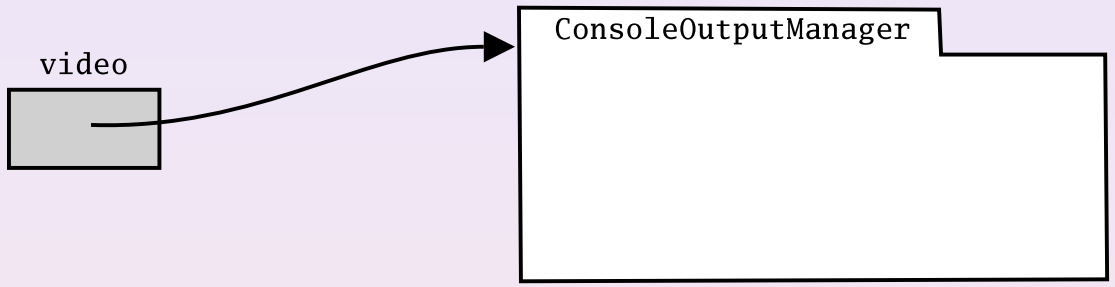
```
ConsoleOutputManager video = new ConsoleOutputManager();
```



`video` è una variabile che contiene come valore un riferimento a un oggetto

Situazione in memoria

```
ConsoleOutputManager video = new ConsoleOutputManager();
```



`video` è una variabile che contiene come valore un riferimento a un oggetto

l'oggetto contiene

- le informazioni che ne caratterizzano lo stato
- le informazioni che servono alla JVM per la gestione dell'oggetto

Invocazione di un metodo

Invocazione di metodo

riferimento_a_oggetto.*nome_metodo*(*lista_argomenti*)

Invocazione di un metodo

Invocazione di metodo

riferimento_a_oggetto.*nome_metodo*(*lista_argomenti*)

Esempio

```
ConsoleOutputManager video = new ConsoleOutputManager();
video.println("Ecco il mio primo programma");
```

L'applicazione

```
import prog.io.ConsoleOutputManager;

class PrimoProgramma {

  public static void main(String[] args) {
    ConsoleOutputManager video = new ConsoleOutputManager();

    video.println("Ecco il mio primo programma!");
  }

}
```

Compilazione ed esecuzione

Compilazione

```
> javac PrimoProgramma.java
```

Compilazione ed esecuzione

Compilazione

```
> javac PrimoProgramma.java
```

Viene generato il file `PrimoProgramma.class` contenente il bytecode

Compilazione ed esecuzione

Compilazione

```
> javac PrimoProgramma.java
```

Viene generato il file `PrimoProgramma.class` contenente il bytecode

Esecuzione

```
> java PrimoProgramma
Ecco il mio primo programma!
```

ConsoleInputManager (prog.io)

Contratto: ConsoleInputManager

Le sue istanze realizzano canali di comunicazione con il dispositivo di input standard, cioè la tastiera.

ConsoleInputManager (prog.io)

Contratto: ConsoleInputManager

Le sue istanze realizzano canali di comunicazione con il dispositivo di input standard, cioè la tastiera.

Messaggi:

- `readLine` legge una riga di testo

ConsoleInputManager (prog.io)

Contratto: ConsoleInputManager

Le sue istanze realizzano canali di comunicazione con il dispositivo di input standard, cioè la tastiera.

Messaggi:

- `readLine` legge una riga di testo
- `readInt` legge un numero intero
- ...

Esempio

```
import prog.io.ConsoleOutputManager;
import prog.io.ConsoleInputManager;

class Pappagallo {
  public static void main(String[] args) {
    //predisposizione dei canali di comunicazione
    ConsoleInputManager tastiera = new ConsoleInputManager();
    ConsoleOutputManager video = new ConsoleOutputManager();

    //lettura e comunicazione
    String messaggio = tastiera.readLine();
    video.println(messaggio);
  }
}
```

Sommario: Protocolli e contratti

Segnature e prototipi

Segnatura di un metodo

È costituita da:

