Introduzione all'uso degli oggetti in Java (parte I)

Walter Didimo

Java

Java è un linguaggio di programmazione orientato agli oggetti; nel seguito vedremo:

- come sono strutturati i programmi Java
- come si scrivono e si eseguono programmi Java

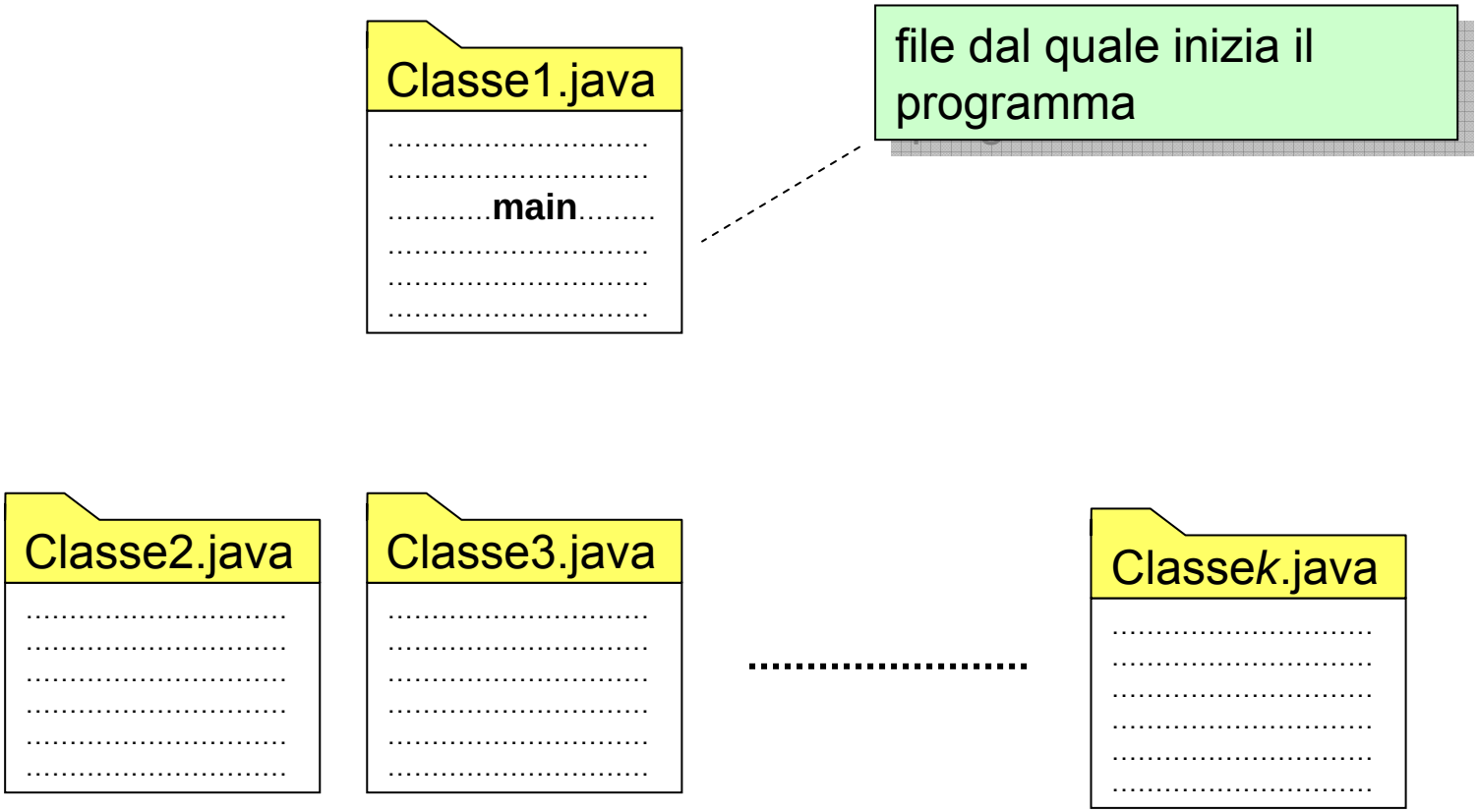
Contestualmente, vedremo come usare classi e oggetti già pronti per scrivere semplici programmi

Struttura di programmi Java

Un programma Java è composto da un insieme di classi:

