Capitolo 2

Protocolli e contratti

Sommario:

Nozioni base della programmazione OO
- Oggetti: stato e comportamento
- Protocolli e contratti
- Classi

Primi passi
- Programmi Java
- La classe `Frazione`
- Creazione degli oggetti
- Invocazione di un metodo
- La classe `ConsoleOutputManager`
- La classe `ConsoleInputManager`
- Prototipi e segnature
- La classe `String`

Variabili e tipi
- Espressioni e tipi
- Dichiarazione e definizione di variabili
- Tipi primitivi e tipi riferimento

I metodi statici

La classe `Frazione`
- Contratto, costruttori e metodi

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 uno dei paradigmi di programmazione:
  - imperativo (`Pascal`, `C`, ...)
  - funzionale (`LISP`, `FP`, ...)
  - logico (`Prolog`)

Esempio: daPinoPasticcino

Vogliamo ordinare una torta per una festa di compleanno.

Come agiamo ?

Esempio: daPinoPasticcino

Vogliamo ordinare una torta per una festa di compleanno.

Come agiamo ?

- Andiamo dal nostro pasticcere di fiducia (daPinoPasticcino)

Esempio: daPinoPasticcino

Vogliamo ordinare una torta per una festa di compleanno.

Come agiamo ?

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

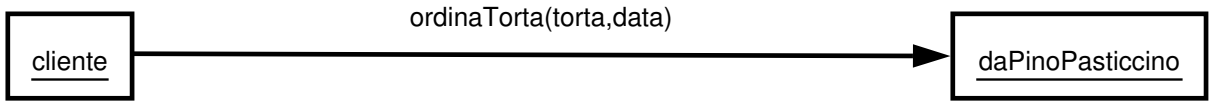
Esempio: daPinoPasticcino

Vogliamo ordinare una torta per una festa di compleanno.

Come agiamo ?

- Andiamo dal nostro pasticcere di fiducia (`daPinoPasticcino`)
- 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: daPinoPasticcino

- Per risolvere il problema abbiamo:
  - individuato un *agente* appropriato
  - *inviato a tale agente un messaggio* contenente la nostra richiesta

Esempio: `daPinoPasticcino`

- Per risolvere il problema abbiamo:
  - individuato un *agente* appropriato
  - *inviato a tale agente un messaggio* contenente la nostra richiesta
- Osserviamo che `daPinoPasticcino`
  - si assume la *responsabilità* di soddisfare la nostra richiesta

Esempio: `daPinoPasticcino`

- Per risolvere il problema abbiamo:
  - individuato un *agente* appropriato
  - *inviato a tale agente un messaggio* contenente la nostra richiesta
- Osserviamo che `daPinoPasticcino`
  - si assume la *responsabilità* di soddisfare la nostra richiesta
  - utilizzerà un qualche *metodo* (algoritmo) per soddisfarla

Oggetti: stato e comportamento

Stato

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

Oggetti: 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: 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: 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,...

Oggetti e messaggi

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

Oggetti e messaggi

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

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

Oggetti e messaggi

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

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

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

Oggetti e messaggi

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

- **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 (contratto)

Oggetti e messaggi

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

- **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 (contratto)
- 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 rispondete al messaggio

Osservazione

- Ci sono due principi impliciti nel meccanismo di soluzione di problemi basato su questo modello:

Osservazione

- Ci sono due principi impliciti nel meccanismo di soluzione di problemi basato su questo modello:
  - Se c'è un compito da svolgere *trova qualcuno che lo svolga per te*

Osservazione

- Ci sono due principi impliciti nel meccanismo di soluzione di problemi basato su questo modello:
  - Se c'è un compito da svolgere *trova qualcuno che lo svolga per te*
  - Chi si assume una responsabilità *deve onorarla*

Osservazione

- Ci sono due principi impliciti nel meccanismo di soluzione di problemi basato su questo modello:
  - Se c'è un compito da svolgere *trova qualcuno che lo svolga per te*
  - Chi si assume una responsabilità *deve onorarla*
- Dal punto di vista della programmazione questo corrisponde alla *scomposizione e al riutilizzo del codice*.

Osservazione

- Ci sono due principi impliciti nel meccanismo di soluzione di problemi basato su questo modello:
  - Se c'è un compito da svolgere *trova qualcuno che lo svolga per te*
  - Chi si assume una responsabilità *deve onorarla*
- Dal punto di vista della programmazione questo corrisponde alla *scomposizione e al riutilizzo del codice*.
- Uno degli obiettivi della OOP è lo sviluppo di *componenti riutilizzabili*

Elementi contenuti in un messaggio

- Dobbiamo stabilire a chi è destinato (cioè il *ricevente*)

Elementi contenuti in un messaggio

- Dobbiamo stabilire a chi è destinato (cioè il *ricevente*)

L'interpretazione del messaggio dipende dal ricevente, cioè il metodo (la sequenza di istruzioni) con cui un agente risponde ad un messaggio dipende dall'agente stesso.

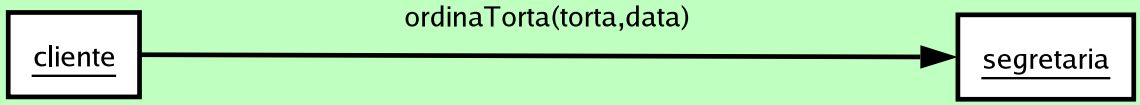
Elementi contenuti in un messaggio

- Dobbiamo stabilire a chi è destinato (cioè il *ricevente*)

L'interpretazione del messaggio dipende dal ricevente, cioè il metodo (la sequenza di istruzioni) con cui un agente risponde ad un messaggio dipende dall'agente stesso.

Alternativa

Invece di andare personalmente dal pasticcere potrei chiedere alla mia segretaria di ordinare la torta.



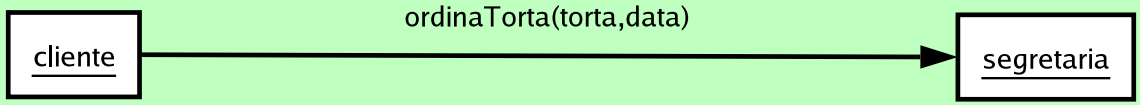
Elementi contenuti in un messaggio

- Dobbiamo stabilire a chi è destinato (cioè il *ricevente*)

L'interpretazione del messaggio dipende dal ricevente, cioè il metodo (la sequenza di istruzioni) con cui un agente risponde ad un messaggio dipende dall'agente stesso.

Alternativa

Invece di andare personalmente dal pasticcere potrei chiedere alla mia segretaria di ordinare la torta.



