Uso di Classi e Oggetti in Java

Emilio Di Giacomo e Walter Didimo

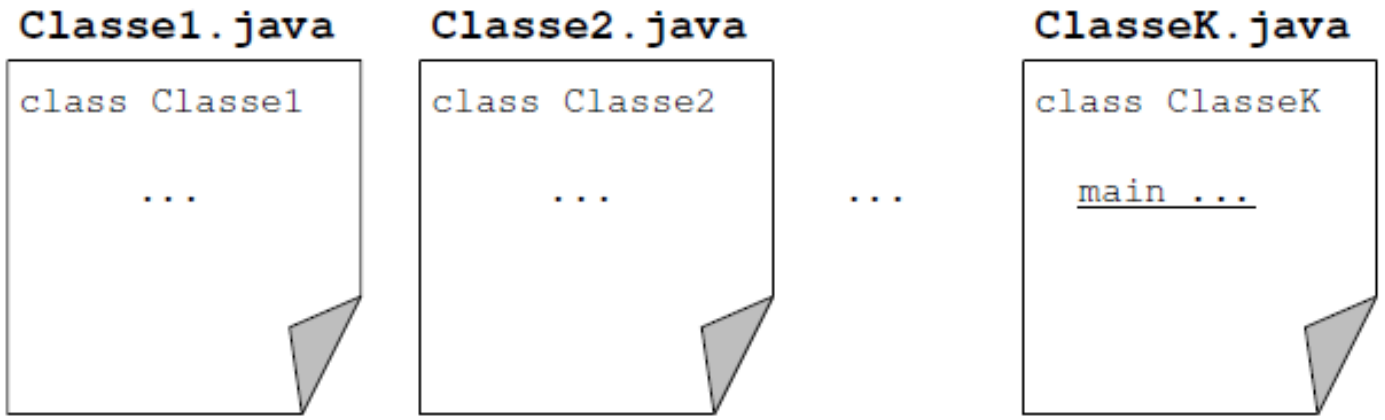
Obiettivi

- Vogliamo vedere come si realizzano semplici programmi in Java
  - In questa fase, gran parte dei programmi che scriveremo sarà già stata realizzata da altri programmatori
  - Noi scriveremo solo piccole parti del programma che utilizzeranno classi di oggetti già definite

Struttura di un programma Java

- Un programma Java è composto da un insieme di classi:
  - il codice di ogni classe va scritto in un *file di testo* che ha il nome della classe e l'estensione *".java"*
  - una delle classi che compongono il programma deve contenere un metodo speciale, chiamato *"main"*
  - l'esecuzione del programma inizia sempre dalla prima istruzione del metodo *main* e termina sempre con l'ultima istruzione del *main*

Struttura di un programma Java

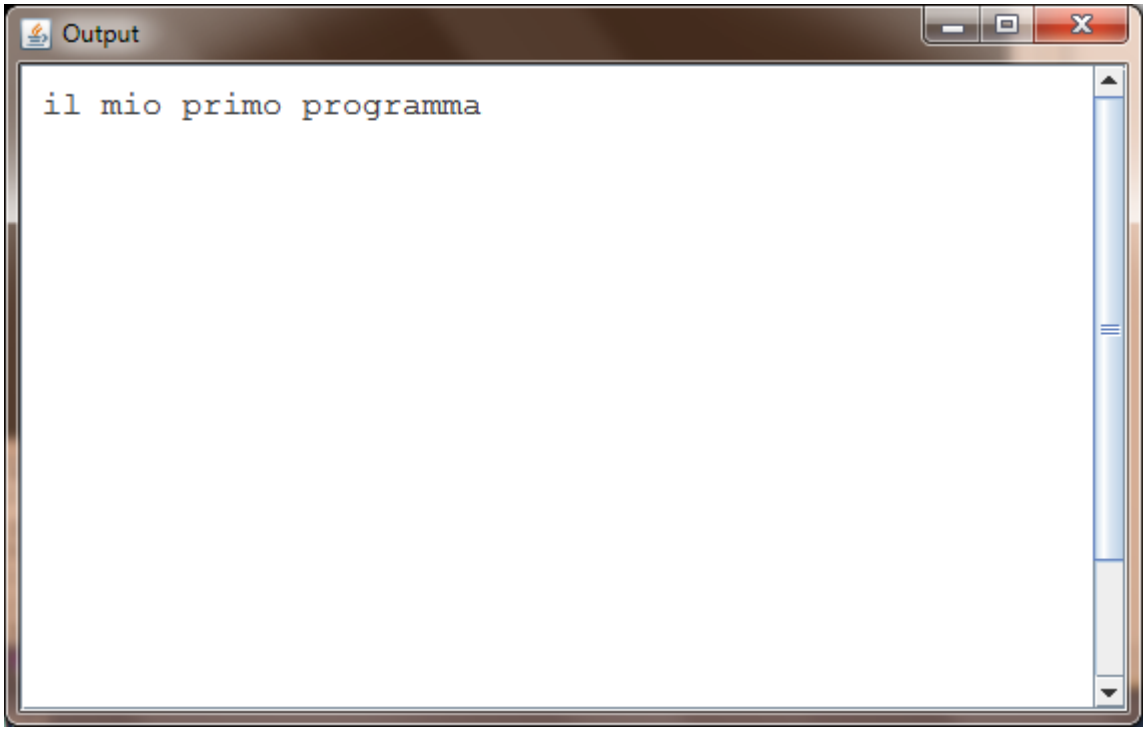


• Il metodo speciale *main* è un metodo di classe; rappresenta infatti un servizio offerto dalla classe che si richiede di “eseguire”

• I file delle varie classi di un programma possono trovarsi distribuiti in diverse directory (come vedremo più avanti)

Il primo programma Java

- Realizziamo un primo programma Java che chiameremo *PrimoProg*
- Esso scriverà a video, all'interno di una finestra grafica, il messaggio "il mio primo programma".



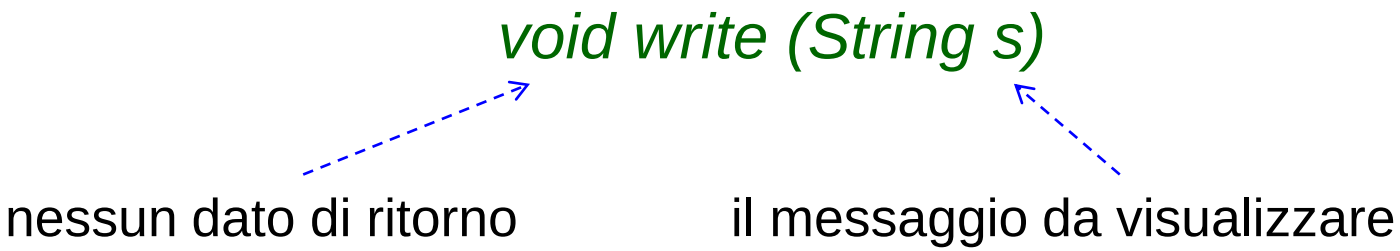
Struttura di PrimoProg

- Il programma sarà composto da tre classi:
  - *String*: è una classe molto usata, fornita dagli sviluppatori di Java; un suo oggetto rappresenta una qualunque sequenza finita di caratteri
  - *OutputWindow*: classe fornita dagli autori di questo corso; un suo oggetto rappresenta una finestra grafica nella quale è possibile scrivere messaggi
  - *PrimoProg*: è la classe che dovremo scrivere noi, ed è quella che conterrà il metodo speciale *main*, dal quale avrà inizio l'esecuzione del programma

La classe OutputWindow

- Come detto, è una classe già scritta e pronta per l'uso; per usarla dobbiamo conoscere il comportamento dei suoi oggetti
  - basta conoscere l'intestazione dei metodi, non la loro implementazione

- Per chiedere ad un oggetto di tipo *OutputWindow* di scrivere un messaggio nella finestra che esso rappresenta possiamo usare il metodo con prototipo seguente:



Codificare una classe

- In Java, la struttura del codice di una classe riflette quella utilizzata con la notazione grafica UML

```
class <nome classe>{
        definizione degli attributi …

        definizione dei metodi …
}
```

corpo

parola chiave del linguaggio Java

La classe PrimoProg

- Per il momento vogliamo imparare a scrivere classi molto semplici:
  - senza attributi
  - con un solo metodo, il metodo speciale *main*

```
class <nome classe>{

        … main (…)

}
```

- La classe *PrimoProg* avrà proprio questa struttura; il suo metodo main dovrà:
  - creare un oggetto di tipo *OutputWindow*
  - invocare il metodo *write* su tale oggetto, passandogli come parametro attuale la stringa “il mio primo programma”