- nome del metodo
- lista degli argomenti con il relativo tipo

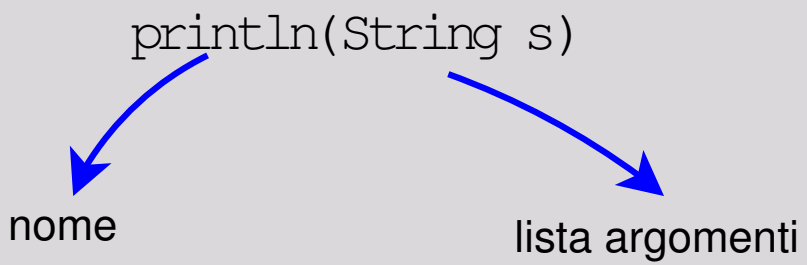
Segnature e prototipi

Segnatura di un metodo

È costituita da:

- nome del metodo
- lista degli argomenti con il relativo tipo

Esempio



Segnature e prototipi

Prototipo di un metodo

È costituito da:

- tipo del valore restituito dal metodo
- nome del metodo
- lista degli argomenti con il relativo tipo

Segnature e prototipi

Prototipo di un metodo

È costituito da:

- tipo del valore restituito dal metodo
- nome del metodo
- lista degli argomenti con il relativo tipo

$$\text{prototipo} = \text{tipo restituito} + \text{segnatura}$$

Segnature e prototipi

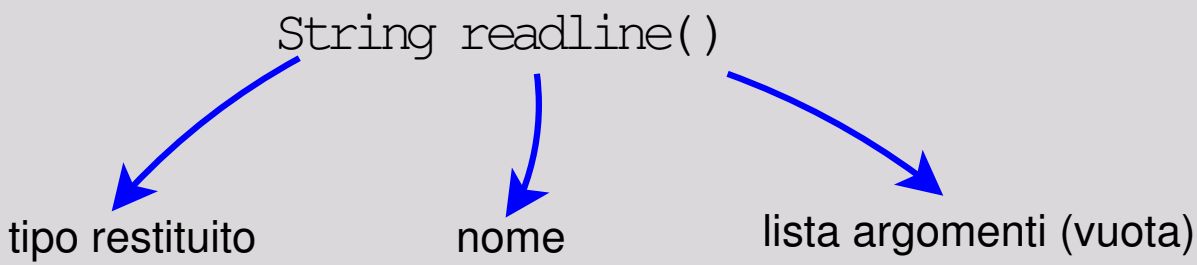
Prototipo di un metodo

È costituito da:

- tipo del valore restituito dal metodo
- nome del metodo
- lista degli argomenti con il relativo tipo

prototipo = tipo restituito + segnatura

Esempio



Alcuni metodi di `ConsoleOutputManager`

tipo restituito	*nome del metodo*	*argomenti*
`void`	`print`	`int`
`void`	`print`	riferimento a `String`
`void`	`println`	nessuno
`void`	`println`	`int`
`void`	`println`	riferimento a `String`

`void`

- È una parola chiave
- Indica che il metodo non restituisce nulla

Alcuni metodi di `ConsoleOutputManager`

tipo restituito	*nome del metodo*	*argomenti*
`void`	`print`	`int`
`void`	`print`	riferimento a `String`
`void`	`println`	nessuno
`void`	`println`	`int`
`void`	`println`	riferimento a `String`

Overloading ("sovraccaricamento")

- Possibilità di avere metodi con lo stesso nome ma segnatura diversa
- Il compilatore è in grado di capire quale metodo intendiamo invocare in base al modo in cui lo invochiamo (alla lista degli argomenti che forniamo)

Sommario: Protocolli e contratti

La classe `String`

Contratto: `String`

Le istanze della classe `String` modellano stringhe, cioè sequenze arbitrarie di caratteri

La classe `String`

Contratto: `String`

Le istanze della classe `String` modellano stringhe, cioè sequenze arbitrarie di caratteri

- Mette a disposizione (fra gli altri) un costruttore che riceve come argomento una sequenza di caratteri compresa fra doppi apici

La classe `String`

Contratto: `String`

Le istanze della classe `String` modellano stringhe, cioè sequenze arbitrarie di caratteri

- Mette a disposizione (fra gli altri) un costruttore che riceve come argomento una sequenza di caratteri compresa fra doppi apici

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

La classe `String`

Contratto: `String`

Le istanze della classe `String` modellano stringhe, cioè sequenze arbitrarie di caratteri

- Mette a disposizione (fra gli altri) un costruttore che riceve come argomento una sequenza di caratteri compresa fra doppi apici