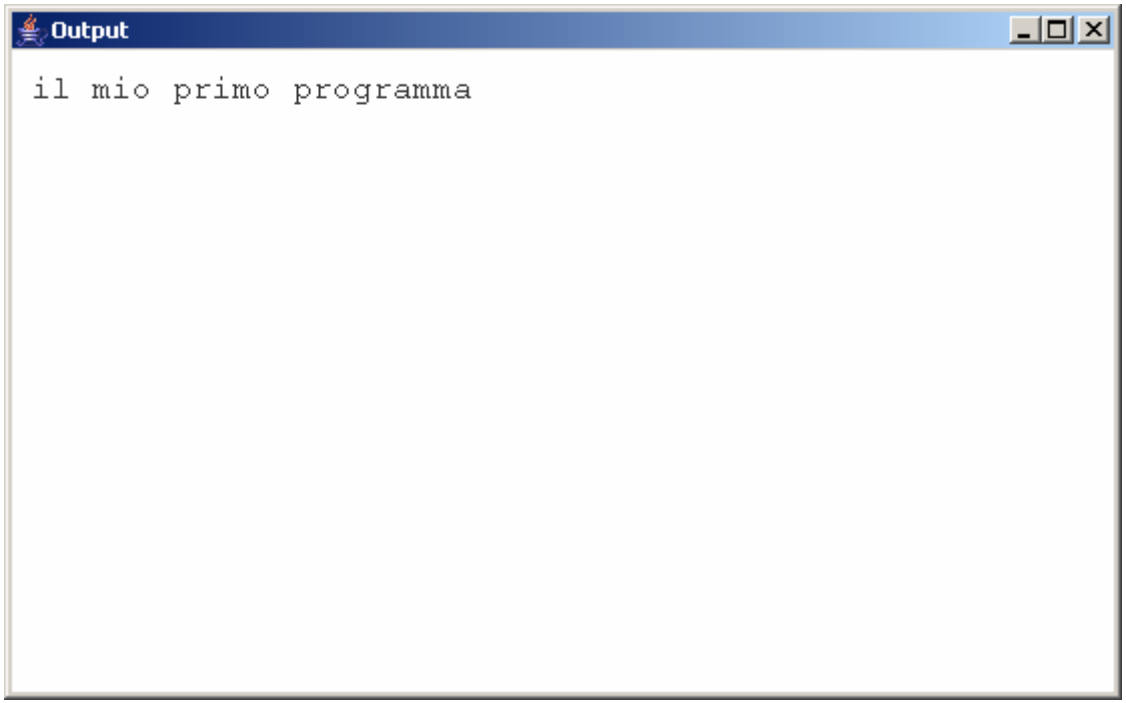
- il codice di ogni classe va scritto in un *file di testo* che ha il nome della classe e l'estensione *".java"*
- una delle classi che compongono il programma deve contenere un metodo speciale, chiamato *"main"*
- l'esecuzione del programma inizia sempre dalla prima istruzione del metodo *main* e termina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

Struttura di programmi Java



Il primo programma Java

Vogliamo realizzare il nostro primo programma.
Il programma dovrà visualizzare il messaggio
“il mio primo programma” in una finestra a video



Struttura del programma

Il programma si comporrà di due classi:

- la classe *OutputWindow*, che descrive oggetti capaci di visualizzare messaggi in una finestra sullo schermo:
  - ogni oggetto *OutputWindow* rappresenta cioè una finestra grafica sullo schermo, nella quale è possibile scrivere messaggi
- la classe *Avvio*, che conterrà il metodo *main* e darà inizio al programma

La classe OutputWindow

Un oggetto della classe *OutputWindow* sarà in grado di offrire servizi di visualizzazione in una finestra sullo schermo

- la classe *OutputWindow* è già stata definita (appositamente per questo corso) – non resta che usarla
- per usare la classe *OutputWindow* (o meglio i suoi oggetti) non serve conoscere come è fatto il codice della classe - è sufficiente conoscere i prototipi dei suoi metodi ed i servizi ad essi associati

La classe OutputWindow

E’ possibile chiedere ad un oggetto della classe *OutputWindow* di visualizzare un messaggio predefinito, invocando un suo metodo di nome *write* – il metodo ha il seguente prototipo

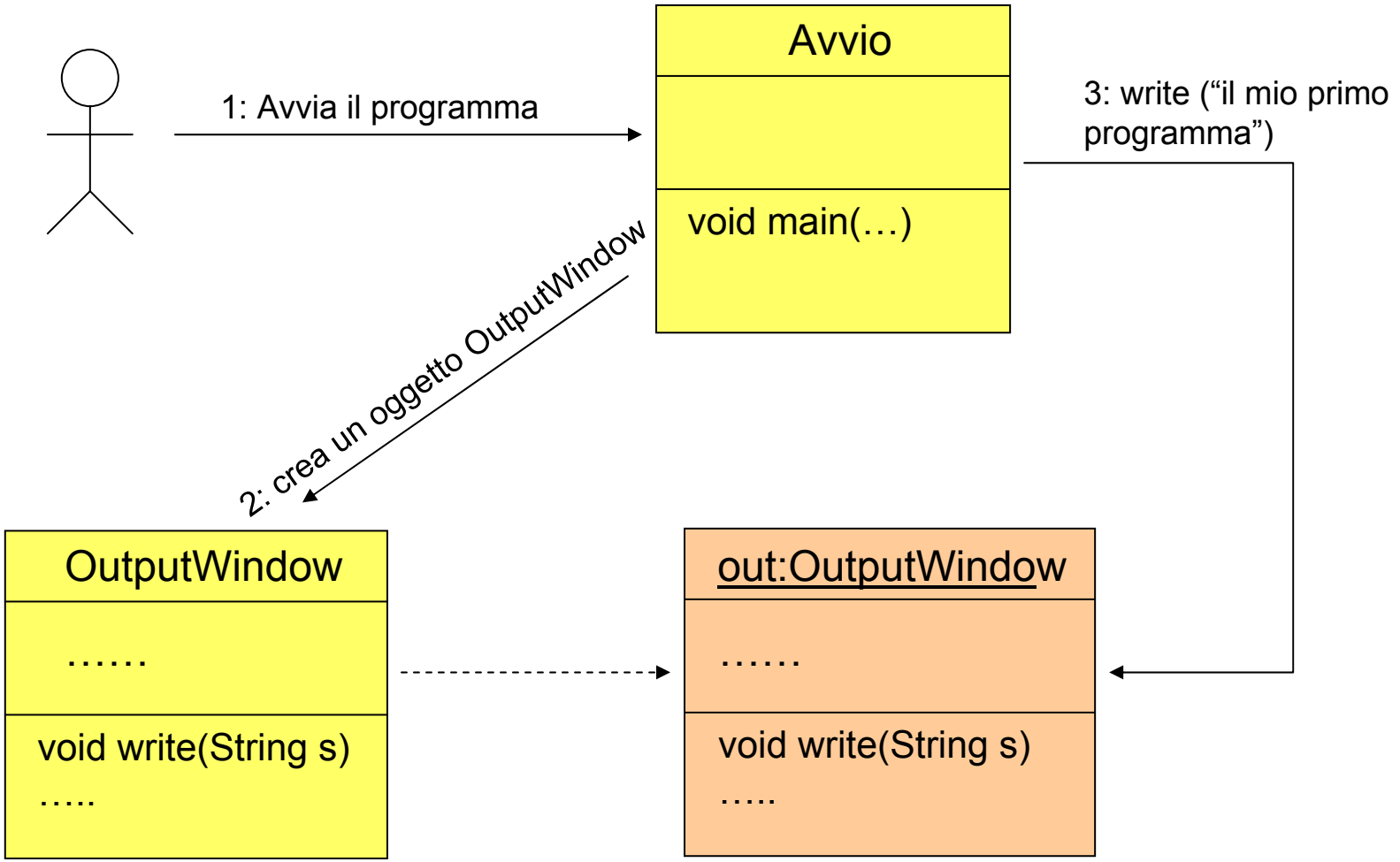
tipo dato di ritorno	*nome metodo*	*tipo parametri*
void	write	(String s)