Codice della classe PrimoProg

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma" */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

Nel seguito discuteremo le varie linee di codice del metodo, introducendo i vari concetti del linguaggio Java coinvolti

Direttive di importazione

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma" */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

La parola *import* è una direttiva di importazione; in questo caso dice al compilatore dove prendere la classe *OutputWindow* che si intende utilizzare – analizzeremo più avanti la struttura della stringa *fond.io.OutputWindow*

La parola “class”

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma" */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

La parola *class* inizia la definizione della classe ed è seguita dal nome della classe; il corpo della classe è racchiuso tra due parentesi graffe

Le regole di indentazione stabiliscono quale convenzione utilizzare per “incolonnare” il codice

Modificatori di accesso

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma" */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

La parola *public* è un modificatore di accesso; serve a dire quali altre classi possono utilizzare *PrimoProg*; il modificatore *public* permette ad ogni altra classe di usare *PrimoProg*

Modificatori di accesso

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma" */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

Un modificatore opposto a *public* è il modificatore *private*; una classe può essere *private* solo se dichiarata all'interno di un'altra classe, che in tal caso sarà l'unica che potrà usarla (vedi lezione D16)

Modificatori di accesso

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma" */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

Se non si specifica un modificatore di accesso, la classe potrà essere usata solo da classi del suo stesso *package*; questo concetto sarà chiarito più avanti

Commenti

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma" */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

Un commento è una parte di testo che serve solo a documentare (spiegare) il codice; esso viene ignorato dal compilatore; il commento usato inizia con la sequenza /* e termina con la sequenza */ (può essere anche molto lungo e svilupparsi su più linee)