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

- Vari metodi per la manipolazione di stringhe

Letterali di tipo `String`

Letterali

Rappresentano i valori di un tipo all'interno di un programma Java

Letterali di tipo `String`

Letterali

Rappresentano i valori di un tipo all'interno di un programma Java

- Un letterale di tipo `String` è una sequenza di caratteri compresi fra doppi apici

Letterali di tipo `String`

Letterali

Rappresentano i valori di un tipo all'interno di un programma Java

- Un letterale di tipo `String` è una sequenza di caratteri compresi fra doppi apici
- Il linguaggio fornisce un meccanismo implicito per creare oggetti di tipo `String` ricorrendo semplicemente al letterale

Letterali di tipo `String`

Letterali

Rappresentano i valori di un tipo all'interno di un programma Java

- Un letterale di tipo `String` è una sequenza di caratteri compresi fra doppi apici
- Il linguaggio fornisce un meccanismo implicito per creare oggetti di tipo `String` ricorrendo semplicemente al letterale

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

è equivalente a

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

Sequenze di escape

Alcuni caratteri che hanno un significato particolare possono essere rappresentati mediante sequenze di escape:

Sequenza di escape	Unicode	Significato
\b	\u0008	backspace BS
\t	\u0009	horizontal tab HT
\n	\u000a	linefeed LF
\f	\u000c	form feed FF
\r	\u000d	carriage return CR
\"	\u0022	double quote
\'	\u0027	single quote '
\\	\u005c	backslash

public String toUpperCase()

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`toUpperCase`	nessuno

public String toUpperCase()

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`toUpperCase`	nessuno

Restituisce come risultato il riferimento a una nuova stringa costituita dagli stessi caratteri della stringa che esegue il metodo, con l'eccezione delle lettere minuscole, che sono trasformate nelle maiuscole corrispondenti.

public String toUpperCase()

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`toUpperCase`	nessuno

Restituisce come risultato il riferimento a una nuova stringa costituita dagli stessi caratteri della stringa che esegue il metodo, con l'eccezione delle lettere minuscole, che sono trasformate nelle maiuscole corrispondenti.

Esempi

(1)
```
String s1 = "Ciao";
String s2 = s1.toUpperCase();
```

public String toUpperCase()

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`toUpperCase`	nessuno

Restituisce come risultato il riferimento a una nuova stringa costituita dagli stessi caratteri della stringa che esegue il metodo, con l'eccezione delle lettere minuscole, che sono trasformate nelle maiuscole corrispondenti.

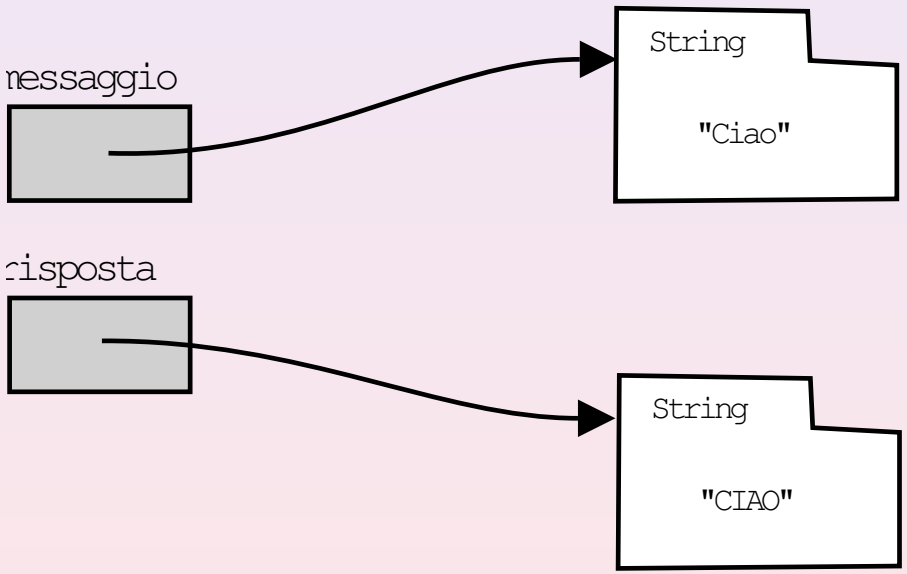
Esempi

(1)
```
String s1 = "Ciao";
String s2 = s1.toUpperCase();
```

(2)
```
String s2 = "Ciao".toUpperCase();
```

Osservazione

```
...
String messaggio = tastiera.readLine();
String risposta = messaggio.toUpperCase();
...
```



public String toLowerCase()