- Il metodo con cui risponderà al messaggio, cioè la sequenza di azioni che svolgerà, sarà diversa da quella dell'agente precedente

daPinoPasticcino è un Pasticcere

E se fossi entrato in una pasticceria a caso?

daPinoPasticcino è un Pasticcere

E se fossi entrato in una pasticceria a caso?

- Ogni `Pasticceria` è in grado di "rispondere" al messaggio

  `ordinaTorta(torta,data)`

  per il solo fatto di appartenere alla *categoria dei pasticceri*

daPinoPasticcino è un Pasticcere

E se fossi entrato in una pasticceria a caso?

- 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

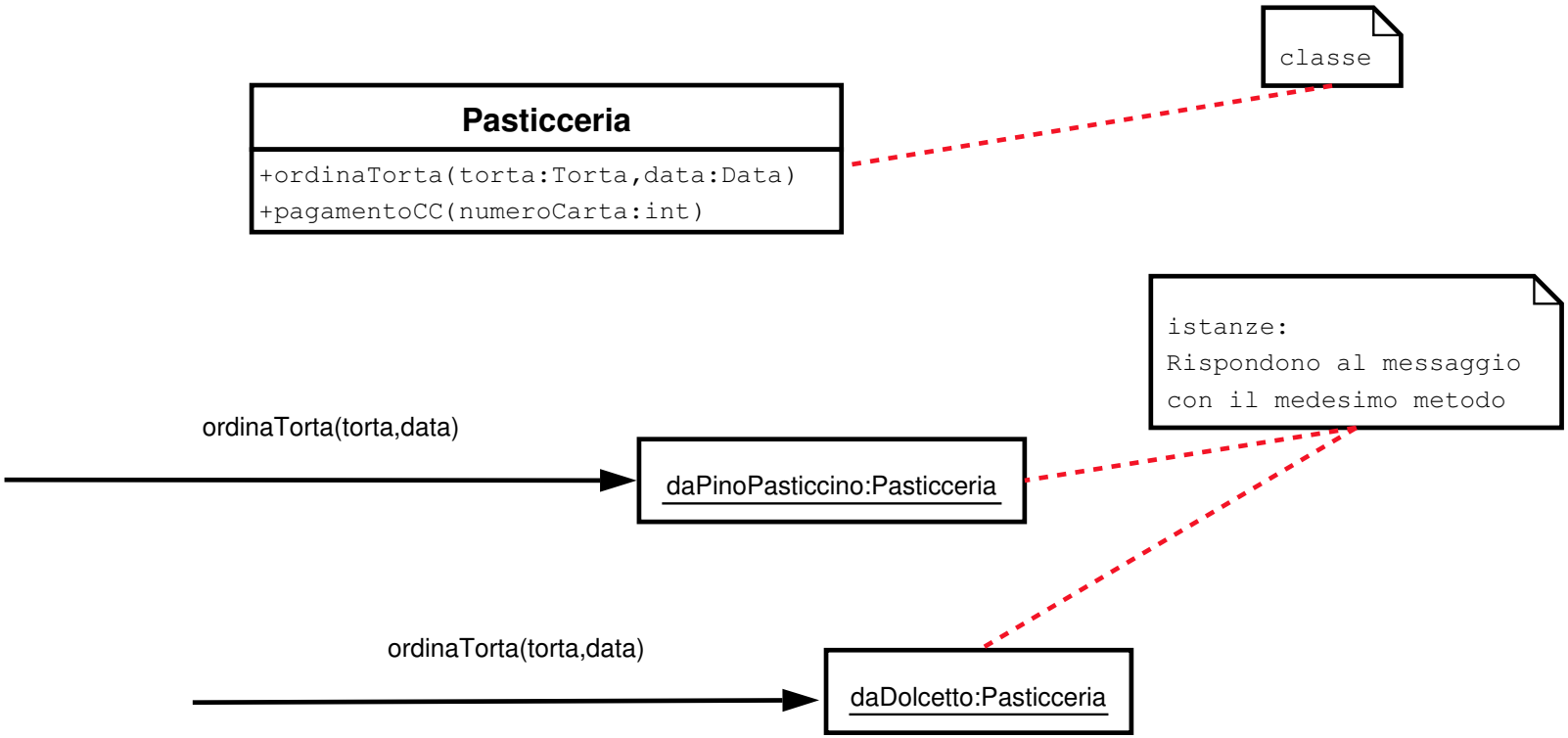
Classi in Java

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



Programmi Java

Programma

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

Programmi Java

Programma

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

- Per realizzare un programma dovremo:

Programmi Java

Programma

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

Programma

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

Programma

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

La classe `Frazione`

Contratto della classe

Le sue istanze modellano frazioni, come 2/3, 1/2, etc.

La classe `Frazione`

Contratto della classe

Le sue istanze modellano frazioni, come 2/3, 1/2, etc.

- La classe mette a disposizione:
  - metodi per compiere le usuali operazioni sulle frazioni
  - metodi per effettuare confronti fra frazioni

La classe `Frazione`

Contratto della classe

Le sue istanze modellano frazioni, come 2/3, 1/2, etc.

- La classe mette a disposizione:
  - metodi per compiere le usuali operazioni sulle frazioni
  - metodi per effettuare confronti fra frazioni
- Per poter utilizzare degli oggetti dobbiamo per prima cosa costruirli, utilizzando un servizio, offerto dalla classe, detto *costruttore*

La classe `Frazione`

Contratto della classe

Le sue istanze modellano frazioni, come 2/3, 1/2, etc.

- La classe mette a disposizione:
  - metodi per compiere le usuali operazioni sulle frazioni
  - metodi per effettuare confronti fra frazioni
- Per poter utilizzare degli oggetti dobbiamo per prima cosa costruirli, utilizzando un servizio, offerto dalla classe, detto *costruttore*
- Un costruttore di una classe è una parte di codice che si occupa di fabbricare un oggetto e di inizializzarne in modo opportuno lo stato

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 Frazione(2, 3)
```

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 Frazione(2, 3)
```

```
new Frazione(2)
```

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 Frazione(2, 3)
```

```
new Frazione(2)
```

```
new Frazione(2,1)
```

Espressioni

Espressioni

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

Espressioni

Espressioni

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

- Hanno un tipo complessivo determinato (in fase di compilazione) dagli operatori e dal tipo degli operandi

Espressioni

Espressioni

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

- Hanno un tipo complessivo determinato (in fase di compilazione) 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*)

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 Frazione(1,2)
```