Commenti di linea

```
import fond.io.OutputWindow;


public class PrimoProg{
  // visualizza "il mio primo programma"
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

Un commento di linea inizia con // e termina automaticamente al primo carattere di accapo (fine linea); può essere usato in alternativa al tipo di commento multilinea, quando il commento è breve

Il metodo “main”: intestazione

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma“ */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

Il metodo speciale *main* ha una intestazione predefinita, che non deve essere cambiata; non ritorna alcun valore (è di tipo *void*) e prende in ingresso un parametro che impareremo a comprendere e a considerare più avanti (vedi lezione D9)

Il metodo “main”: intestazione

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma“ */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

Il metodo speciale *main* è un metodo di classe (cioè statico); per questo viene definito con la parola chiave *static* davanti

Il metodo “main”: intestazione

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma“ */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

Il metodo speciale *main* è di tipo *public*; ogni elemento (metodo o attributo) può essere preceduto da un modificatore di accesso: *public* indica che l’elemento è accessibile da ogni istruzione; *private* indica che l’elemento è accessibile solo da istruzioni di metodi della sua classe

Il metodo “main”: intestazione

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma“ */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

Un terzo modificatore chiamato *protected* definisce un livello intermedio di accesso (spiegato nella lezione D10)
In assenza di specifiche, l’elemento sarà accessibile solo da istruzioni di classi del suo stesso *package* (vedi più avanti)

Il metodo “main”: corpo

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma“ */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

corpo

Il corpo del metodo *main* descrive la sua implementazione, cioè quali istruzioni verranno svolte quando il metodo sarà eseguito; il corpo di un metodo va sempre racchiuso tra due parentesi graffe; ogni istruzione è terminata dal simbolo “;”, detto terminatore di istruzione

Creazione di oggetti

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma“ */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

La prima istruzione del metodo *main* è un’istruzione composta, cioè formata da più istruzioni semplici, che servono a creare un oggetto di tipo *OutputWindow* e a memorizzarne il riferimento in modo da poterlo successivamente utilizzare; analizziamole di seguito

Creazione di oggetti

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma“ */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

L’istruzione evidenziata (a destra del simbolo “=”) crea un oggetto di tipo *OutputWindow* – per chiarire l’istruzione, introduciamo prima un nuovo concetto chiave

Costruttori

- Gli oggetti di una classe possono essere creati solo tramite dei metodi speciali chiamati costruttori
  - il nome di un costruttore deve sempre coincidere con il nome della classe
  - se una classe ha più costruttori, essi debbono differire per la lista di parametri formali
  - un costruttore non ha tipo di ritorno (neanche *void*)
  - un costruttore può essere preceduto da un modificatore di accesso (tipicamente è *public*)
  - un costruttore non può mai essere *static* (è sempre assimilabile ad un metodo di istanza)

Ruolo di un costruttore

- Un costruttore va richiamato opportunamente per poter creare un oggetto della sua classe
- Il principale ruolo di un costruttore è quello di inizializzare lo stato dell'oggetto creato, cioè dare agli attributi dell'oggetto un primo valore

Costruttori in OutputWindow

- La classe *OutputWindow* dispone di vari costruttori; uno di essi è un costruttore senza parametri formali

OutpuWindow ()

- Tale costruttore inizializza lo stato dell'oggetto creato in modo che rappresenti una finestra *vuota* di *dimensioni prefissate* e con titolo *"Output"*

Usare i costruttori: l'istruzione "new"

- Per creare un oggetto tramite un costruttore, è necessario usare la parola chiave *new*, seguita dal nome del costruttore e dagli eventuali parametri attuali che il costruttore si aspetta di ricevere

new <nome costruttore> (<parametri attuali>)

- Ad esempio, per creare un oggetto *OutputWindow* utilizzando il costruttore senza parametri, occorre scrivere l'istruzione:

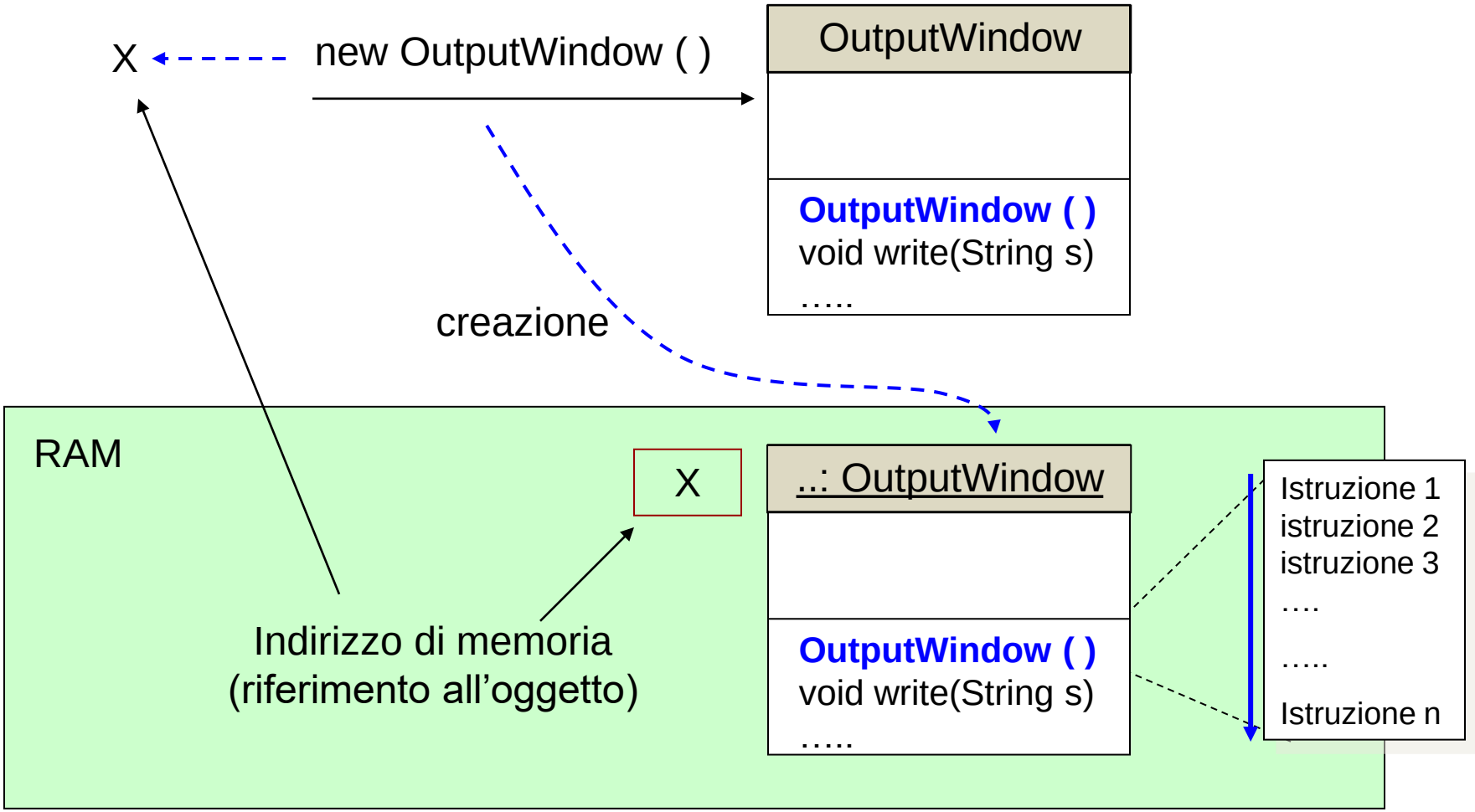
new OutputWindow()

Usare i costruttori: l'istruzione "new"

- Per creare l'oggetto, l'istruzione *new* riserva uno spazio nella memoria fisica (RAM) del calcolatore
  - in tale spazio vengono scritte tutte le informazioni sull'oggetto creato, cioè il suo stato ed i suoi metodi
  - affinché l'oggetto creato possa essere utilizzato, l'istruzione *new* restituisce il suo riferimento (che equivale all'indirizzo della zona di memoria in cui risiede l'oggetto)

- Dopo aver creato l'oggetto, il *new* richiama il costruttore specificato, il quale inizializzerà lo stato di tale oggetto

Creazione oggetti: schema



Operatori ed espressioni

- La parola chiave *new* è un operatore;
- Un operatore è un “simbolo speciale” del linguaggio che viene applicato ad un certo numero di operandi e che restituisce un valore come risultato:
  - l’operatore *new* è un operatore unario, cioè applicato ad un solo operando (il costruttore che lo segue)
  - l’operatore *new* restituisce il riferimento all’oggetto creato, cioè all’istruzione *new* eseguita rimarrà associato il riferimento all’oggetto

- Una istruzione alla quale rimane associato un valore finale è detta espressione

Memorizzazione del riferimento

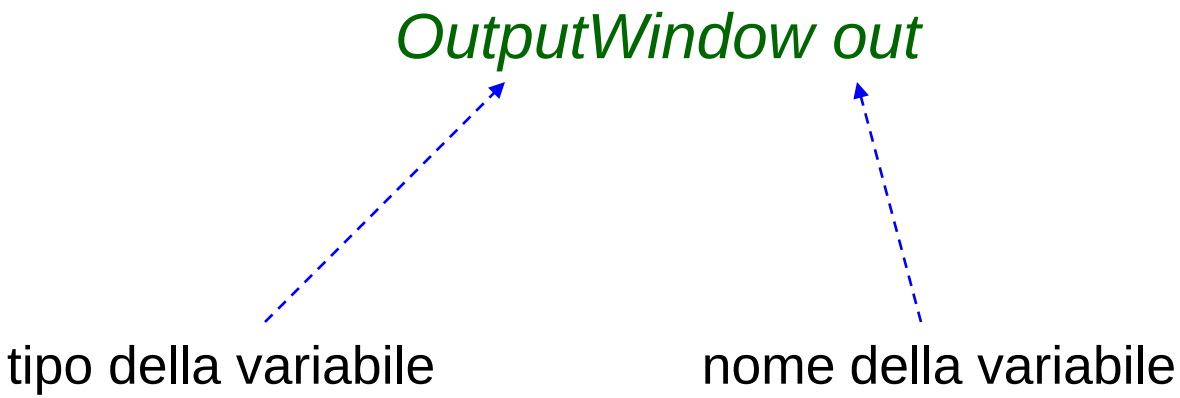
- Per utilizzare l'oggetto di tipo *OutputWindow* creato dobbiamo memorizzarne il riferimento
  - Il riferimento restituito da *new* può essere memorizzato in una variabile
  - una variabile è un "contenitore" che può memorizzare un valore di un certo tipo, detto il tipo della variabile (esattamente come per gli attributi)
  - il valore di una variabile può cambiare nel tempo, ma il suo tipo rimane sempre lo stesso

Variabili riferimento

- Le variabili che permettono di memorizzare il riferimento ad un oggetto di una classe si chiamano variabili riferimento
  - il tipo di una variabile riferimento è la classe degli oggetti che la variabile può referenziare
  - il valore di una variabile riferimento è il riferimento (indirizzo) ad un oggetto del tipo della variabile

Definizione di variabili riferimento

- Una variabile riferimento si definisce specificando il suo tipo (il nome di una classe) ed il suo nome



Assegnamento di valori

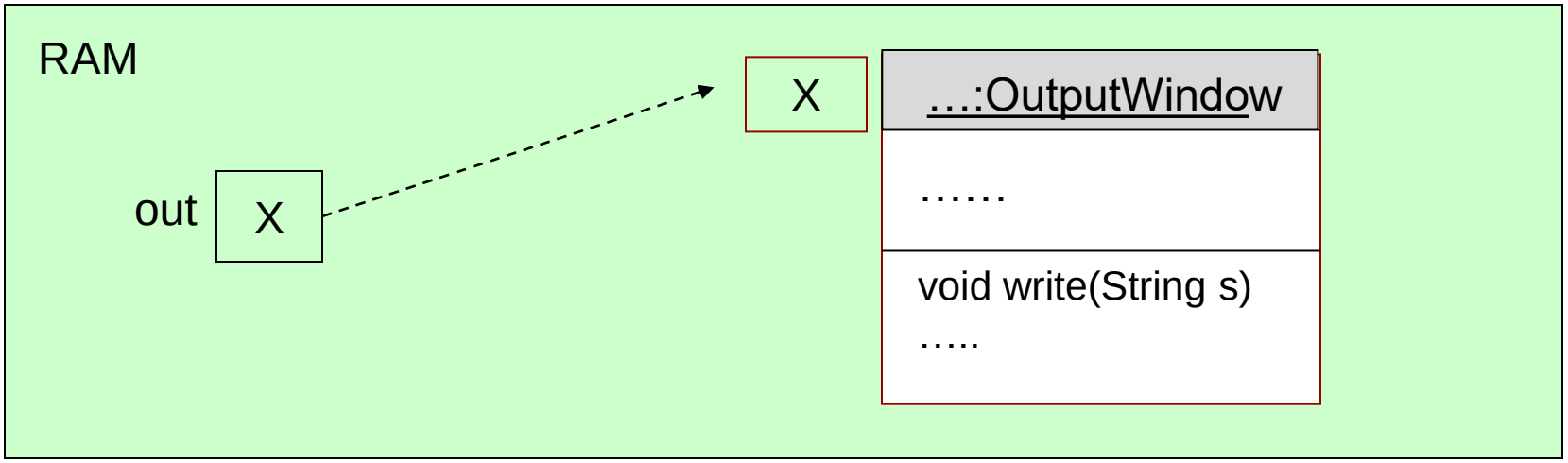
- Per assegnare un valore ad una variabile riferimento si deve usare il simbolo "=" (operatore di assegnazione)

out = <un riferimento OutputWindow>

Variabili riferimento e memoria

• Una variabile riferimento ha una zona di memoria (RAM) associata, che serve per memorizzare il suo valore corrente (il riferimento ad un oggetto)

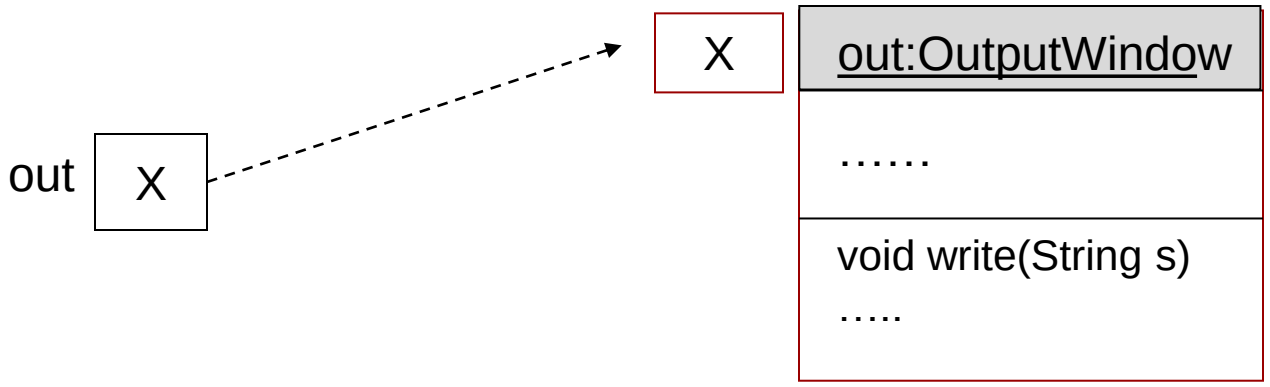
OutputWindow out = new OutputWindow ()



Variabili riferimento e memoria

- Una variabile riferimento può essere usata nel programma per indicare l'oggetto che referenzia;
  - il nome della variabile può essere pensato come un nome dell'oggetto

OutputWindow out = new OutputWindow ()



Scomposizione di istruzioni

- Una istruzione composta può essere scomposta in istruzioni più semplici.
- Ad esempio, l'istruzione

