

SUPSI

# Introduzione alle Classi

Loris Grossi, Fabio Landoni, Andrea Baldassari

Contenuto realizzato in collaborazione con: T. Leidi, A.E. Rizzoli, S. Pedrazzini

Fondamenti di Informatica

Bachelor in Ingegneria Informatica

# Obiettivo

Essere in grado di utilizzare la classe `Scanner` e la classe `String` messe a disposizione da Java.

Obiettivi della lezione:

- Capire il concetto della classe quale contenitore di sottoprogrammi e di costanti.
- Comprendere la differenza tra procedura e funzione.
- Capire il concetto della classe quale tipo di dato.
- Conoscere la classe `Scanner` e i metodi per la lettura di informazioni da tastiera.
- Conoscere la classe `String` e le funzionalità che mette a disposizione (concatenazione, confronto, conversione da tipo primitivo a stringa e viceversa, e tanti altri).
- Conoscere l'utilizzo dei blocchi di testo.

## Classi come contenitori di sottoprogrammi

Per ora, le **classi** sono state introdotte esclusivamente come contenitori di programmi, cioè delle **procedure main ()**.

In Java, anche il programma più semplice va dichiarato all'interno di una **classe**.

Però, le classi di Java vengono utilizzate anche come contenitori di **sottoprogrammi** (detti anche procedure o funzioni).

Questi sottoprogrammi sono **istruzioni** che si possono utilizzare all'interno dei programmi in sviluppo.

Un sottoprogramma è un insieme di istruzioni che vengono isolate per:

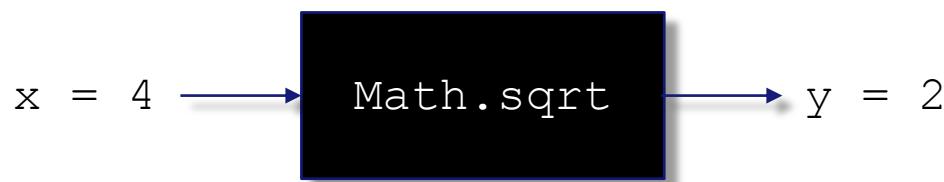
- facilitarne il **riutilizzo**;
- rendere il programma più **leggibile**;
- rendere il codice **facilmente manutenibile**.

# Classi come contenitori di sottoprogrammi

Esempi di classi che contengono sottoprogrammi, sono:

- *System*
- *Math*
- *String*
- *Scanner*

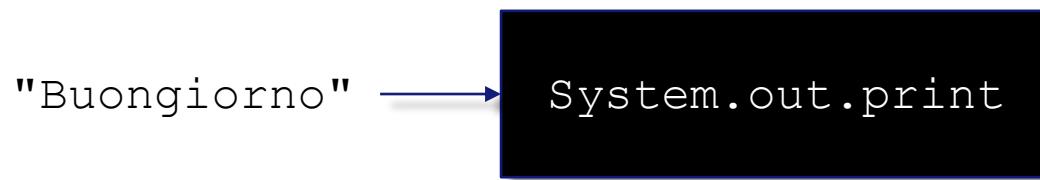
Le procedure e funzioni contenute sono da utilizzare come scatole nere:



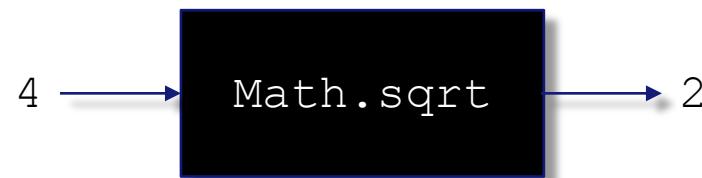
# Sottoprogrammi: procedure e funzioni

I sottoprogrammi si dividono in:

- **Procedura**: sottoprogramma che effettua delle operazioni utilizzando gli eventuali argomenti d'ingresso ma **non restituisce un valore** in uscita.



- **Funzione**: sottoprogramma che effettua delle operazioni utilizzando gli eventuali argomenti d'ingresso e **restituisce un valore** in uscita.



## Esempi della classe System

```
System.out.print("Hello World");
```

visualizza la stringa "Hello World" sullo standard out (monitor).

```
System.exit(1);
```

causa la terminazione del programma con codice di uscita 1.

## Classi come contenitori di costanti

In maniera simile ai sottoprogrammi, le classi di Java possono anche contenere delle **costanti**.

Potete utilizzare queste costanti liberamente nel vostro codice. Ad esempio:

```
public class Sfera {  
    public static void main(String[] args) {  
        double raggio = 1.5;  
        double superficie = 4 * Math.PI * raggio * raggio;  
        System.out.println("Superficie: " + superficie);  
    }  
}
```

## Classi come tipi di dato

Un altro ruolo delle classi è quello di permettere la definizione di **tipi di dato strutturati detti oggetti**.