- **Tipo**: `Frazione`

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 Frazione(1,2)
```

- **Tipo**: `Frazione`
- **Valore**: riferimento ad un oggetto di tipo `Frazione`

Memorizzazione del riferimento

- Per memorizzare il riferimento restituito da un'espressione di creazione dobbiamo:

Memorizzazione del riferimento

- Per memorizzare il riferimento restituito da un'espressione di creazione dobbiamo:
  - *dichiarare una variabile* del tipo opportuno

Memorizzazione del riferimento

- Per memorizzare il riferimento restituito da un'espressione di creazione dobbiamo:
  - *dichiarare una variabile* del tipo opportuno
  - *assegnarle* il valore restituito dall'espressione di creazione

Memorizzazione del riferimento

- Per memorizzare il riferimento restituito da un'espressione di creazione dobbiamo:
  - *dichiarare una variabile* del tipo opportuno
  - *assegnarle* il valore restituito dall'espressione di creazione

```
Frazione f;
f = new Frazione(2, 3);
```

Memorizzazione del riferimento

- Per memorizzare il riferimento restituito da un'espressione di creazione dobbiamo:
  - *dichiarare una variabile* del tipo opportuno
  - *assegnarle* il valore restituito dall'espressione di creazione

```
Frazione f;
f = new Frazione(2, 3);
```

oppure

Memorizzazione del riferimento

- Per memorizzare il riferimento restituito da un'espressione di creazione dobbiamo:
  - *dichiarare una variabile* del tipo opportuno
  - *assegnarle* il valore restituito dall'espressione di creazione

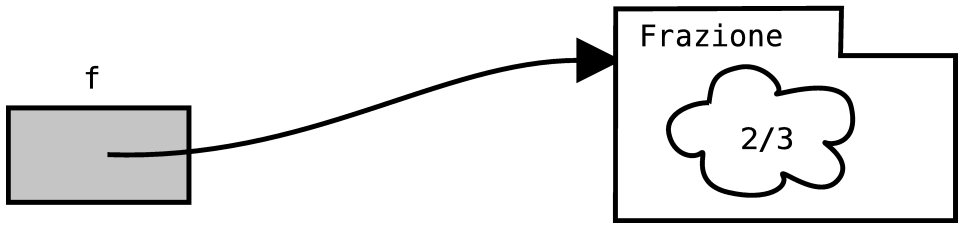
```
Frazione f;
f = new Frazione(2, 3);
```

oppure

```
Frazione f = new Frazione(2, 3);
```

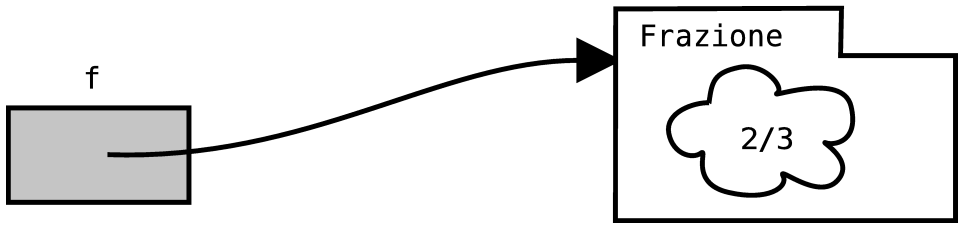
Situazione in memoria

```
Frazione f = new Frazione(2, 3);
```



Situazione in memoria

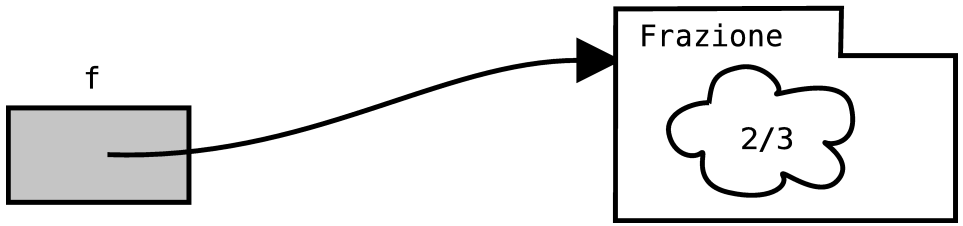
```
Frazione f = new Frazione(2, 3);
```



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

Situazione in memoria

```
Frazione f = new Frazione(2, 3);
```



- `f` è una variabile che contiene come valore un riferimento a un oggetto
- l'oggetto contiene
  - le informazioni che ne caratterizzano lo stato
  - 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

```
Frazione f = new Frazione(2, 3);
Frazione g = new Frazione(1, 2);
```

Invocazione di un metodo

Invocazione di metodo

riferimento_a_oggetto.*nome_metodo*(*lista_argomenti*)

Esempio

```
Frazione f = new Frazione(2, 3);
Frazione g = new Frazione(1, 2);
```

```
f.piu(g)
```

- In seguito all'invocazione di metodo viene costruito un nuovo oggetto di tipo `Frazione` che rappresenta il valore della frazione riferita da `f` più quello della frazione riferita da `g`.

Invocazione di un metodo

Invocazione di metodo

riferimento_a_oggetto.*nome_metodo*(*lista_argomenti*)

Esempio

```
Frazione f = new Frazione(2, 3);
Frazione g = new Frazione(1, 2);
```

```
f.piu(g)
```

- In seguito all'invocazione di metodo viene costruito un nuovo oggetto di tipo `Frazione` che rappresenta il valore della frazione riferita da `f` più quello della frazione riferita da `g`.
- Il metodo `piu` restituisce il riferimento alla nuova frazione che può essere assegnato a una variabile di tipo riferimento a `Frazione`

Situazione in memoria (1)

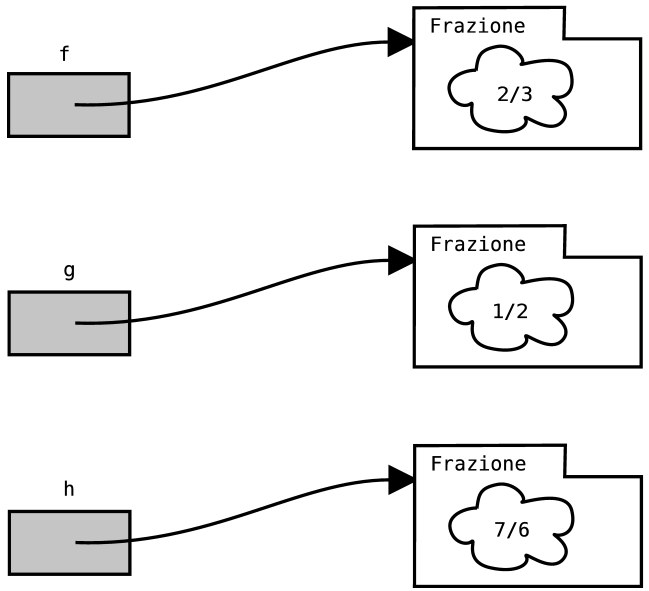
```
Frazione f = new Frazione(2, 3);
Frazione g = new Frazione(1, 2);