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`toLowerCase`	nessuno

public String toLowerCase()

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`toLowerCase`	nessuno

Contratto

Restituisce come risultato il riferimento a una nuova stringa costituita dagli stessi caratteri della stringa che esegue il metodo, con l'eccezione delle lettere maiuscole, che sono trasformate nelle minuscole corrispondenti.

public String toLowerCase()

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`toLowerCase`	nessuno

Contratto

Restituisce come risultato il riferimento a una nuova stringa costituita dagli stessi caratteri della stringa che esegue il metodo, con l'eccezione delle lettere maiuscole, che sono trasformate nelle minuscole corrispondenti.

Esempi

(1)
```
String s1 = "CIAO";
String s2 = s1.toLowerCase();
```

public String toLowerCase()

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`toLowerCase`	nessuno

Contratto

Restituisce come risultato il riferimento a una nuova stringa costituita dagli stessi caratteri della stringa che esegue il metodo, con l'eccezione delle lettere maiuscole, che sono trasformate nelle minuscole corrispondenti.

Esempi

(1)
```
String s1 = "CIAO";
String s2 = s1.toLowerCase();
```

(2)
```
String s2 = "ciao".toLowerCase();
```

public int length()

tipo restituito	*nome del metodo*	*argomenti*
`int`	`length`	nessuno

public int length()

tipo restituito	*nome del metodo*	*argomenti*
`int`	`length`	nessuno

Contratto

Restituisce un valore di tipo `int`, cioè un numero intero, uguale alla lunghezza della stringa rappresentata dall'oggetto che esegue il metodo.

public int length()

tipo restituito	*nome del metodo*	*argomenti*
`int`	`length`	nessuno

Contratto

Restituisce un valore di tipo `int`, cioè un numero intero, uguale alla lunghezza della stringa rappresentata dall'oggetto che esegue il metodo.

(1) `"Ciao".length()` restituisce 4

public int length()

tipo restituito	*nome del metodo*	*argomenti*
int	length	nessuno

Contratto

Restituisce un valore di tipo `int`, cioè un numero intero, uguale alla lunghezza della stringa rappresentata dall'oggetto che esegue il metodo.

(1) `"Ciao".length()` restituisce 4

(2) `"".length()` restituisce 0

public int length()

tipo restituito	*nome del metodo*	*argomenti*
`int`	`length`	nessuno

Contratto

Restituisce un valore di tipo `int`, cioè un numero intero, uguale alla lunghezza della stringa rappresentata dall'oggetto che esegue il metodo.

(1) `"Ciao".length()` restituisce 4

(2) `"".length()` restituisce 0

Il letterale `""` rappresenta la stringa vuota

public String concat(String str)

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`concat`	riferimento a `String`

public String concat(String str)

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`concat`	riferimento a `String`

Contratto

Restituisce un riferimento alla stringa ottenuta concatenando alla stringa che esegue il metodo la stringa fornita come argomento.

public String concat(String str)

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`concat`	riferimento a `String`

Contratto

Restituisce un riferimento alla stringa ottenuta concatenando alla stringa che esegue il metodo la stringa fornita come argomento.

(1)
```
String nome = "Pippo";
String risposta = "Buongiorno ".concat(nome).concat("!");
```

public String concat(String str)

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`concat`	riferimento a `String`

Contratto

Restituisce un riferimento alla stringa ottenuta concatenando alla stringa che esegue il metodo la stringa fornita come argomento.