Ad esempio, abbiamo visto che String è una classe che si usa come tipo di dato per dichiarare delle variabili:

```
String hello = "Hello, World!";
```

Agli oggetti possono essere associati **ulteriori sottoprogrammi**, come ad esempio le funzioni equals() e length() di String.

## Classi come tipi di dato

Anche gli oggetti vanno utilizzati come scatole nere:

```
input.nextInt()
```

 42

Prima dell'utilizzo è necessario creare gli oggetti. Di solito, lo si fa utilizzando l'operatore **new**:

```
Scanner input = new Scanner(System.in);  
int valore = input.nextInt();
```

# Tipi riferimento

Gli oggetti fanno parte dei tipi riferimento.

Arrays:

```
int[] ilMioArray;  
ilMioArray = new int[10];
```

Classi:

```
Scanner input;  
input = new Scanner(System.in);
```

In Java non esiste l'equivalente della struct (come in C) o del record (come in Ada). Sono stati sostituiti dalla classe.

I tipi riferimento verranno approfonditi nel corso del semestre.

## Nomi composti e forme qualificate

Gli **identificatori** dei sottoprogrammi e delle costanti possono fare riferimento a strutture organizzate gerarchicamente.

Viene chiamata **forma qualificata**. È necessario specificare il percorso completo per accedere all'identificatore.

Esempio:

```
System.out.println("Hello, World!");
```

la procedura: **println**

è applicata all'oggetto: **out**

della classe: **System**

Si usa il **punto** ‘.’ come separatore.

## Import della classe

Per usare una classe delle librerie predefinite di Java, la dobbiamo importare utilizzando ***L'istruzione import.***

```
import nome.qualificato.Classe;
```

Ad esempio, se si vuole acquisire informazione dalla tastiera utilizzando la classe Scanner, è necessario importare la classe utilizzando l'istruzione:

```
import java.util.Scanner;
```

A questa regola fanno eccezione alcune classi, quali ad esempio le ***classi System e Math***, che vengono importate automaticamente. Queste classi sono contenute nel package `java.lang`.

## Procedure e funzioni per l'input e l'output

Per realizzare programmi utili, è essenziale poter **leggere dati in ingresso e comunicare i risultati in uscita**.

Java è stato ideato per usare soprattutto le interfacce grafiche.

Però, nel corso di questo semestre, useremo Java in maniera semplice. Ogni interazione con il programma verrà fatta **tramite la console**.

Le interfacce grafiche verranno trattate successivamente.

## Lettura da tastiera

La classe Scanner mette a disposizione una serie di metodi utili a **leggere diversi tipi di dato**:

- Per le stringhe: next() e nextLine().
- Per gli interi: nextByte(), nextShort(), nextInt() e nextLong().
- Per i valori a virgola mobile: nextFloat() e nextDouble().
- Per i booleani: nextBoolean().

## La classe Scanner: esempio

```
import java.util.Scanner;

public class CiaoAmico {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Lettura del nome
        System.out.print("Ciao! Come ti chiami? ");
        String nome = input.nextLine();
        System.out.println("Ciao " + nome);

        // Lettura dell'età
        System.out.print("E quanti anni hai? ");
        int eta = input.nextInt();

        System.out.println("Ah, hai già " + eta + " anni!");
        input.close();
    }
}
```

## Lettura dei diversi tipi di dato

La classe Scanner mette a disposizione una serie di metodi utili a **scoprire se un dato in input è di un determinato tipo**:

- Per le stringhe: `hasNext()` e `hasNextLine()`.
- Per gli interi: `hasNextByte()`, `hasNextShort()`, `hasnextInt()` e `hasNextLong()`.
- Per i valori a virgola mobile: `hasNextFloat()` e `hasNextDouble()`.
- Per i booleani: `hasNextBoolean()`.

## La classe Scanner: esempio

```
import java.util.Scanner;

public class UtilizzoDiScanner {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int intero = 0;
        double virgola = 0.0;
        String stringa = "";

        if (scanner.hasNextInt()) {
            intero = scanner.nextInt();
        } else if (scanner.hasNextDouble()) {
            virgola = scanner.nextDouble();
        } else {
            stringa = scanner.nextLine();
        }
        // ...

        scanner.close();
    }
}
```

## La classe Scanner: esempio

```
import java.util.Scanner;

public class TestInt {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Inserire un intero: ");
        // Attendi l'immissione del valore da parte dell'utente e
        // verifica se può essere convertito in valore di tipo int
        while (!scanner.hasNextInt()) {
            System.out.print("Inserire un intero!! Riprova: ");
            // Ignora il valore inserito dall'utente visto che non
            // è del tipo desiderato (int)
            scanner.nextLine();
        }
        // Converti il valore inserito dall'utente in un valore
        // intero e assegna alla variabile numero.
        int numero = scanner.nextInt();
        System.out.println("Numero: " + numero);

        scanner.close();
    }
}
```

## Anomalia dell'input

**Attenzione:** purtroppo, la classe Scanner ha un comportamento che può risultare **anomalo quando si legge da tastiera.**

Se si fa un `nextInt()` oppure un qualsiasi altro “next numerico”, il carattere di ‘a capo’ rimane nel buffer di entrata (non viene letto).