non restituisce nulla

stringa alfanumerica
(il messaggio da visualizzare)

Lo schema del programma

Diagramma di collaborazione UML



Dallo schema al codice

Una volta capito lo schema logico, il programma va scritto tramite il linguaggio Java

- il codice della classe *OutputWindow* è già stato scritto, e si trova in un file *OutputWindow.java*
- non ci interessa vedere il contenuto del file *OutputWindow.java*
- dobbiamo scrivere il codice della classe *Avvio*, in un file *Avvio.java*

Codificare una classe

La scrittura del codice di una classe consiste nel definire in linguaggio Java i campi ed i metodi dei suoi oggetti

La struttura del codice di una classe segue quella vista fino ad ora negli schemi logici

```
class <nomeDellaClasse>{
        definizione dei campi ….

        definizione dei metodi ….
}
```

Codificare una classe

Imparare a definire e codificare una classe nei dettagli sarà oggetto di una successiva lezione

Per ora ci vogliamo solo concentrare sulla definizione della classe *Avvio*:

- la classe *Avvio* è una classe semplificata
- non definisce campi
- definisce solo il metodo speciale *main*, dal quale inizia e finisce il programma

Il codice della classe Avvio

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

Discutiamo nel seguito le varie linee di codice della classe *Avvio*

La parola chiave “class”

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

In Java, la definizione di una classe inizia con la parola chiave class (tutta in minuscolo), seguita dal nome della classe

Il corpo della classe

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

corpo della classe

I campi ed i metodi definiti dalla classe vanno scritti di seguito al nome, compresi tra due **parentesi graffe** – la parte tra le parentesi graffe si chiama corpo della classe

Commenti

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

Il messaggio tra i due simboli **/*** , ***/** è un commento.
Non ha nessuna rilevanza ai fini del funzionamento del programma, ma può servire ai programmatori per capire cosa fa il codice

Altri modi di commentare

Oltre al commento con i simboli **/*** , ***/** si possono usare altre forme di commento in Java.

Ad esempio si può scrivere

```
// questo è un commento
```

Un commento che inizia con // si chiama commento di linea, perché si estende fino alla fine della linea da cui inizia

Il metodo “main”

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

Il metodo main ha un **prototipo predefinito** (deve essere sempre lo stesso) – ci sono inoltre alcune “**parole**” aggiuntive che vanno inserite all’inizio

Il metodo “main”

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

Per ora non siamo in grado di capire tutta la definizione del metodo *main* – accettiamola per come è, ma facciamo alcune osservazioni