```
OutputWindow out = new OutputWindow();
```

si può riscrivere con due istruzioni semplici:

```
OutputWindow out;
out = new OutputWindow();
```

Invocazione di metodi

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma“ */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

L’ultima istruzione del metodo *main* è l’invocazione del metodo *write* sull’oggetto *out;* la sintassi generale per invocare un metodo su un oggetto è: *<riferimento oggetto>.<nome metodo>(<parametri attuali>)*

Invocazione di metodi

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma“ */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

Quando si invoca il metodo *write*, gli si passa il riferimento ad un oggetto della classe *String*, che come detto rappresenta una qualunque sequenza finita di caratteri

Invocazione di metodi

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma“ */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

In particolare, in luogo di un parametro di tipo *String* si può passare una sequenza di caratteri tra apici doppi, chiamata letterale stringa; il metodo *write* è implementato in modo da far apparire la stringa passatagli come parametro, nella finestra rappresentata dall'oggetto *out*

Invocazione di metodi

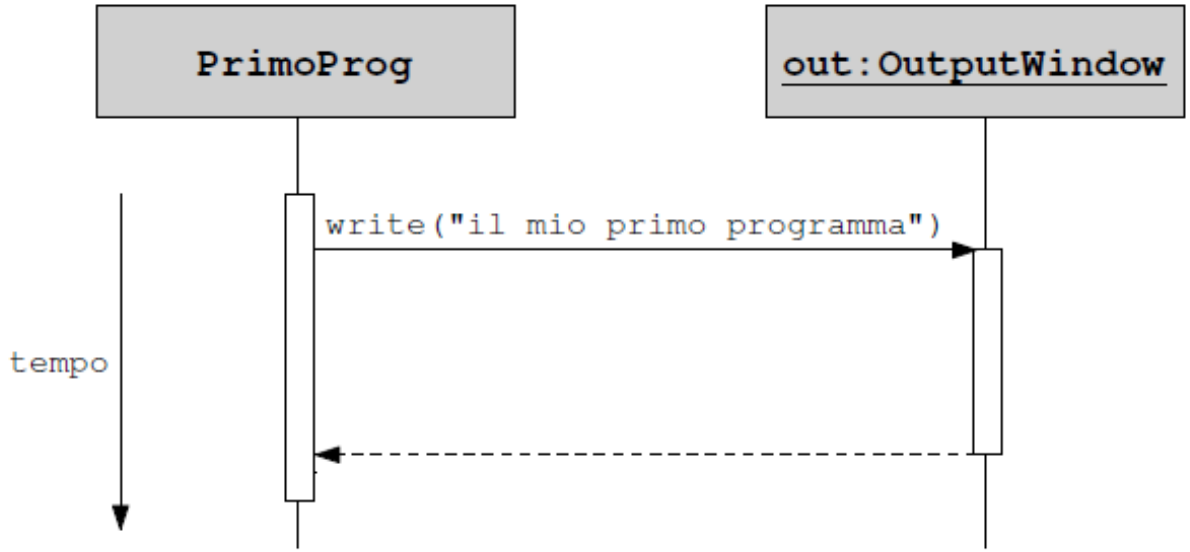
```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma“ */
  public static void main(String[] args){
    OutputWindow out = new OutputWindow();
    out.write("il mio primo programma");
  }
}
```

Quando il metodo *write* viene invocato, il controllo del programma passa temporaneamente dal metodo *main* della classe *PrimoProg* al metodo *write* dell'oggetto out di tipo *OutputWindow*

Diagramma di sequenza

- I diagrammi di sequenza UML illustrano i passaggi di controllo tra le classi e gli oggetti interagenti



Versione compatta di PrimoProg

```
import fond.io.OutputWindow;


public class PrimoProg{
  /* visualizza "il mio primo programma“ */
  public static void main(String[] args){
       new OutputWindow().write("il mio primo programma");
  }
}
```

In questa versione si invoca direttamente il metodo *write* sul riferimento restituito dall'operatore *new* – l'oggetto può essere usato una sola volta, poiché non si è memorizzato il suo riferimento in una variabile

Fase operativa

- Vediamo ora come è possibile operativamente effettuare le seguenti attività:
  - *scrittura* del programma
  - *compilazione* del programma
  - *esecuzione* del programma

- Per alcuni dettagli tecnici si rimanda al testo di riferimento del corso

Scrittura

- Abbiamo detto che ogni classe di nome *C* va scritta in un file di testo distinto di nome *C.java*
  - in realtà questa regola si applica ad ogni classe pubblica (nel nostro caso tutte)

- Le classi *String* e *OutputWindow* sono già state scritte da qualcun altro nei file *String.java* e *OutputWindow.java*

- Dobbiamo dunque scrivere la sola classe *PrimoProg* all'interno di un file di testo di nome *PrimoProg.java*

Package

- Vogliamo ora capire meglio:
  - in che modo si possano utilizzare classi già pronte in *PrimoProg* (o in altre classi che realizzeremo);
  - in che modo sia possibile compilare ed eseguire correttamente programmi composti da più classi

- A tal fine è necessario definire il concetto di package

Package

- Un package in Java rappresenta un gruppo di classi con funzionalità affini (es. grafica, input/output, …)
- Esiste una *associazione tra package e directory*:
  - per definire un package di nome *x*, occorre creare una directory di nome *x*
  - per definire un package *y* interno ad *x*, occorre creare una directory *y* all'interno di *x*
  - ogni classe *C* di un package *x* va scritta in un file *C.java* all'interno della directory *x*; all'inizio del file *C.java* va riportata la specifica: *package x;*

Package di default

- Se in un file *C.java* non è specificato il package di appartenenza, la classe *C* viene associata ad un così detto package di default
  - tutte le classi in una stessa directory che non hanno la specifica di un package vengono associate allo stesso package di default

Utilizzo di package

- Una classe può utilizzare liberamente classi del suo stesso package (a meno che non siano private)
- Una classe può utilizzare solo classi pubbliche di un package diverso dal suo:
  - è necessario utilizzare una direttiva di importazione, che specifichi l'intero percorso della classe che si vuol usare;
  - ad esempio, se una classe *A* deve usare un'altra classe *B* di un package *y* interno ad un package *x*, nel file *A.java* devo scrivere: *import x.y.B*
  - Scrivendo *import x.y.** potrò usare tutte le classi che appartengono direttamente al package *y* all'interno di *x*

API di Java

- Java offre un ampio insieme di classi già pronte per l'uso, suddivise in vari package
- Questa libreria di classi si chiama API (Application Programming Interface) di Java
  - Ogni package dell'API di Java ha come radice un package di nome *java* oppure *javax*

- La classe *String* si trova nel package *java.lang*
  - tale package raggruppa classi di uso molto frequente
  - per usare le classi di questo package non è richiesta la direttiva di importazione

Package del corso

- In questo corso vengono fornite classi aggiuntive a quelle dell'API di Java (es. la classe *OutputWindow*)
- Le classi fornite appartengono tutte ad un package di nome *fond*
  - la classe *OutputWindow* si trova nel package *fond.io*
  - ecco perché per usarla nella classe *PrimoProg* abbiamo usato la direttiva *import fond.io.OutputWindow*

Ambiente di programmazione integrato

- Per scrivere, compilare ed eseguire programmi Java, si possono utilizzare dei software detti «ambienti di programmazione integrati» (IDE)
- In questo corso faremo uso del software gratuito Eclipse (https://www.eclipse.org/), uno degli ambienti di programmazione più potenti e diffusi, sia in ambito accademico che industriale
  - nel materiale didattico integrativo del corso viene fornita una documentazione di dettaglio su come scaricare ed installare Eclipse.

Alternativa: uso della linea di comando

- In alternativa all'uso di un ambiente integrato, si possono compilare ed eseguire programmi java direttamente da linea di comando (prompt del sistema operativo), dopo averli scritti e salvati con un qualunque editor di testi

- In questo caso occorre scaricare (gratuitamente) il software JDK (Java Development Kit) dal sito: *www.oracle.com*

- JDK include tra l'altro:
  - un compilatore Java, che permette di tradurre un file sorgente nel bytecode eseguibile dalla JVM
  - un esecutore Java, che permette di eseguire i bytecode

Alternativa: uso della linea di comando

- Per compilare un file *C.java* si può scrivere in un prompt dei comandi il seguente comando:

  javac C.java

- In assenza di errori sintattici, verrà generato il bytecode all'interno di un file di nome *C.class*

- Se la classe *C* contiene il metodo speciale *main*, è possibile eseguire la classe con il comando:

  java C

Uso del package fond.jar

- Per compilare ed eseguire *PrimoProg* (e gli altri programmi del corso) occorre prima scaricare il package *fond.jar* e poi configurare opportunamente Eclipse o il sistema operativo
  - si rimanda alla documentazione integrativa del corso per dettagli tecnici

- In caso di uso della linea di comando, si potrà compilare la classe *PrimoProg* con il comando *javac PrimoProg.java* e poi eseguirla con il comando *java PrimoProg*

Errori di compilazione

- Se il codice di una classe contiene errori sintattici, il compilatore non produrrà il bytecode, e rileverà tali errori con degli opportuni messaggi:
  - i messaggi di errore del compilatore *aiutano* a comprendere gli errori e a correggerli
  - nel testo di riferimento del corso sono contenuti vari suggerimenti su come interpretare i messaggi del compilatore e su come procedere per correggere errori sintattici

Errori di esecuzione

- Un programma può anche contenere o generare errori in fase di esecuzione, di varia natura:
  - errori di robustezza: dovuti a situazioni limite mal controllate dal programmatore (es. si assegna ad una variabile numerica un valore al di fuori del suo dominio)
  - errori di natura esterna: un evento esterno imprevisto impedisce la corretta prosecuzione del programma
  - errori di logica: il programma fornisce output inesatti a causa di un errore di logica nell'implementazione di qualche algoritmo da parte del programmatore

Ancora sulla classe OutputWindow

- Nel corso useremo diffusamente oggetti di tipo *OutputWindow* per inviare messaggi a video
- La classe offre altri tre costruttori oltre a quello visto
  - *OutputWindow (String s)*: permette di specificare il titolo della finestra di output
  - *OutputWindow (int altezza, int larghezza)*: permette di specificare le dimensioni della finestra di output
  - *OutputWindow (String s, int altezza, int larghezza)*: permette di specificare titolo e dimensioni della finestra di output

Esempio: la classe TreFinestre