Frazione h = f.piu(g);
```

Situazione in memoria (1)

```
Frazione f = new Frazione(2, 3);
Frazione g = new Frazione(1, 2);

Frazione h = f.piu(g);
```



Situazione in memoria (2)

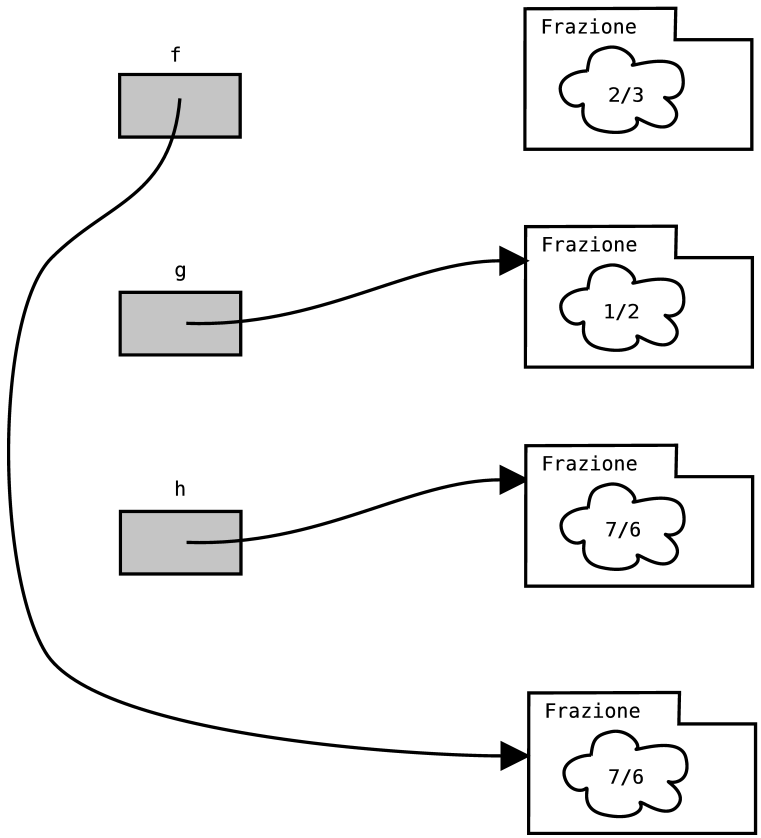
```
Frazione f = new Frazione(2, 3);
Frazione g = new Frazione(1, 2);
Frazione h = f.piu(g);

f = f.piu(g);
```

Situazione in memoria (2)

```
Frazione f = new Frazione(2, 3);
Frazione g = new Frazione(1, 2);
Frazione h = f.piu(g);

f = f.piu(g);
```



Situazione in memoria (3)

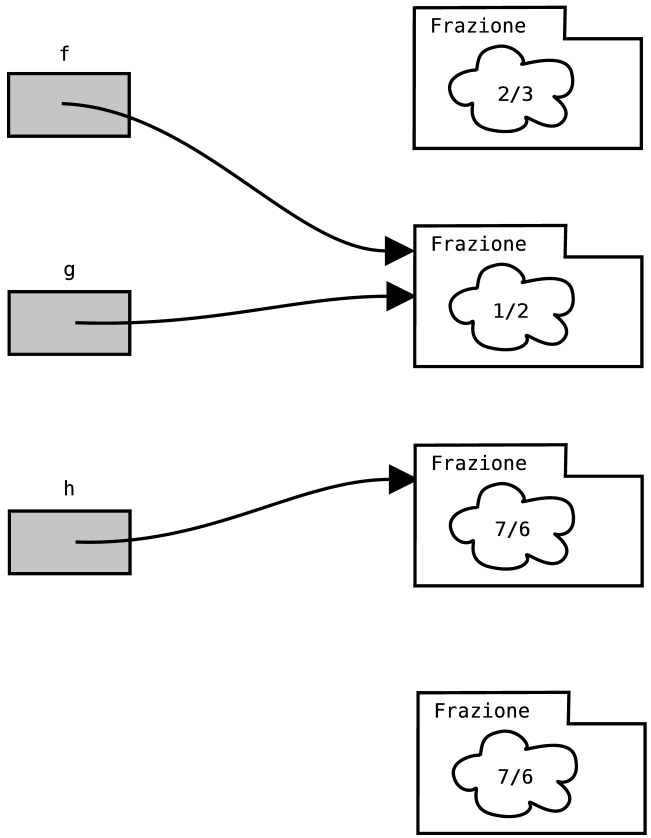
```
Frazione f = new Frazione(2, 3);
Frazione g = new Frazione(1, 2);
Frazione h = f.piu(g);
f = f.piu(g);

f = g;
```

Situazione in memoria (3)

```
Frazione f = new Frazione(2, 3);
Frazione g = new Frazione(1, 2);
Frazione h = f.piu(g);
f = f.piu(g);

f = g;
```



La classe ConsoleOutputManager (prog.io)

Contratto della classe

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

La classe ConsoleOutputManager (prog.io)

Contratto della classe

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

- La classe mette a disposizione:
  - un costruttore privo di argomenti
  - metodi che consentono di visualizzare sul monitor vari tipi di dati

La classe ConsoleOutputManager (prog.io)

Contratto della classe

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

- La classe mette a disposizione:
  - un costruttore privo di argomenti
  - metodi che consentono di visualizzare sul monitor vari tipi di dati

La classe ConsoleOutputManager (prog.io)

Contratto della classe

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

- La classe mette a disposizione:
  - un costruttore privo di argomenti
  - metodi che consentono di visualizzare sul monitor vari tipi di dati

Esempio

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

L'applicazione PrimoProgramma

PrimoProgramma.java

```
import prog.io.ConsoleOutputManager;

class PrimoProgramma {

  public static void main(String[] args) {
    ConsoleOutputManager video = new ConsoleOutputManager();
    video.println("Ecco il mio primo programma!");
  }

}
```

L'applicazione PrimoProgramma

PrimoProgramma.java

```
import prog.io.ConsoleOutputManager;

class PrimoProgramma {

  public static void main(String[] args) {
    ConsoleOutputManager video = new ConsoleOutputManager();
    video.println("Ecco il mio primo programma!");
  }