Il metodo “main”

```
class Avvio{
  /* 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 una parola chiave del linguaggio Java – si mette davanti ai metodi che si vuole siano invocabili (fruibili) da qualunque oggetto o classe

Il metodo “main”

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

Non consideriamo per ora la parola chiave **static** – il suo significato sarà chiarito più avanti nel corso

Il metodo “main”

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

Il metodo *main* non restituisce dati a chi lo invoca (il tipo di ritorno è **void**)

Il metodo “main”

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

Il metodo *main* prende in ingresso un parametro (**String[ ] args**) che al momento non siamo in grado di comprendere – possiamo ignorarlo

Il corpo del metodo “main”

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

corpo del metodo main

Il corpo del metodo *main* consiste nella sequenza di istruzioni che esso deve eseguire; il corpo di un metodo è sempre scritto tra due **parentesi graffe** che seguono il suo prototipo

Creazione di oggetti

Il metodo *main* deve chiedere ad un oggetto della classe *OutputWindow* di visualizzare il messaggio "il mio primo programma".

La classe *OutputWindow* definisce le caratteristiche dei suoi oggetti:

- come sono fatti;
- cosa possono fare e come lo debbono fare.

Per usare un oggetto di tipo *OutputWindow*, il metodo *main* deve prima crearne uno

Costruttori

Una classe ha solitamente uno o più metodi speciali, chiamati costruttori:

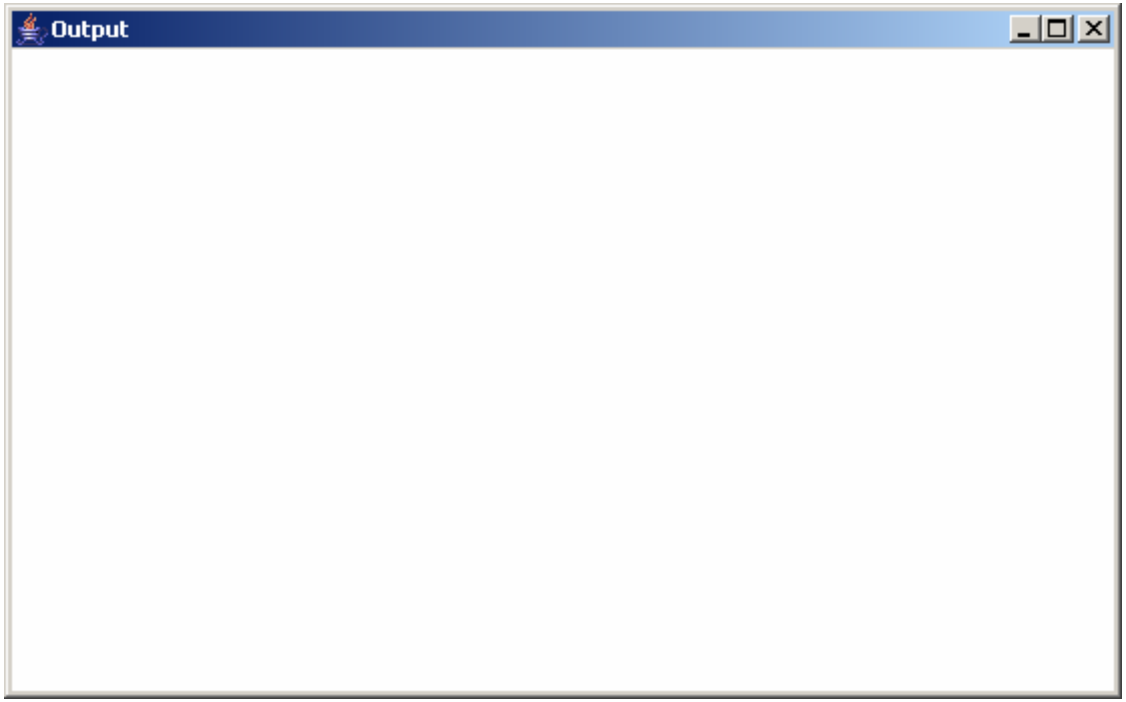
- Un costruttore di una classe serve a creare un suo oggetto (cioè una sua istanza); esso esegue operazioni che *inizializzano* lo stato dell'oggetto creato, cioè che danno un primo valore ai suoi campi

- Il costruttore di una classe ha sempre lo stesso nome della classe ed il suo prototipo non ha alcun tipo di dato di ritorno (neanche *void*)

Creazione di oggetti OutputWindow

La classe *OutputWindow* ha un costruttore che non prende parametri

- permette di creare un oggetto *OutputWindow* che rappresenta una finestra vuota di dimensioni fissate



L’operatore new

Per creare un oggetto tramite un costruttore di una classe, occorre usare un operatore Java chiamato new, seguito dal nome del costruttore

new OutputWindow( )

L’operatore *new* crea l’oggetto e invoca il costruttore, cioè fa sì che tutte le istruzioni del costruttore siano eseguite

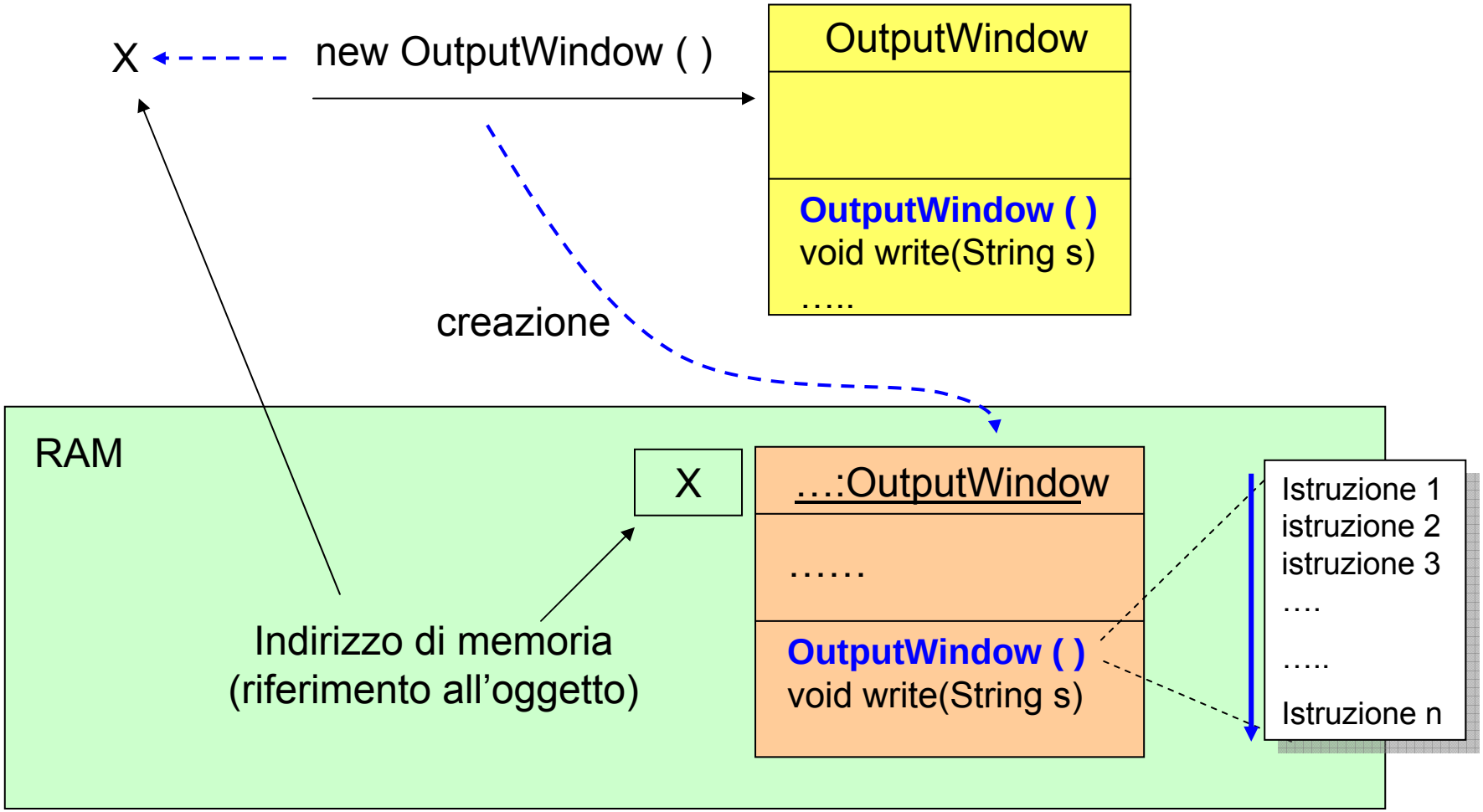
L’operatore new

Per creare l’oggetto, l’operatore 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’operatore *new* restituisce il suo riferimento (che equivale all’indirizzo della zona di memoria in cui risiede l’oggetto)

L’operatore new - schema



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 permette di memorizzare un valore di un certo tipo, detto il tipo della variabile
- il valore di una variabile può cambiare nel tempo, ma il suo tipo rimane sempre lo stesso

Variabili riferimento

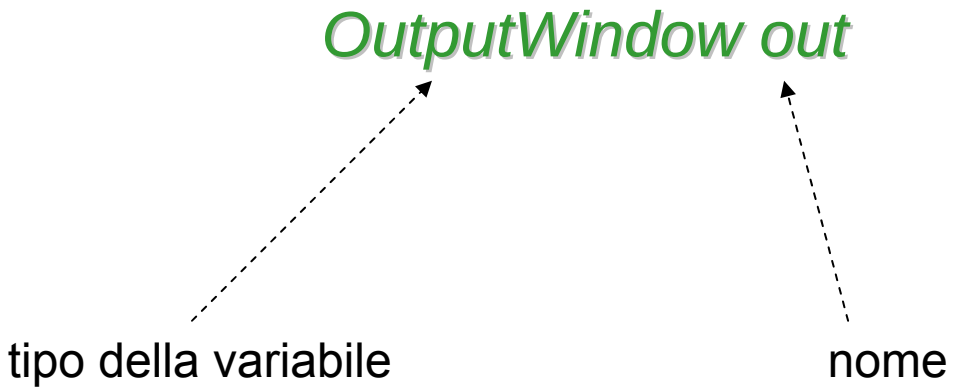
Le variabili che permettono di memorizzare il riferimento ad un oggetto di una classe si chiamano variabili riferimento

- 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

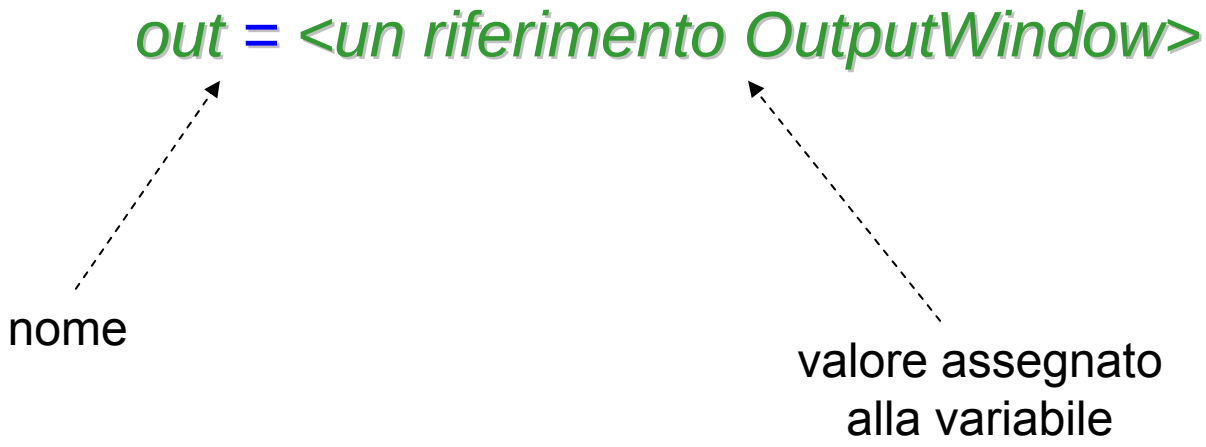
Definizione di variabili riferimento

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