(1)
```
String nome = "Pippo";
String risposta = "Buongiorno ".concat(nome).concat("!");
```

(2)
```
String nome = "Pippo";
String risposta = "Buongiorno ".concat(nome.concat("!"));
```

L'operatore + su stringhe

Fornisce un'abbreviazione per la concatenazione di stringhe:

L'operatore + su stringhe

Fornisce un'abbreviazione per la concatenazione di stringhe:

```
String nome = "Pippo";
String risposta = "Buongiorno ".concat(nome).concat("!");
```

è equivalente a

```
String risposta = "Buongiorno " + nome + "!";
```

Spezzare le righe

Non è possibile scrivere letterali di tipo `String` che occupino più di una riga

```
String mess = "Questo e' un messaggio particolarmente
               lungo che dobbiamo spezzare in piu'
               righe";
```

non viene accettata dal compilatore

Spezzare le righe

Non è possibile scrivere letterali di tipo `String` che occupino più di una riga

```
String mess = "Questo e' un messaggio particolarmente
               lungo che dobbiamo spezzare in piu'
               righe";
```

non viene accettata dal compilatore

```
String mes = "Questo e' un messaggio particolarmente " +
             "lungo che dobbiamo spezzare in piu' " +
             "righe";
```

public String substring(int begin, int end)

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`substring`	due `int`

`public String substring(int begin, int end)`

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`substring`	due `int`

Contratto

Restituisce il riferimento a una stringa formata dai caratteri che vanno dalla posizione `begin` fino alla posizione `end - 1` della stringa che esegue il metodo. Le posizioni di una stringa lunga n vanno da 0 a $n-1$.

`public String substring(int begin, int end)`

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`substring`	due `int`

Contratto

Restituisce il riferimento a una stringa formata dai caratteri che vanno dalla posizione `begin` fino alla posizione `end - 1` della stringa che esegue il metodo. Le posizioni di una stringa lunga n vanno da 0 a $n-1$.

Esempi

(1) `"distruggere"`

ha 11 caratteri, `d` è in posizione 0, la ultima `e` in posizione 10

public String substring(int begin, int end)

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`substring`	due `int`

Contratto

Restituisce il riferimento a una stringa formata dai caratteri che vanno dalla posizione `begin` fino alla posizione `end - 1` della stringa che esegue il metodo. Le posizioni di una stringa lunga n vanno da 0 a $n-1$.

Esempi

(1) `"distruggere"`

ha 11 caratteri, `d` è in posizione 0, la ultima `e` in posizione 10

(2) `"distruggere".substring(2, 9)`

fornisce come risultato un riferimento alla stringa `"strugge"`

public String substring(int begin)

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`substring`	un `int`

public String substring(int begin)

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`substring`	un `int`

Contratto

Restituisce un riferimento a una stringa formata da tutti i caratteri della stringa che esegue il metodo che si trovano tra la posizione specificata nell'argomento e la fine della stringa.

public String substring(int begin)

tipo restituito	*nome del metodo*	*argomenti*
riferimento a `String`	`substring`	un `int`

Contratto

Restituisce un riferimento a una stringa formata da tutti i caratteri della stringa che esegue il metodo che si trovano tra la posizione specificata nell'argomento e la fine della stringa.

Esempi

```
"distruggere".substring(8)
```

restituisce un riferimento alla stringa `"ere"`.

Sommario: Protocolli e contratti

Java: linguaggio fortemente tipato

Il tipo di ogni variabile e di ogni espressione che appare in un programma può essere identificato leggendo il testo del programma e, dunque, è noto al momento della compilazione.

Java: linguaggio fortemente tipato

Il tipo di ogni variabile e di ogni espressione che appare in un programma può essere identificato leggendo il testo del programma e, dunque, è noto al momento della compilazione.

- Durante la compilazione sono effettuati tutti i controlli relativi alla compatibilità dei tipi e sono stabilite eventuali conversioni implicite

Java: linguaggio fortemente tipato

Il tipo di ogni variabile e di ogni espressione che appare in un programma può essere identificato leggendo il testo del programma e, dunque, è noto al momento della compilazione.

- Durante la compilazione sono effettuati tutti i controlli relativi alla compatibilità dei tipi e sono stabilite eventuali conversioni implicite
- Le nozioni di variabile, tipo ed espressione, sono fondamentali in tutti i linguaggi di programmazione