}
```

File contenente il sorgente

- Salviamo l'applicazione in un file di testo di nome `PrimoProgramma.java`

L'applicazione PrimoProgramma

PrimoProgramma.java

```
import prog.io.ConsoleOutputManager;

class PrimoProgramma {

  public static void main(String[] args) {
    ConsoleOutputManager video = new ConsoleOutputManager();
    video.println("Ecco il mio primo programma!");
  }

}
```

File contenente il sorgente

- Salviamo l'applicazione in un file di testo di nome `PrimoProgramma.java`
- Il nome del file deve avere obbligatoriamente l'estensione `.java`

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 della classe

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

ConsoleInputManager (prog.io)

Contratto della classe

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

- Fornisce un costruttore privo di argomenti

ConsoleInputManager (prog.io)

Contratto della classe

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

- Fornisce un costruttore privo di argomenti
- Fornisce metodi per effettuare la lettura di vari tipi di dati:
  - `readLine` legge una riga di testo

ConsoleInputManager (prog.io)

Contratto della classe

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

- Fornisce un costruttore privo di argomenti
- Fornisce metodi per effettuare la lettura di vari tipi di dati:
  - `readLine` legge una riga di testo
  - `readInt` legge un numero intero
  - `...`

Esempio

Pappagallo.java

```
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);
  }
}
```

Esempio

Pappagallo.java

```
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);
  }
}
```

Esempio: SommaFrazioni.java

```
//predisposizione dei canali di comunicazione
ConsoleInputManager in = new ConsoleInputManager();
ConsoleOutputManager out = new ConsoleOutputManager();
```

Esempio: SommaFrazioni.java

```
//predisposizione dei canali di comunicazione
ConsoleInputManager in = new ConsoleInputManager();
ConsoleOutputManager out = new ConsoleOutputManager();

//lettura e costruzione della prima frazione
int num = in.readInt("Numeratore prima frazione? ");
int den = in.readInt("Denominatore prima frazione? ");
Frazione f1 = new Frazione(num, den);
```

Esempio: SommaFrazioni.java

```
//predisposizione dei canali di comunicazione
ConsoleInputManager in = new ConsoleInputManager();
ConsoleOutputManager out = new ConsoleOutputManager();

//lettura e costruzione della prima frazione
int num = in.readInt("Numeratore prima frazione? ");
int den = in.readInt("Denominatore prima frazione? ");
Frazione f1 = new Frazione(num, den);

//lettura e costruzione della seconda frazione
num = in.readInt("Numeratore seconda frazione? ");
den = in.readInt("Denominatore seconda frazione? ");
Frazione f2 = new Frazione(num, den);
```

Esempio: SommaFrazioni.java

```
//predisposizione dei canali di comunicazione
ConsoleInputManager in = new ConsoleInputManager();
ConsoleOutputManager out = new ConsoleOutputManager();

//lettura e costruzione della prima frazione
int num = in.readInt("Numeratore prima frazione? ");
int den = in.readInt("Denominatore prima frazione? ");
Frazione f1 = new Frazione(num, den);

//lettura e costruzione della seconda frazione
num = in.readInt("Numeratore seconda frazione? ");
den = in.readInt("Denominatore seconda frazione? ");
Frazione f2 = new Frazione(num, den);

//calcolo della somma
Frazione somma = f1.piu(f2);
```

Esempio: SommaFrazioni.java

```
//predisposizione dei canali di comunicazione
ConsoleInputManager in = new ConsoleInputManager();
ConsoleOutputManager out = new ConsoleOutputManager();

//lettura e costruzione della prima frazione
int num = in.readInt("Numeratore prima frazione? ");
int den = in.readInt("Denominatore prima frazione? ");
Frazione f1 = new Frazione(num, den);

//lettura e costruzione della seconda frazione
num = in.readInt("Numeratore seconda frazione? ");
den = in.readInt("Denominatore seconda frazione? ");
Frazione f2 = new Frazione(num, den);

//calcolo della somma
Frazione somma = f1.piu(f2);

//comunicazione
out.print("La somma è ");
out.println(somma.toString());
```

Esempio: SommaFrazioni.java

```
//predisposizione dei canali di comunicazione
ConsoleInputManager in = new ConsoleInputManager();
ConsoleOutputManager out = new ConsoleOutputManager();

//lettura e costruzione della prima frazione
int num = in.readInt("Numeratore prima frazione? ");
int den = in.readInt("Denominatore prima frazione? ");
Frazione f1 = new Frazione(num, den);

//lettura e costruzione della seconda frazione
num = in.readInt("Numeratore seconda frazione? ");
den = in.readInt("Denominatore seconda frazione? ");
Frazione f2 = new Frazione(num, den);

//calcolo della somma
Frazione somma = f1.piu(f2);

//comunicazione
out.print("La somma è ");
out.println(somma.toString());
```

Segnature e prototipi

Segnatura di un metodo

È costituita da:

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

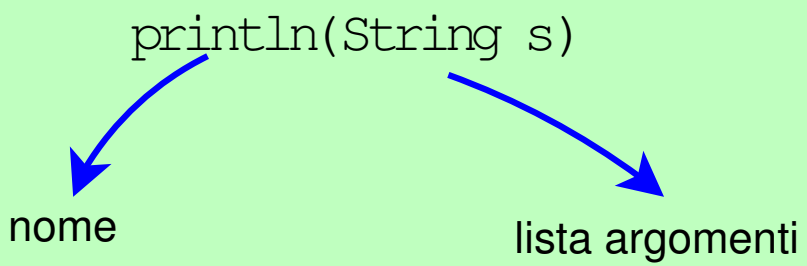
Segnature e prototipi

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

prototipo = **tipo restituito** + **segnatura**

Segnature e prototipi

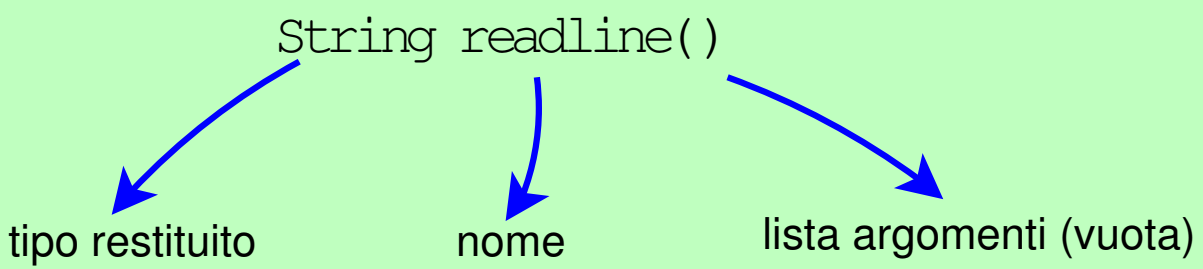
Prototipo di un metodo

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`

- `void`
  È una parola chiave, indica che il metodo non restituisce nulla
- **Overloading** ("sovraccaricamento")
  - Possibilità di avere metodi con *lo stesso nome ma segnatura diversa*

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
- **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)

La classe `String`

Contratto della classe

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

La classe `String`

Contratto della classe

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 della classe

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 della classe

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

Contratto del metodo

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

Contratto del metodo

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

Contratto del metodo

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();
```