```
import fond.io.OutputWindow;

public class TreFinestre{
  /* crea tre diverse finestre di output */
  public static void main(String[] args){
    new OutputWindow(40, 50).write("ciao");
    new OutputWindow("Output1").write("ciao");
    new OutputWindow("Output2", 10, 20).write("ciao");
  }
}
```

Metodi della classe OutputWindow

- Gli oggetti di tipo *OutputWindow* offrono altri metodi utili:
  - un metodo *write* diverso per ogni “possibile” tipo di dato che si vuol visualizzare

```
void write(int i)

void write(double d)

...
```

  - per ogni *write* esiste anche la variante *writeln* che cambia linea alla fine del dato visualizzato
  - un metodo per cambiare tipo e dimensione del font

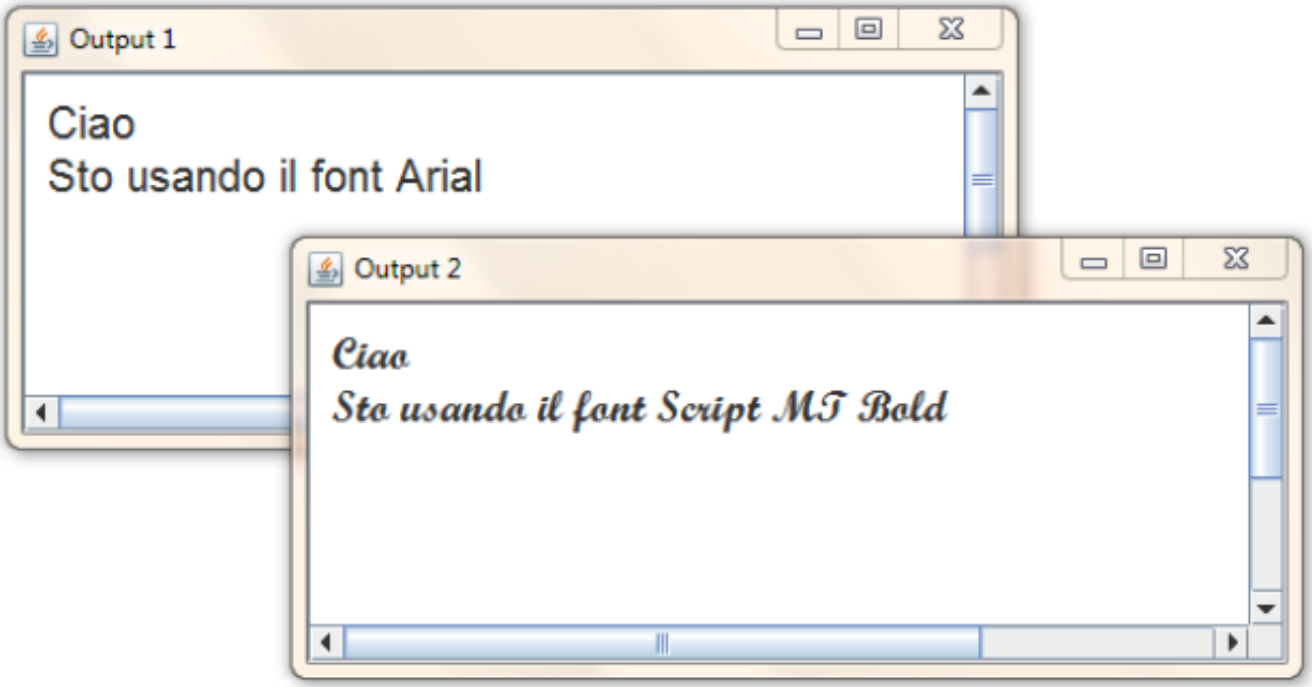
```
void setFont(String f, int dim)
```

Esempio: la classe ProvaFont

```
import fond.io.OutputWindow;

public class ProvaFont{
  /* crea due finestre di output con font diversi */
  public static void main(String[] args){
    OutputWindow out1 = new OutputWindow("Output 1", 10, 40);
    OutputWindow out2 = new OutputWindow("Output 2", 10, 40);
    out1.setFont("Arial", 20);
    out2.setFont("Script MT Bold", 20);
    out1.writeln("Ciao");
    out1.write("Sto usando il font Arial");
    out2.writeln("Ciao");
    out2.write("Sto usando il font Script MT Bold");
  }
}
```

Esecuzione della classe ProvaFont



Lo standard output

- L'API di Java offre una classe di nome *System*, che rappresenta una astrazione del sistema (JVM, sistema operativo, hardware)
  - la classe *System* è definita nel package *java.lang*, e quindi si può usare senza direttive di importazione
- Nella classe *System* è definito un attributo di tipo riferimento, chiamato *out*, che rappresenta il così detto standard output:
  - in Windows, lo standard output è il terminale video testuale (prompt dei comandi) dal quale si lancia il programma
  - lo standard output può essere ridefinito

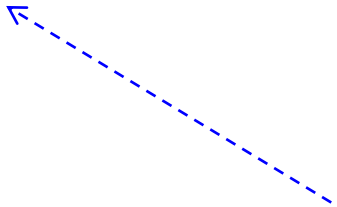
L’oggetto System.out

- Dunque, l’espressione *System.out* referenzia l’oggetto che rappresenta lo standard output
  - *System.out* è un oggetto già pronto per l’uso, e non va creato esplicitamente come gli oggetti *OutputWindow*

- L’oggetto *System.out* ha i metodi *print* e *println*, che svolgono lo stesso ruolo dei metodi *write* e *writeln* degli oggetti *OutputWindow*
  - I messaggi stampati con*System.out* non appaiono però su una finestra grafica personalizzabile, bensì sul prompt dei comandi (finestra testuale)

Esempio di uso di System.out