Espressioni e tipi

Esempio

Il significato dell'espressione `x + y` dipende dai tipi delle variabili `x` e `y`.

```
int x;
int y;
```

operatore `+` denota una somma tra numeri interi

Espressioni e tipi

Esempio

Il significato dell'espressione `x + y` dipende dai tipi delle variabili `x` e `y`.

```
int x;
int y;
```

operatore `+` denota una somma tra numeri interi

risultato è di tipo `int`

Espressioni e tipi

Esempio

Il significato dell'espressione `x + y` dipende dai tipi delle variabili `x` e `y`.

```
int x;
int y;
```

operatore `+` denota una somma tra numeri interi

risultato è di tipo `int`

valutazione viene fatta sommando i valori contenuti nelle due variabili

Espressioni e tipi

Esempio

Il significato dell'espressione `x + y` dipende dai tipi delle variabili `x` e `y`.

```
String x;
String y;
```

operatore `+` denota una concatenazione tra stringhe

Espressioni e tipi

Esempio

Il significato dell'espressione `x + y` dipende dai tipi delle variabili `x` e `y`.

```
String x;
String y;
```

operatore `+` denota una concatenazione tra stringhe

risultato è un riferimento a un oggetto di tipo `String`

Espressioni e tipi

Esempio

Il significato dell'espressione `x + y` dipende dai tipi delle variabili `x` e `y`.

```
String x;
String y;
```

operatore `+` denota una concatenazione tra stringhe

risultato è un riferimento a un oggetto di tipo `String`

valutazione viene costruita la stringa costituita dalla concatenazione di quella a cui fa riferimento `x` con quella a cui fa riferimento `y`

Espressioni e tipi

Esempio

Il significato dell'espressione `x + y` dipende dai tipi delle variabili `x` e `y`.

```
String x;
int y;
```

operatore `+` denota una concatenazione tra stringhe

Espressioni e tipi

Esempio

Il significato dell'espressione `x + y` dipende dai tipi delle variabili `x` e `y`.

```
String x;
int y;
```

operatore `+` denota una concatenazione tra stringhe

risultato è un riferimento a un oggetto di tipo `String`

Espressioni e tipi

Esempio

Il significato dell'espressione `x + y` dipende dai tipi delle variabili `x` e `y`.

```
String x;
int y;
```

operatore `+` denota una concatenazione tra stringhe

risultato è un riferimento a un oggetto di tipo `String`

valutazione viene costruita la stringa costituita dalla concatenazione di quella a cui fa riferimento `x` con quella che rappresenta il valore memorizzato in `y` (conversione implicita)

Dichiarazione e definizione di variabili

Dichiarazione di variabili

Tipo var1, var2, ...;

Dichiarazione e definizione di variabili

Dichiarazione di variabili

Tipo var1, var2, ...;

Definizione di variabili

Tipo var1 = espr1, var2 = espr2, ...;

Dichiarazione e definizione di variabili

Dichiarazione di variabili

Tipo var1, var2, ...;

Definizione di variabili

Tipo var1 = espr1, var2 = espr2, ...;

Esempio

```
int x, y;
String s;

int i = 4, j = 3;
String nome = "pippo";
```

Tipi primitivi e tipi riferimento

- Tipi primitivi (come `int`)
  Una variabile di tipo primitivo può essere immaginata come una scatola in grado di contenere direttamente un valore

Tipi primitivi e tipi riferimento

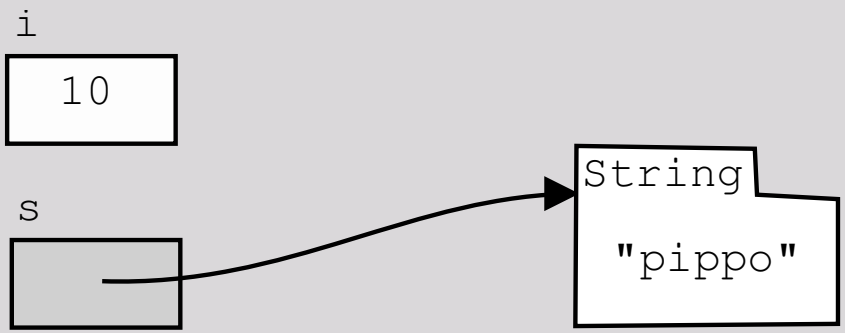
- Tipi primitivi (come `int`)
  Una variabile di tipo primitivo può essere immaginata come una scatola in grado di contenere direttamente un valore
- Tipi riferimento (come `String`)
  Una variabile di un tipo riferimento contiene il riferimento che permette di accedere all'oggetto riferito.

Tipi primitivi e tipi riferimento

- Tipi primitivi (come `int`)
  Una variabile di tipo primitivo può essere immaginata come una scatola in grado di contenere direttamente un valore
- Tipi riferimento (come `String`)
  Una variabile di un tipo riferimento contiene il riferimento che permette di accedere all'oggetto riferito.