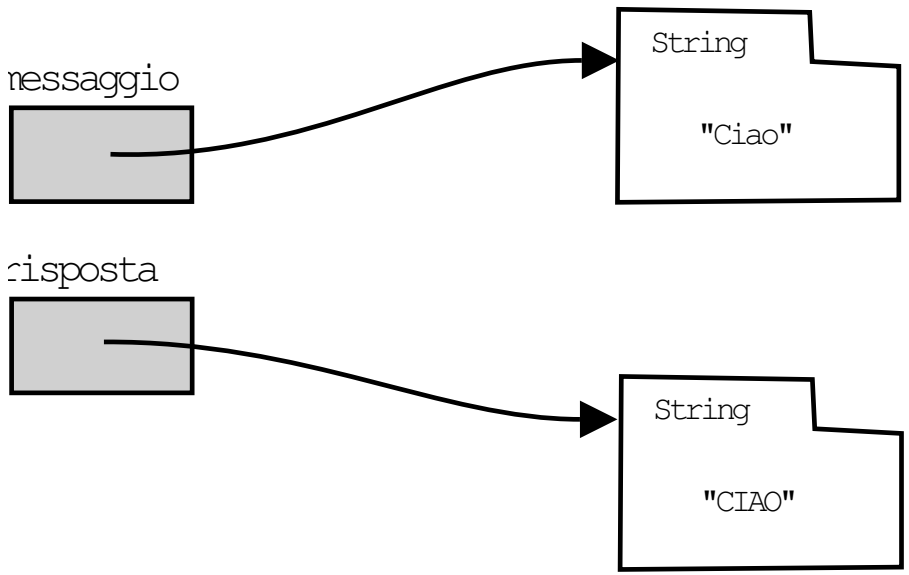
(2)
```
String s2 = "Ciao".toUpperCase();
```

Situazione in memoria

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

Situazione in memoria

```
...
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 del metodo

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 del metodo

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 del metodo

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 del metodo

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 del metodo

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

Esempi

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

public int length()

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

Contratto del metodo

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

Esempi

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

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

public int length()

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

Contratto del metodo

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

Esempi

(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 del metodo

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 del metodo

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

Esempi

(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 del metodo

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

Esempi

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

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

L'operatore + su stringhe

- L'operatore '+' fornisce un'*abbreviazione* per la concatenazione di stringhe

L'operatore + su stringhe

- L'operatore '+' fornisce un'*abbreviazione* per la concatenazione di stringhe

Esempio

```
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

Spezzare le righe

- Non è possibile scrivere letterali di tipo `String` che occupino più di una riga
- Ad esempio l'istruzione

```
String mes = "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
- Ad esempio l'istruzione

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

non viene accettata dal compilatore

- L'istruzione corretta è

```
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 del metodo

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 del metodo

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 del metodo

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 del metodo

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 del metodo

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"`.

Java: linguaggio fortemente tipizzato

Definizione

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 tipizzato

Definizione

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 tipizzato

Definizione

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`.

Espressioni e tipi

Esempio

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

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

- l'**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;
```

- l'**operatore** `+` denota una somma tra numeri interi
- il **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;
```

- l'**operatore** + denota una somma tra numeri interi
- il **risultato** è di tipo `int`
- la **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;
```

- l'**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;
```

- l'**operatore** `+` denota una concatenazione tra stringhe
- il **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;
```

- l'**operatore** `+` denota una concatenazione tra stringhe
- il **risultato** è un riferimento a un oggetto di tipo `String`
- la **valutazione** viene fatta costruendo 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;
```

- l'**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;
```

- l'**operatore** `+` denota una concatenazione tra stringhe
- il **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;
```

- l'**operatore** `+` denota una concatenazione tra stringhe
- il **risultato** è un riferimento a un oggetto di tipo `String`
- la **valutazione** viene fatta costruendo la stringa costituita dalla concatenzione 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

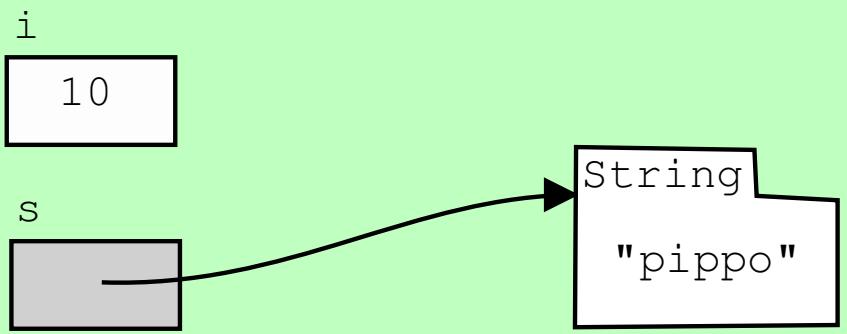
- **Tipi primitivi** (come `int`)
  Una variabile di tipo primitivo può essere immaginata come una scatola in grado di contenere *direttamente un valore*
- **Tipi riferimento** (come `String`)
  Una variabile di un tipo riferimento contiene il *riferimento* che permette di accedere all'oggetto riferito