```
public class CiaoMondo{
   /* stampa "ciao mondo" sullo standard output */
   public static void main(String[] args){
      System.out.print("ciao mondo");
   }
}
```



L'oggetto *System.out* può essere usato direttamente

Altri esempi di uso di oggetti

- Fino ad ora abbiamo visto come inviare messaggi a video
- Molti programmi necessitano anche di acquisire dati dall'esterno (ad esempio dall'utente)

- Vedremo ora altri esempi di uso di oggetti introducendo un nuovo tipo di oggetti che useremo per acquisire dati immessi dall'utente tramite la tastiera

La classe InputWindow

- In questo corso forniamo la classe *InputWindow*, che consente ai programmi di acquisire dati immessi da tastiera
  - la classe *InputWindow* si trova nel package *fond.io*
  - la classe *InputWindow* ha un solo costruttore per creare oggetti, il costruttore che non prende parametri

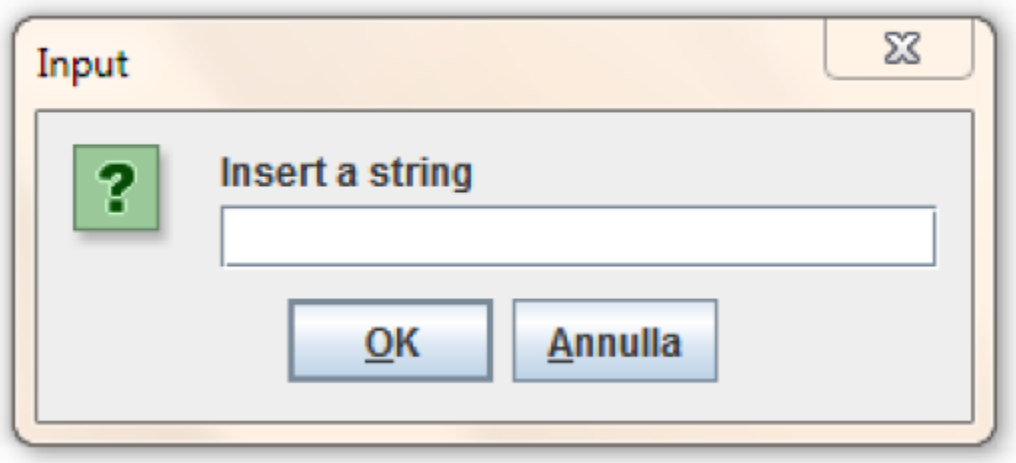
La classe InputWindow: metodi

- Un oggetto di tipo *InputWindow* ha metodi di istanza per acquisire dati di varia natura:
  - *int readInt(); // acquisisce un intero*
  - *double readDouble(); // acquisisce un numero reale*
  - *String readString(); // acquisisce una stringa*

- È possibile acquisire anche altri tipi di dato (di cui parleremo nella lezione D6)

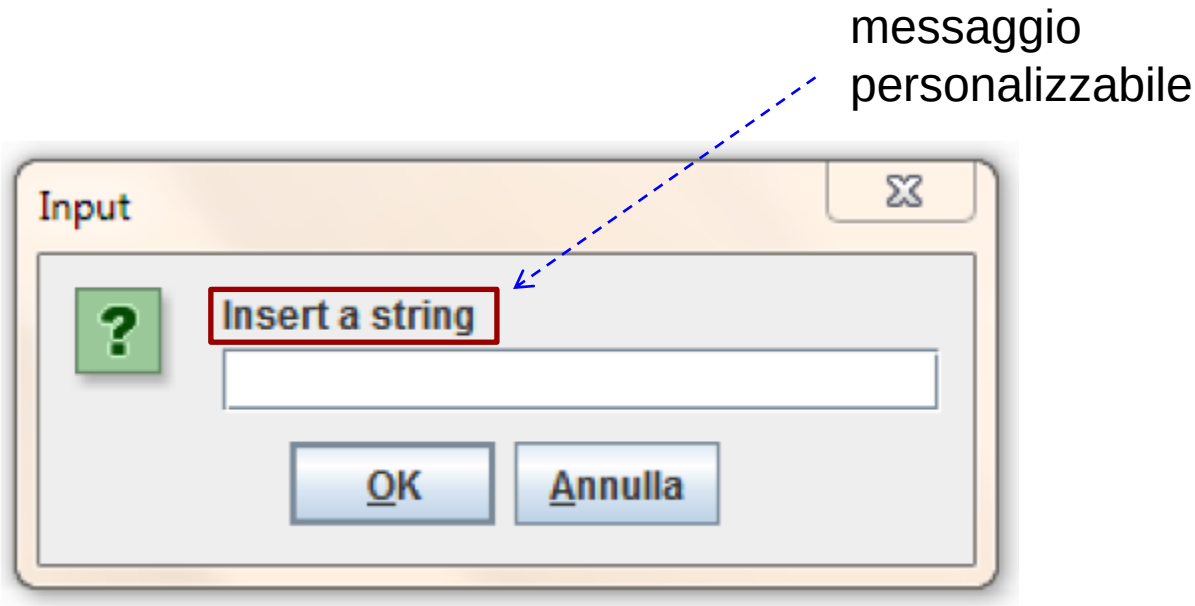
La classe InputWindow: metodi

- Quando si invoca un metodo di acquisizione dati su un oggetto *InputWindow*, tale metodo apre a video un pannello grafico in cui l'utente può inserire il dato
- Ad esempio, invocando il metodo *readString()* apparirebbe il seguente pannello



La classe InputWindow: metodi

- Per ogni metodo *readX()* della classe *InputWindow*, esiste anche la variante *readX(String s)*, che permette di personalizzare il messaggio di richiesta inserimento del dato con la stringa passata come parametro



La classe Echo

- Vogliamo scrivere un programma che:
  - chieda all'utente di inserire una stringa qualsiasi
  - stampi in una finestra grafica la stringa inserita dall'utente
- A tal fine realizzeremo una semplice classe di nome *Echo*, dotata al solito del solo metodo speciale *main*
  - mostreremo dapprima l'intero codice della classe *Echo*, e discuteremo poi le parti più rilevanti

La classe Echo: il codice

```
import fond.io.*;

public class Echo{
  /* visualizza una stringa inserita dall'utente */
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    String str = in.readString();
    OutputWindow out = new OutputWindow("Echo");
    out.write(str);
  }
}
```

Creazione oggetto InputWindow

```
import fond.io.*;

public class Echo{
  /* visualizza una stringa inserita dall'utente */
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    String str = in.readString();
    OutputWindow out = new OutputWindow("Echo");
    out.write(str);
  }
}
```

L'istruzione crea un oggetto di tipo *InputWindow* con l'unico costruttore possibile (quello senza parametri) e memorizza poi il riferimento a tale oggetto in una variabile riferimento di tipo *InputWindow*

Lettura del dato stringa

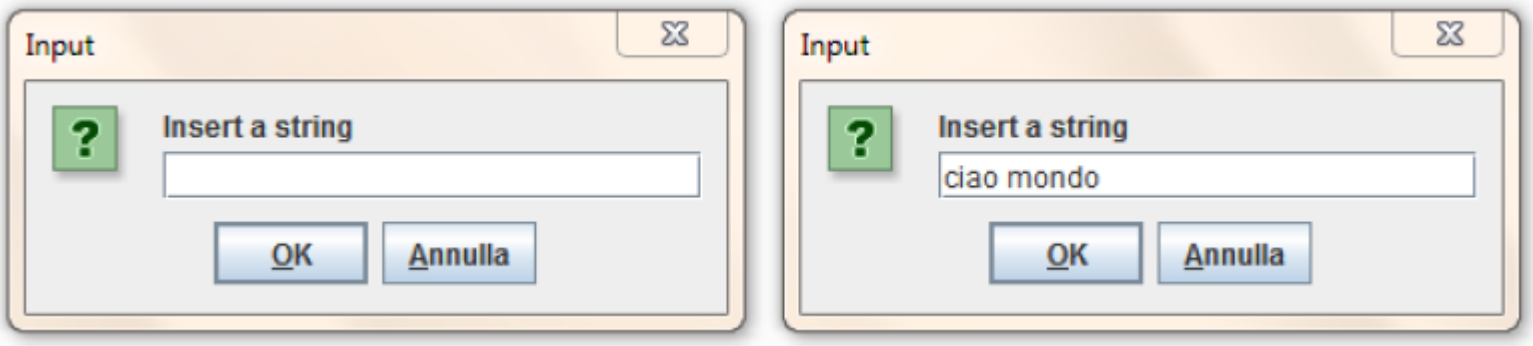
```
import fond.io.*;