Assegnamento di valori

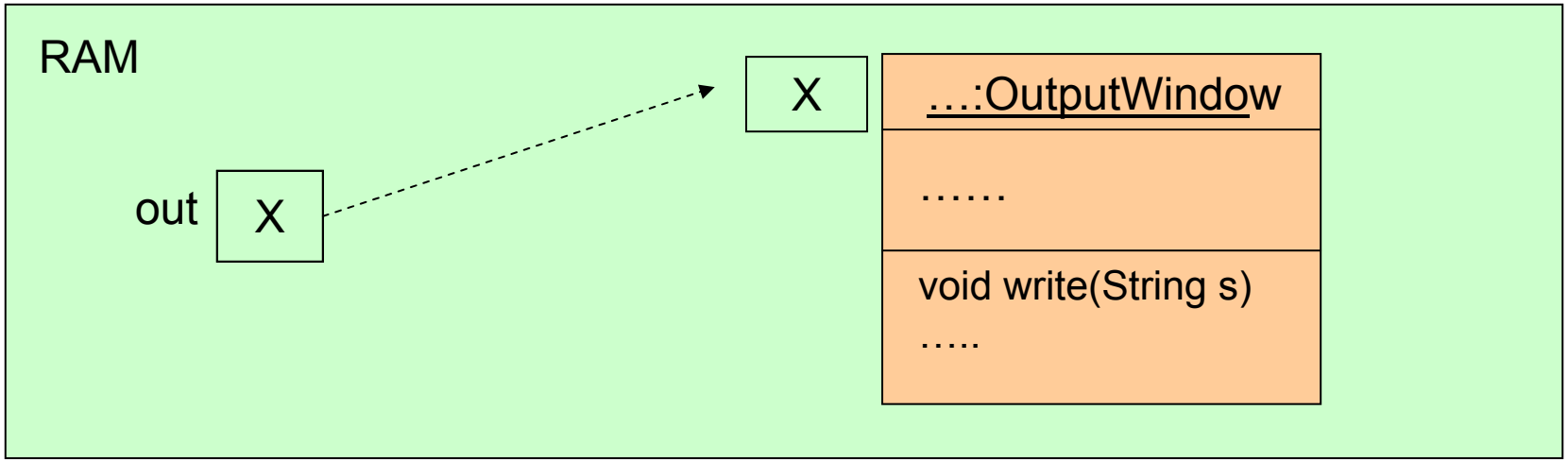
Per assegnare un valore ad una variabile riferimento si deve usare il simbolo “=” (operatore di assegnazione)



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 ()

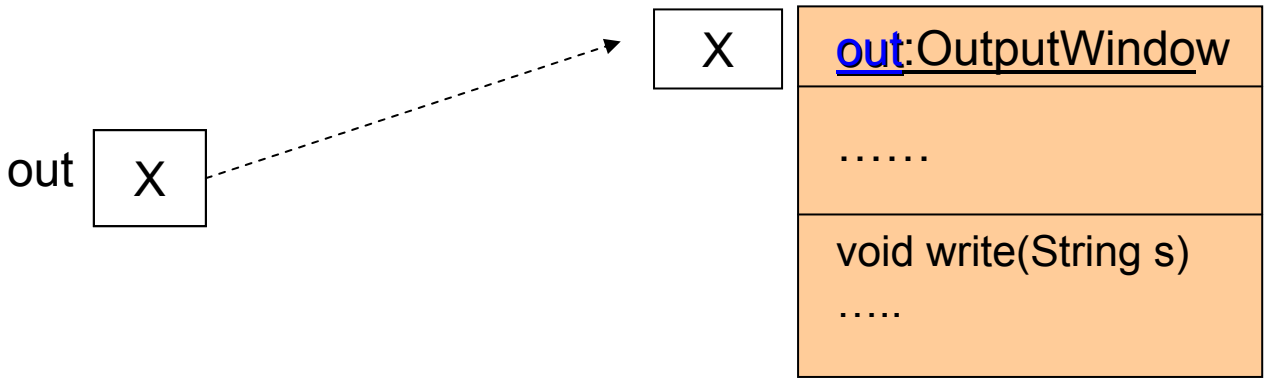


Variabili riferimento e oggetti

Una variabile riferimento può essere usata nel programma per indicare l'oggetto che referenzia;

- il nome della variabile può essere pensato come il nome dell'oggetto

OutputWindow out =new OutputWindow ()



Creazione di oggetti OutputWindow

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

Un oggetto *OutputWindow* viene creato con l’operatore *new* seguito da un costruttore – il riferimento restituito da *new* viene memorizzato in una opportuna variabile riferimento tramite l’operatore di assegamento

Istruzioni e terminatori di istruzioni

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

Il simbolo **“;”** è un terminatore di istruzione, cioè segnala la fine di una istruzione – l’istruzione segnalata è una istruzione composta; crea un oggetto, crea una variabile, assegna alla variabile il riferimento all’oggetto

Un codice equivalente

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

L’istruzione può essere spezzata in due istruzioni distinte, producendo un effetto del tutto equivalente

Invocare un metodo

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

Per invocare il metodo **write** sull’oggetto *OutputWindow* creato, si deve usare un riferimento all’oggetto seguito da un punto e dal nome del metodo con i suoi parametri

Il parametro di write

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

Il metodo *write* vuole come parametro una stringa, cioè una sequenza di caratteri alfanumerici – i **valori stringa** (detti letterali stringa) si specificano tra apici doppi **“ ”**

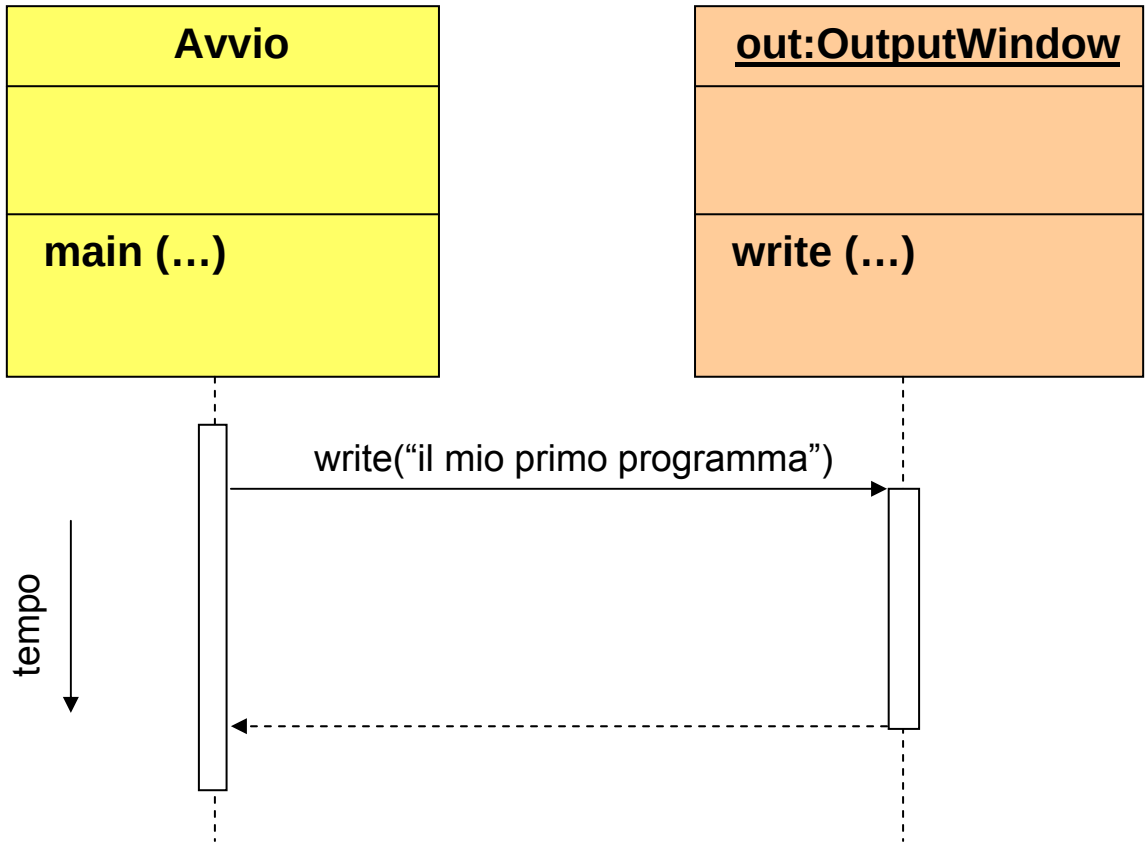
Controllo del programma

Quando un metodo A invoca un metodo B, il controllo del programma passa temporaneamente da A a B:

- quando B termina le sue istruzioni, il controllo del programma torna ad A, a partire dalla istruzione successiva all'invocazione di B

Controllo del programma - schema

Un diagramma sequenza UML schematizza il controllo del programma nel tempo



Scrivere ed eseguire il programma

Per realizzare fisicamente il programma visto occorre:

- comporre (cioè scrivere) i file di testo che lo formano (un file per ogni classe)
- compilare ciascun file, cioè tradurre i file scritti nel linguaggio Java in file scritti in linguaggio macchina (bytecode)
- eseguire il bytecode della classe che contiene il metodo main (la classe *Avvio* nel nostro caso)

Comporre i file Java

Un file Java è un normale *file di testo* (file ASCII) e può dunque essere scritto con un comune *editor di testi*:

- il file *OutputWindow.java* è fornito in questo corso, e non va quindi scritto
- basta scrivere il file *Avvio.java*
- per far funzionare il programma è necessario che i file *OutputWindow.java* ed *Avvio.java* stiano in uno stesso direttorio (cioè in una stessa cartella)

Compilare i file Java

I file Java vanno tradotti (compilati) in file scritti in un linguaggio eseguibile da una "macchina virtuale", nota come Java Virtual Machine

- Per compilare in automatico i file *OutputWindow.java* e *Avvio.java* basta scrivere