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
float double
char
boolean
```

Tipi primitivi e tipi riferimento

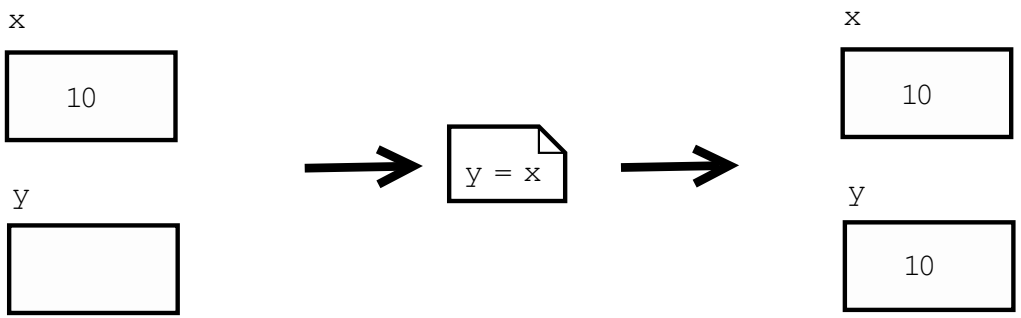
- **Tipi primitivi**

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

- **Tipi riferimento**
  - classi
  - interfacce
  - array

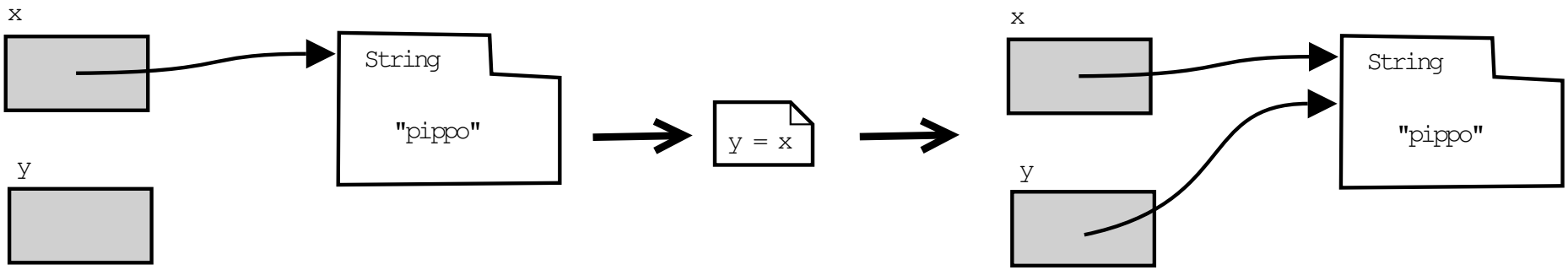
Assegnamento fra variabili di tipo primitivo

```
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
```

Metodi statici

Servizi forniti dalle classi anziché dai singoli oggetti

Metodi statici

Servizi forniti dalle classi anziché dai singoli oggetti

Usati per:

- Costruire oggetti della classe stessa a partire da oggetti o valori di altro tipo

Metodi statici

Servizi forniti dalle classi anziché dai singoli oggetti

Usati per:

- Costruire oggetti della classe stessa a partire da oggetti o valori di altro tipo

  Conversione di un `int` nella stringa che lo rappresenta

Metodi statici

Servizi forniti dalle classi anziché dai singoli oggetti

Usati per:

- Costruire oggetti della classe stessa a partire da oggetti o valori di altro tipo

  Conversione di un `int` nella stringa che lo rappresenta

- Fornire operazioni utili su oggetti o su tipi primitivi

Metodi statici

Servizi forniti dalle classi anziché dai singoli oggetti

Usati per:

- Costruire oggetti della classe stessa a partire da oggetti o valori di altro tipo

  Conversione di un `int` nella stringa che lo rappresenta

- Fornire operazioni utili su oggetti o su tipi primitivi

  La classe `Math` del package `java.lang` mette a disposizione metodi statici per realizzare alcune funzioni matematiche

Metodi statici

Servizi forniti dalle classi anziché dai singoli oggetti

Usati per:

- Costruire oggetti della classe stessa a partire da oggetti o valori di altro tipo

  Conversione di un `int` nella stringa che lo rappresenta

- Fornire operazioni utili su oggetti o su tipi primitivi

  La classe `Math` del package `java.lang` mette a disposizione metodi statici per realizzare alcune funzioni matematiche

- Definire proprietà che influenzano il comportamento di tutti gli oggetti di una classe

Metodi statici

Servizi forniti dalle classi anziché dai singoli oggetti

Usati per:

- Costruire oggetti della classe stessa a partire da oggetti o valori di altro tipo

  Conversione di un `int` nella stringa che lo rappresenta

- Fornire operazioni utili su oggetti o su tipi primitivi

  La classe `Math` del package `java.lang` mette a disposizione metodi statici per realizzare alcune funzioni matematiche

- Definire proprietà che influenzano il comportamento di tutti gli oggetti di una classe

  Stabilire il formato di rappresentazione di una data

Metodi statici

- Riconoscibili dalla presenza della parola riservata `static` nell'intestazione.

Metodi statici

- Riconoscibili dalla presenza della parola riservata `static` nell'intestazione.

Classe `String`

```
public static String valueOf(int i)
```

Restituisce il riferimento alla stringa che rappresenta il valore dell'`int` fornito come argomento.

Metodi statici

- Riconoscibili dalla presenza della parola riservata `static` nell'intestazione.

Classe `String`

```
public static String valueOf(int i)
```

Restituisce il riferimento alla stringa che rappresenta il valore dell'`int` fornito come argomento.

- Sono invocati utilizzando come destinatario del messaggio il nome della classe anziché il riferimento a un oggetto.

Invocazione di metodo statico

nome_classe.*nome_metodo*(*lista_argomenti*)

Metodi statici

java.lang.Math

- `public static double cos(double a)`
- `public static double log(double a)`
- `public static double log10(double a)`

Metodi statici

java.lang.Math

- `public static double cos(double a)`
- `public static double log(double a)`
- `public static double log10(double a)`

java.lang.Integer

- `public static int parseInt(String s)`
  Restituisce il valore di tipo `int` corrispondente alla sequenza di cifre decimali contenuta nella stringa fornita come argomento.
  Se la stringa fornita come argomento non rappresenta un numero intero si verifica un errore in fase di esecuzione.

Metodi statici

java.lang.Math

- `public static double cos(double a)`
- `public static double log(double a)`
- `public static double log10(double a)`

java.lang.Integer

- `public static int parseInt(String s)`
  Restituisce il valore di tipo `int` corrispondente alla sequenza di cifre decimali contenuta nella stringa fornita come argomento.
  Se la stringa fornita come argomento non rappresenta un numero intero si verifica un errore in fase di esecuzione.

- `public static String toBinaryString(int i)`

Esempio: SommaInColonna.java

```
//predisposizione dei canali di comunicazione
ConsoleInputManager in = new ConsoleInputManager();
ConsoleOutputManager out = new ConsoleOutputManager();
```

Esempio: SommaInColonna.java

```
//predisposizione dei canali di comunicazione
ConsoleInputManager in = new ConsoleInputManager();
ConsoleOutputManager out = new ConsoleOutputManager();

//le righe utilizzate per costruire il risultato
String rigaMeno = "----------"; //la linea di separazione contiene
                                //al più 10 caratteri

String spazi = "         "; //la massima indentazione
                            //necessaria è di 9 spazi
```

Esempio: SommaInColonna.java

```
//predisposizione dei canali di comunicazione
ConsoleInputManager in = new ConsoleInputManager();
ConsoleOutputManager out = new ConsoleOutputManager();

//le righe utilizzate per costruire il risultato
String rigaMeno = "----------"; //la linea di separazione contiene
                                //al più 10 caratteri

String spazi = "         "; //la massima indentazione
                            //necessaria è di 9 spazi

int op1, op2, somma; //variabili per operandi e risultato

//lettura dei dati
op1 = in.readInt("Inserisci il primo operando: ");
op2 = in.readInt("Inserisci il secondo operando: ");

//calcolo della somma
somma = op1 + op2;

...
```

Esempio: SommaInColonna.java (2)

```
...