public class Echo{
  /* visualizza una stringa inserita dall'utente */
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    String str = in.readString();
    OutputWindow out = new OutputWindow("Echo");
    out.write(str);
  }
}
```

L'istruzione *in.readString()* invoca il metodo *readString()* sull'oggetto *in*. Il metodo causa l'apertura di un pannello di dialogo, in cui l'utente dovrà inserire una stringa; fatto ciò, il metodo restituisce la stringa inserita, la quale viene memorizzata nella variabile riferimento *str*.

Lettura del dato stringa

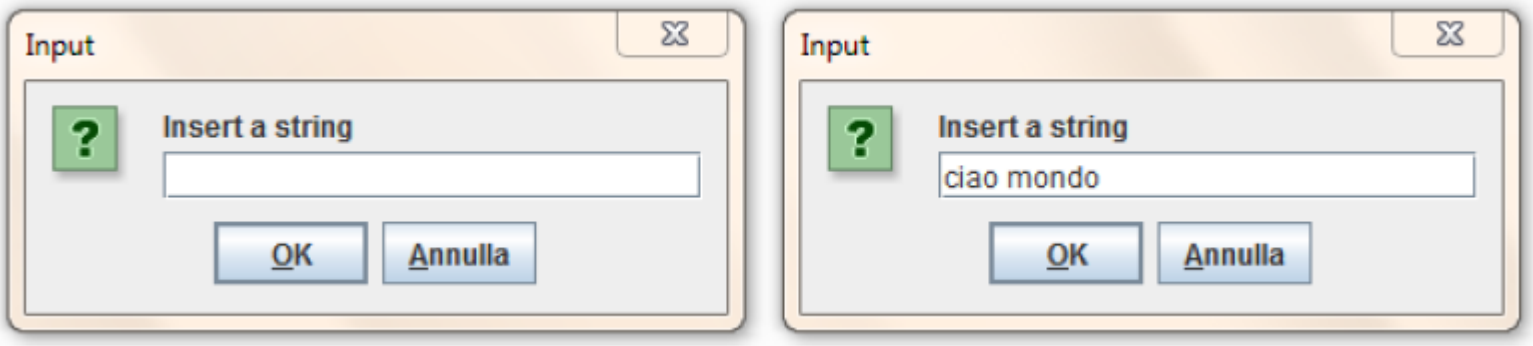
- Ecco come apparirebbe l'output del programma durante l'esecuzione del metodo *readString()*



- Il metodo *readString()* rimane in attesa fino a quando l'utente non conferma l'inserimento del dato, premendo il bottone **OK**.

Lettura del dato stringa

- Ecco come apparirebbe l'output del programma durante l'esecuzione del metodo *readString()*



- Se l'utente premesse **Annulla,** il metodo restituirebbe un riferimento "nullo", denotato dalla parola chiave *null* (torneremo più avanti su questo concetto)

Creazione oggetto OutputWindow

```
import fond.io.*;

public class Echo{
  /* visualizza una stringa inserita dall'utente */
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    String str = in.readString();
    OutputWindow out = new OutputWindow("Echo");
    out.write(str);
  }
}
```

Viene quindi creato un oggetto *OutputWindow*, la cui finestra grafica associata avrà come titolo “Echo”. Il riferimento all’oggetto creato è memorizzato nella variabile riferimento *out*.

Stampa nella finestra grafica

```
import fond.io.*;

public class Echo{
  /* visualizza una stringa inserita dall'utente */
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    String str = in.readString();
    OutputWindow out = new OutputWindow("Echo");
    out.write(str);
  }
}
```

Infine, si invoca su *out* il metodo *write(str)*, al fine di stampare nella finestra grafica la stringa memorizzata da *str*. Si osservi che il parametro *str* non è un letterale!!

Un altro esempio

- Vogliamo realizzare un programma capace di calcolare e fornire le soluzioni reali di una equazione di secondo grado:
  - il programma chiederà all'utente di inserire i coefficienti a, b, c di una equazione della forma: $ax^2 + bx + c = 0$
  - successivamente stamperà le soluzioni reali in una finestra grafica e anche sullo standard output

La classe EquazioneDiSecondoGrado

- Per realizzare il programma possiamo usufruire di una classe già pronta (fornita dagli autori del corso), chiamata *EquazioneDiSecondoGrado*
  - La classe è disponibile nel package *fond.esempi.capitolo5*

- Costruttori e metodi della classe:
  - *EquazioneDiSecontoGrado(double a, double b, double c)*: costruttore per creare un oggetto che rappresenta l'equazione nella forma $ax^2 + bx + c = 0$
  - *void stampaSoluzioni()*: stampa le soluzioni dell'equazione sullo standard output
  - *void stampaSoluzioni(OutputWindow out)*: stampa le soluzioni dell'equazione nella finestra grafica *out* specificata come parametro

La classe CalcolaEquazione

- Scriviamo una classe con il solo metodo speciale *main*, che ci permetterà di avviare il programma

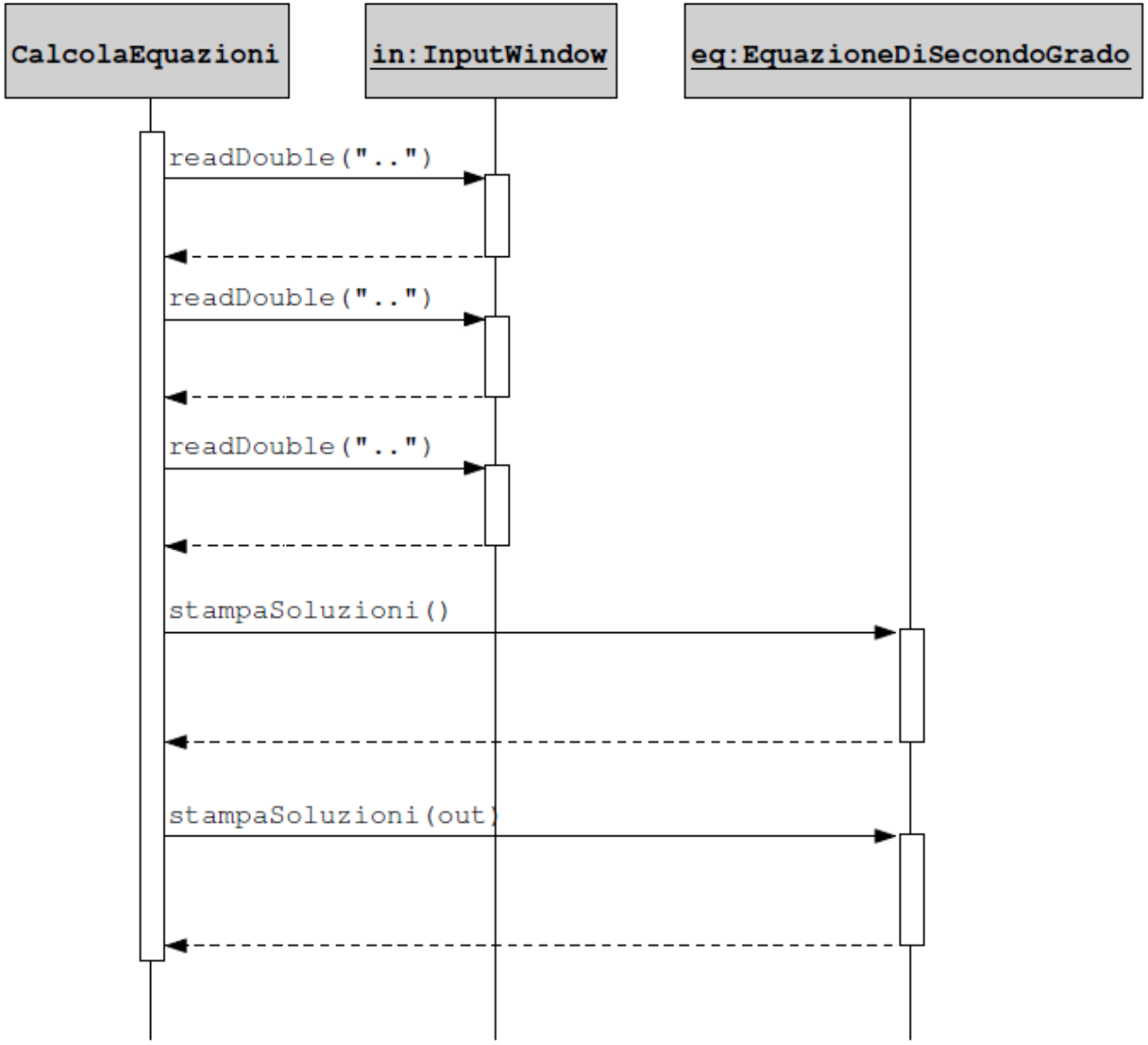
- Chiamiamo questa classe *CalcolaEquazione*; il suo metodo *main* dovrà utilizzare oggetti di tipo: *InputWindow*, *EquazioneDiSecondoGrado*, *OutputWindow* e l'oggetto *System.out*

La classe CalcolaEquazione