Esempio

```
int i = 10;
String s = pippo;
```



Tipi primitivi e tipi riferimento

- Tipi primitivi

```
byte short int long
char
boolean
float double
```

Tipi primitivi e tipi riferimento

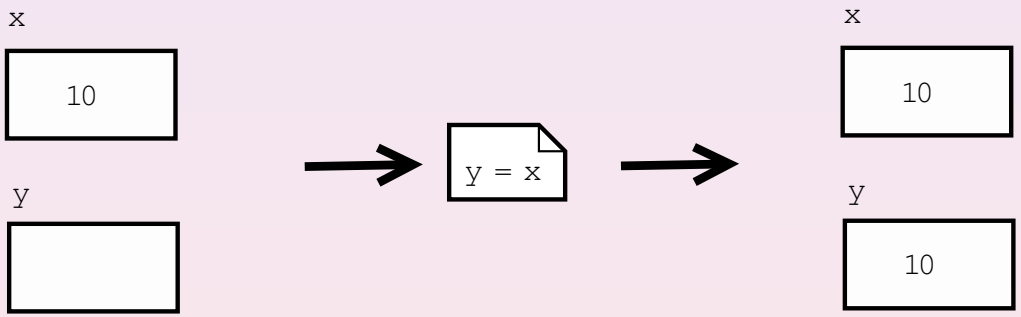
- Tipi primitivi

  ```
  byte short int long
  char
  boolean
  float double
  ```

- Tipi riferimento
  - classi
  - interfacce
  - array

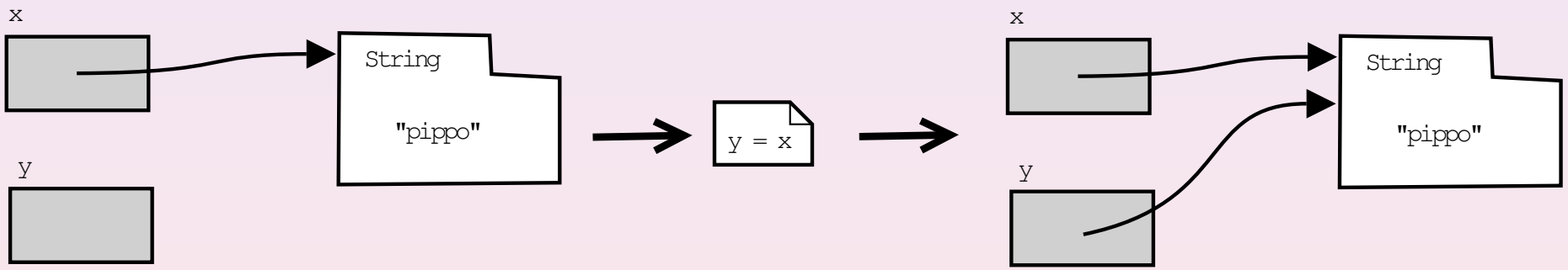
Assegnamento fra variabili di tipo primitivo

```
int x, y;
y = x;
```



Assegnamento fra variabili di tipo riferimento

```
String x, y;
y = x;
```



Il letterale `null`

- Rappresenta un valore assegnabile a tutte le variabili di tipo riferimento

Il letterale `null`

- Rappresenta un valore assegnabile a tutte le variabili di tipo riferimento
- Indica convenzionalmente che la variabile non fa riferimento ad alcun oggetto

Il letterale `null`

- Rappresenta un valore assegnabile a tutte le variabili di tipo riferimento
- Indica convenzionalmente che la variabile non fa riferimento ad alcun oggetto
- Il tentativo di accedere a un oggetto tramite un riferimento `null` provoca un errore di esecuzione (si richiede un servizio ad un oggetto inesistente)

Il letterale `null`

- Rappresenta un valore assegnabile a tutte le variabili di tipo riferimento
- Indica convenzionalmente che la variabile non fa riferimento ad alcun oggetto
- Il tentativo di accedere a un oggetto tramite un riferimento `null` provoca un errore di esecuzione (si richiede un servizio ad un oggetto inesistente)

Esempio

```
String s;
s = null;
int x = s.length();  //ERRORE IN FASE DI ESECUZIONE
```