//costruzione delle stringhe che rappresentano i valori
String strOp1 = String.valueOf(op1);
String strOp2 = String.valueOf(op2);
String strSomma = String.valueOf(somma);
```

Esempio: SommaInColonna.java (2)

```
...

//costruzione delle stringhe che rappresentano i valori
String strOp1 = String.valueOf(op1);
String strOp2 = String.valueOf(op2);
String strSomma = String.valueOf(somma);

//costruzione delle righe da stampare
String riga1 = spazi.substring(0, strSomma.length() - strOp1.length())
                 + strOp1 + "+";
String riga2 = spazi.substring(0, strSomma.length() - strOp2.length())
                 + strOp2 + "=";
String riga3 = rigaMeno.substring(0, strSomma.length());
```

Esempio: SommaInColonna.java (2)

```
...

//costruzione delle stringhe che rappresentano i valori
String strOp1 = String.valueOf(op1);
String strOp2 = String.valueOf(op2);
String strSomma = String.valueOf(somma);

//costruzione delle righe da stampare
String riga1 = spazi.substring(0, strSomma.length() - strOp1.length())
                + strOp1 + "+";
String riga2 = spazi.substring(0, strSomma.length() - strOp2.length())
                + strOp2 + "=";
String riga3 = rigaMeno.substring(0, strSomma.length());

//visualizzazione della somma
out.println(riga1);
out.println(riga2);
out.println(riga3);
out.println(strSomma);
```

La classe `Frazione`

`Frazione`

Le sue istanze modellano frazioni.

La classe `Frazione`

`Frazione`

Le sue istanze modellano frazioni.

Costruttori

- `public Frazione(int x)`
  Costruisce una nuova `Frazione` il cui numeratore è uguale all'argomento e il cui denominatore è 1.

La classe `Frazione`

`Frazione`

Le sue istanze modellano frazioni.

Costruttori

- `public Frazione(int x)`
  Costruisce una nuova `Frazione` il cui numeratore è uguale all'argomento e il cui denominatore è `1`.

- `public Frazione(int x, int y)`
  Costruisce una nuova `Frazione` il cui valore è il rapporto fra il primo argomento e il secondo argomento.

Metodi: operazioni

- `public Frazione piu(Frazione f)`
  Restituisce il riferimento a un nuovo oggetto che rappresenta la frazione ottenuta sommando la frazione specificata come argomento a quella che esegue il metodo.

Metodi: operazioni

- `public Frazione piu(Frazione f)`
  Restituisce il riferimento a un nuovo oggetto che rappresenta la frazione ottenuta sommando la frazione specificata come argomento a quella che esegue il metodo.

- `public Frazione meno(Frazione f)`
  Restituisce il riferimento a un nuovo oggetto che rappresenta la frazione ottenuta sottraendo la frazione specificata come argomento da quella che esegue il metodo.

Metodi: operazioni

- `public Frazione piu(Frazione f)`
  Restituisce il riferimento a un nuovo oggetto che rappresenta la frazione ottenuta sommando la frazione specificata come argomento a quella che esegue il metodo.
- `public Frazione meno(Frazione f)`
  Restituisce il riferimento a un nuovo oggetto che rappresenta la frazione ottenuta sottraendo la frazione specificata come argomento da quella che esegue il metodo.
- `public Frazione per(Frazione f)`

Metodi: operazioni

- `public Frazione piu(Frazione f)`
  Restituisce il riferimento a un nuovo oggetto che rappresenta la frazione ottenuta sommando la frazione specificata come argomento a quella che esegue il metodo.
- `public Frazione meno(Frazione f)`
  Restituisce il riferimento a un nuovo oggetto che rappresenta la frazione ottenuta sottraendo la frazione specificata come argomento da quella che esegue il metodo.
- `public Frazione per(Frazione f)`
- `public Frazione diviso(Frazione f)`

Metodi: operazioni di confronto

- `public boolean equals(Frazione f)`
  Restituisce:
  - `true` se le due frazioni hanno lo stesso valore
  - `false` in caso contrario

Metodi: operazioni di confronto

- `public boolean equals(Frazione f)`
  Restituisce:
  - `true` se le due frazioni hanno lo stesso valore
  - `false` in caso contrario

- `public boolean isMinore(Frazione f)`
  Restituisce:
  - `true` se la frazione che esegue il metodo è minore di quella specificata come argomento
  - `false` in caso contrario

Metodi: operazioni di confronto

- `public boolean equals(Frazione f)`
  Restituisce:
  - `true` se le due frazioni hanno lo stesso valore
  - `false` in caso contrario

- `public boolean isMinore(Frazione f)`
  Restituisce:
  - `true` se la frazione che esegue il metodo è minore di quella specificata come argomento
  - `false` in caso contrario

- `public boolean isMaggiore(Frazione f)`
  Restituisce:
  - `true` se la frazione che esegue il metodo è maggiore di quella specificata come argomento
  - `false` in caso contrario

Metodi

- `public int getNumeratore()`
  Restituisce il numeratore della frazione rappresentata dall'oggetto che esegue il metodo.

Metodi

- `public int getNumeratore()`
  Restituisce il numeratore della frazione rappresentata dall'oggetto che esegue il metodo.

- `public int getDenominatore()`
  Restituisce il denominatore della frazione rappresentata dall'oggetto che esegue il metodo.

Metodi

- `public int getNumeratore()`
  Restituisce il numeratore della frazione rappresentata dall'oggetto che esegue il metodo.
- `public int getDenominatore()`
  Restituisce il denominatore della frazione rappresentata dall'oggetto che esegue il metodo.
- `public String toString()`
  Restituisce una stringa di caratteri che descrive la frazione rappresentata dall'oggetto che esegue il metodo.