```
import fond.io.*;
import fond.esempi.capitolo5.EquazioneDiSecondoGrado;

public class CalcolaEquazione{
  /* calcola le soluzioni reali di equazioni di secondo
   grado espresse nella forma: ax^2 + bx + c = 0.*/
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    double a = in.readDouble("Inserisci il coefficiente a");
    double b = in.readDouble("Inserisci il coefficiente b");
    double c = in.readDouble("Inserisci il coefficiente c");
    EquazioneDiSecondoGrado eq;
    eq = new EquazioneDiSecondoGrado(a, b, c);
    eq.stampaSoluzioni();
    eq.stampaSoluzioni(new OutputWindow());
  }
}
```

Diagramma di sequenza (parziale)



Rimozione di oggetti

- In tutti gli esempi visti sin qui, i programmi si sono solo preoccupati di creare oggetti, ma mai di rimuoverli
- Se durante l'esecuzione di un programma un oggetto diventa inutile, esso andrebbe eliminato dalla memoria, al fine di liberare risorse a disposizione del programma
  - un oggetto diventa inutile quando non ci sono più variabili che lo referenziano (esso diventa irraggiungibile)

Garbage collection

- Java adotta un meccanismo automatico di rimozione degli oggetti inutili, chiamato garbage collection;
  - esiste una sorta di sottoprocesso (“thread demone”), detto garbage collector, che monitora costantemente gli oggetti creati durante l’esecuzione di un programma
  - se il garbage collector rileva che un oggetto è divenuto inutile, lo rimuove alla prima occasione possibile
  - non c’è modo di controllare il garbage collector, ma lo si può “sollecitare” (vedi esempio più avanti)

Uso di metodi statici

- Negli esempi visti sin qui abbiamo solo utilizzato metodi di istanza; sappiamo però che una classe può avere anche metodi statici (cioè metodi di classe)

- Ricordiamo che i metodi statici sono servizi offerti dalla classe e vanno invocati sulla classe con la seguente sintassi generale:

  <nome classe>.<nome metodo> (<parametri>)

Uso di metodi statici

- Durante il corso vedremo vari esempi di metodi statici (ad esempio parlando della classe *String*); per ora mostriamo un semplice esempio di uso di metodi statici
- La classe *System* dispone di vari metodi statici, tra cui i due seguenti:
  - *static long currentTimeMillis()*: restituisce i millisecondi trascorsi dalle ore 0:00 del 1 gennaio 1970 (UTC) (*long* è un sovrainsieme del tipo *int,* come vedremo nella lezione D6)
  - *static void gc()*: sollecita l'esecuzione immediata del garbage collector; la JVM terrà conto di questa chiamata, senza però fornire garanzie che il garbage collector entrerà subito in azione.

Esempio

- Scriviamo un piccolo programma che:
  - chiede all'utente di inserire una stringa;
  - stampa all'utente la stringa inserita e il tempo che lui ha impiegato per inserire la stringa da quando gli viene fatta la richiesta
- Il programma farà uso dei metodi statici:
  - *static long currentTimeMillis()*, per calcolare il tempo impiegato dall'utente ad inserire il dato
  - *static void gc()*, per chiedere alla JVM di rimuovere l'oggetto *InputWindow* con cui si è acquisito il dato, una volta che esso non serve più

Esempio: codice del programma

```
import fond.io.*;

public class ProvaSystem{
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    long t0 = System.currentTimeMillis();
    String str = in.readString();
    long t1 = System.currentTimeMillis();
    long diff = t1-t0;
    in = null;
    System.gc();
    OutputWindow out = new OutputWindow();
    out.write("Stringa inserita: ");
    out.writeln(str);
    out.write("Tempo impiegato per l'inserimento (in ms): ");
    out.writeln(diff);
  }
}
```

Esempio: codice del programma

```
import fond.io.*;

public class ProvaSystem{
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    long t0 = System.currentTimeMillis();
    String str = in.readString();
    long t1 = System.currentTimeMillis();
    long diff = t1-t0;
    in = null;
    System.gc();
    OutputWindow out = new OutputWindow();
    out.write("Stringa inserita: ");
    out.writeln(str);
    out.write("Tempo impiegato per l'inserimento (in ms): ");
    out.writeln(diff);
  }
}
```

crea oggetto *InputWindow*

Esempio: codice del programma

```
import fond.io.*;

public class ProvaSystem{
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    long t0 = System.currentTimeMillis();
    String str = in.readString();
    long t1 = System.currentTimeMillis();
    long diff = t1-t0;
    in = null;
    System.gc();
    OutputWindow out = new OutputWindow();
    out.write("Stringa inserita: ");
    out.writeln(str);
    out.write("Tempo impiegato per l'inserimento (in ms): ");
    out.writeln(diff);
  }
}
```

registra il tempo corrente

Esempio: codice del programma

```
import fond.io.*;

public class ProvaSystem{
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    long t0 = System.currentTimeMillis();
    String str = in.readString();
    long t1 = System.currentTimeMillis();
    long diff = t1-t0;
    in = null;
    System.gc();
    OutputWindow out = new OutputWindow();
    out.write("Stringa inserita: ");
    out.writeln(str);
    out.write("Tempo impiegato per l'inserimento (in ms): ");
    out.writeln(diff);
  }
}
```

effettua la richiesta di inserimento all'utente

Esempio: codice del programma

```
import fond.io.*;

public class ProvaSystem{
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    long t0 = System.currentTimeMillis();
    String str = in.readString();
    long t1 = System.currentTimeMillis();
    long diff = t1-t0;
    in = null;
    System.gc();
    OutputWindow out = new OutputWindow();
    out.write("Stringa inserita: ");
    out.writeln(str);
    out.write("Tempo impiegato per l'inserimento (in ms): ");
    out.writeln(diff);
  }
}
```

registra di nuovo il tempo corrente

Esempio: codice del programma

```
import fond.io.*;

public class ProvaSystem{
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    long t0 = System.currentTimeMillis();
    String str = in.readString();
    long t1 = System.currentTimeMillis();
    long diff = t1-t0; <----------------------------
    in = null;
    System.gc();
    OutputWindow out = new OutputWindow();
    out.write("Stringa inserita: ");
    out.writeln(str);
    out.write("Tempo impiegato per l'inserimento (in ms): ");
    out.writeln(diff);
  }
}
```

calcola la differenza tra i tempi registrati (tempo trascorso)

Esempio: codice del programma

```
import fond.io.*;

public class ProvaSystem{
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    long t0 = System.currentTimeMillis();
    String str = in.readString();
    long t1 = System.currentTimeMillis();
    long diff = t1-t0;
    in = null;
    System.gc();
    OutputWindow out = new OutputWindow();
    out.write("Stringa inserita: ");
    out.writeln(str);
    out.write("Tempo impiegato per l'inserimento (in ms): ");
    out.writeln(diff);
  }
}
```

annulla il riferimento memorizzato nella variabile *in* (da questo momento l'oggetto *InputWindow* non è più raggiungibile, quindi diventa inutile)

Esempio: codice del programma

```
import fond.io.*;

public class ProvaSystem{
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    long t0 = System.currentTimeMillis();
    String str = in.readString();
    long t1 = System.currentTimeMillis();
    long diff = t1-t0;
    in = null;
    System.gc();
    OutputWindow out = new OutputWindow();
    out.write("Stringa inserita: ");
    out.writeln(str);
    out.write("Tempo impiegato per l'inserimento (in ms): ");
    out.writeln(diff);
  }
}
```

sollecita la rimozione degli oggetti inutili (quindi dell'oggetto *InputWindow*)

Esempio: codice del programma

```
import fond.io.*;

public class ProvaSystem{
  public static void main(String[] args){
    InputWindow in = new InputWindow();
    long t0 = System.currentTimeMillis();
    String str = in.readString();
    long t1 = System.currentTimeMillis();
    long diff = t1-t0;
    in = null;
    System.gc();
    OutputWindow out = new OutputWindow();
    out.write("Stringa inserita: ");
    out.writeln(str);
    out.write("Tempo impiegato per l'inserimento (in ms): ");
    out.writeln(diff);
  }
}
```

stampa i dati a video