```
javac OutputWindow.java
javac Avvio.java
```

- I file compilati si chiamano bytecode, ed hanno i seguenti nomi *OutputWindow.class* ed *Avvio.class*

Ancora sulla compilazione

Il comando *javac* lancia il compilatore Java, cioè un traduttore automatico di file Java in bytecode

- la compilazione va a buon fine solo se non ci sono errori sintattici nel programma, cioè solo se il programma è corretto dal punto di vista della sintassi Java

- se il compilatore rileva errori di sintassi, allora segnala tali errori, suggerendo talvolta come correggerli

Eseguire il programma

Per eseguire il programma occorre eseguire il file *Avvio.class*

- ciò viene fatto scrivendo il comando
  java Avvio

- il comando *java* lancia l'interprete Java, cioè un software che traduce ogni istruzione di bytecode in una equivalente istruzione del processore su cui il programma Java deve essere eseguito

Ulteriori considerazioni

Il codice del metodo *main* della classe *Avvio* può essere ulteriormente semplificato

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

All’espressione `new OutputWindow()` rimane infatti associato il valore restituito dal new, cioè il riferimento all’oggetto creato

Ancora sui costruttori di OutputWindow

La classe *OutputWindow* ha altri costruttori oltre a quello visto; ecco l'elenco di tutti i costruttori della classe:

/*titolo e dimensioni di default*/

OutputWindow ( )

/*titolo specificato e dimensioni di default*/

OutputWindow (String s)

/*titolo di default e dimensioni specificate*/

OutputWindow (int rows, int cols)

/*titolo e dimensioni specificate*/

OutputWindow (String s, int rows, int cols)

Esempio di uso dei costruttori

Il seguente programma crea 3 finestre di output differenti, utilizzando diversi costruttori:

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

Ancora sui metodi di OutputWindow

Gli oggetti di tipo *OutputWindow* offrono altri metodi (servizi) 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

- Esiste un metodo per cambiare il tipo e la dimensione del font

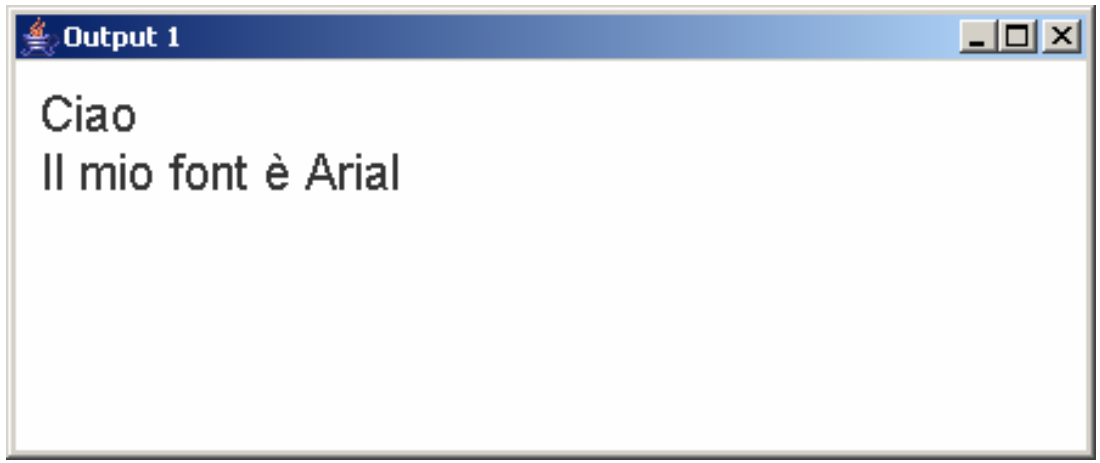
```
void setFont (String f, int dim)
```

Esempio di uso dei metodi

Il seguente programma crea 2 finestre di output con font differenti.

```
class Avvio{
  /* 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 ("Il mio font è Arial");
        out2.writeln ("Ciao");
        out2.write ("Il mio font è Script MT Bold");
  }
}
```

Il risultato del programma



La API di Java

Java fornisce un ampio insieme di classi ed oggetti già definiti e pronti per essere usati; tale insieme si chiama API (Application Programming Interface)

- per usare l'API di Java non serve sapere come sono definite le sue classi, ma solo quali servizi offrono
- ad esempio *String* è una classe definita nella API di Java – in una prossima lezione del corso impareremo a conoscerla nel dettaglio

Java e packages

La API di java è suddivisa in pacchetti, chiamati packages

- ogni package raggruppa un insieme di classi e oggetti con funzionalità affini

- se in un file Java si vuol usare una classe X contenuta in un package Y, occorre scrivere all'inizio del file (prima della definizione della classe) la seguente frase

import Y.X;

I packages di Java

Ogni package di Java ha un nome che inizia per java o javax:

- *java.lang* (classi di frequente utilizzo)
- *java.io* (classi per input/output)
- *java.net* (classi per la programm. di rete)
- *javax.swing* (classi per le GUI)
- ….

Per poter usare tutte le classi di un package (es. java.io) si può scrivere *import java.io.*;*

Il package java.lang

Nel presente corso si useranno per lo più classi contenute nel package *java.lang* (es. la classe *String*)

Per usare le classi di *java.lang* non serve importare il package esplicitamente - esso è sempre importato automaticamente

Input/Output in Java

Il package *java.io* offre un'ampia gamma di classi per gestire l'input e l'output

- l'uso di tali classi richiede però nozioni che non possono essere fornite in un corso di base

- per visualizzare messaggi useremo la classe *OutputWindow* vista in precedenza

- per leggere dati immessi da tastiera useremo la classe *InputWindow* che vedremo nella prossima parte di questa lezione

L'oggetto System.out di Java

In realtà, per visualizzare messaggi su una finestra non grafica, l'API di Java offre un oggetto predefinito di semplice uso, l'oggetto *System.out*

- *System.out* non è una classe, ma è un oggetto di una classe (*PrintStream*) che si trova nel package *java.io;* l'oggetto *System.out* è però creato nel package *java.lang*

- per visualizzare un messaggio con *System.out*, si può scrivere: *System.out.print ("……");*

Un esempio di uso di System.out

Ecco come si potrebbe riscrivere il programma che visualizza il messaggio “il mio primo programma”, usando *System.out*

```
class Avvio{
  /* visualizza “il mio primo programma” */
  public static void main (String[] args){
      System.out.println (“il mio primo
programma”);
  }
}
```

Glossario dei termini principali

Termine	Significato
Diagramma di collaborazione	Diagramma UML che rappresenta lo schema di interazione tra le classi e gli oggetto di un programma
class	Parola chiave del linguaggio Java, usata per iniziare la definizione di una classe
Corpo della classe	Insieme di tutto il codice che definisce i campi e i metodi di una classe
Corpo di un metodo	Insieme delle istruzioni che compongono un metodo
main	Un metodo speciale dal quale inizia l'esecuzione di un programma
Costruttore	Metodo speciale che serve per creare oggetti di una classe
new	Operatore Java usato insieme ai costruttori per creare oggetti
Variabile riferimento	Contenitore che può memorizzare il riferimento ad un oggetto
=	Operatore di assegnamento (permette di dare un valore ad una variabile)
Terminatore di istruzione ';'	Simbolo di terminazione di una istruzione
Letterale stringa	Sequenza di caratteri tra due apici doppi

Glossario dei termini principali

Termine	Significato
Diagramma di sequenza	Diagramma UML che schematizza il controllo del programma da parte di oggetti e metodi nel tempo
Java Virtual Machine	Macchina virtuale ideata dagli sviluppatori del linguaggio Java
Bytecode	Sequenza di istruzioni eseguibili dalla Java Virtual Machine
Compilatore Java	Software che traduce codice Java in bytecode
Interprete Java	Software che traduce istruzioni di bytecode in istruzioni per uno specifico processore e che fa eseguire al processore le istruzioni tradotte
Errori sintattici	Errori “grammaticali” nella scrittura di un programma; un compilatore rileva la presenza di tali errori se ce ne sono
API (Application Programming Interface) di Java	Libreria di funzionalità predefinite (classi, oggetti e metodi) a supporto della programmazione con Java
Package	Gruppo di classi con funzionalità affini nello sviluppo in Java; la API di Java è organizzata in packages
import	Parola chiave del linguaggio Java usata per importare una classe o tutte le classi di un package
System.out	Oggetto Java definito nella API; rappresenta lo standard output di un programma