Questa situazione non è problematica se successivamente si fa un altro `nextInt()` oppure un altro “next numerico”, perché il carattere di ‘a capo’ viene ignorato. Ma la situazione diventa problematica se si fa un `nextLine()`.

**Il `nextLine()` legge il carattere di ‘a capo’ che è rimasto nel buffer come se fosse una nuova linea di testo. Di conseguenza, la lettura sembra andare a vuoto.**

## Anomalia dell'input

```
import java.util.Scanner;

public class TestScanner {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Inserire un numero: ");
        int numero = scanner.nextInt();

        System.out.print("Inserire del testo: ");
        String testo = scanner.nextLine();

        System.out.println("Numero: " + numero);
        System.out.println("Testo: " + testo);

        scanner.close();
    }
}
```

### Output:

```
Inserire un numero: 666
Inserire del testo: Numero: 666
Testo:
```

## Anomalia dell'input

Per ovviare al problema, se il codice presenta una sequenza del tipo:

- Uno o più `nextInt()` (o un qualsiasi altro “next numerico”),
- seguito da uno o più `nextLine()` ...

... ***basta fare un `nextLine()` supplementare dopo l'ultimo `nextInt()`*** (e prima del primo `nextLine()`) per assicurarsi di catturare il carattere di ‘a capo’ rimasto nel buffer.

Riassumendo:

```
int numero = scanner.nextInt();
// o qualsiasi altro 'next numerico'

scanner.nextLine(); // svuota il buffer

String testo = scanner.nextLine();
```

## Anomalia dell'input

```
import java.util.Scanner;

public class TestScanner {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Inserire un numero: ");
        int numero = scanner.nextInt();
scanner.nextLine(); // Svuota il buffer

        System.out.print("Inserire del testo: ");
        String testo = scanner.nextLine();

        System.out.println("Numero: " + numero);
        System.out.println("Testo: " + testo);

        scanner.close();
    }
}
```

### Output:

```
Inserire un numero: 666
Inserire del testo: Java
Numero: 666
Testo: Java
```

# Ricapitolando ...

## Le istruzioni

```
System.out.print(testo);
```

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

permettono di visualizzare a schermo praticamente qualsiasi tipo di dato.

Per quanto riguarda l'input da tastiera, l'oggetto `Scanner` mette a disposizione comandi come:

```
next(), nextLine()  
nextByte(), nextShort()  
nextInt(), nextLong()  
nextFloat(), nextDouble()  
nextBoolean()
```

```
hasNext(), hasNextLine()  
hasNextByte(), hasNextShort()  
hasNextInt(), hasNextLong()  
hasNextFloat(), hasNextDouble()  
hasNextBoolean()
```

# Stringhe

In Java, tutte le stringhe sono oggetti della classe `java.lang.String`.

Però, Java mette a disposizione un supporto speciale per le stringhe di caratteri **che le fa somigliare a tipi di dato primitivi**.

Come per tutti gli oggetti, è possibile creare le stringhe utilizzando l'operatore `new` (che discuteremo fra qualche lezione), ma solitamente non lo si fa.

Tipicamente, le stringhe vengono create utilizzando **letterali o tramite funzioni**.

# Stringhe

Quando si racchiude una stringa ***fra virgolette (letterale)***, viene automaticamente creato un oggetto di tipo stringa. Inoltre molte funzionalità di Java per l'input e l'output (come ad esempio lo Scanner), creano automaticamente le stringhe prima di restituirle.

```
String s1 = "Questa è una stringa";
```

```
Scanner scanner = new Scanner(System.in);
String s2 = scanner.nextLine();

// ...

scanner.close();
```

## Letterali per le stringhe

I letterali di tipo **String** vanno sempre messi tra **virgolette** (esempio: "Ciao").

All'interno dei letterali si possono usare le **escape sequences** (vedi slides sui letterali di tipo `char`).

Ad esempio:

```
System.out.print("Chiesi:\n\t\"Mi state ascoltando?\"\n");
```

Mostrerà a schermo:

Chiesi:  
"Mi state ascoltando?"

## Confronti fra stringhe

Importante: non è corretto confrontare i valori delle stringhe utilizzando l'operatore `==`. Per **confrontare i valori delle stringhe** bisogna utilizzare la funzione **`equals()`**.

```
String str1 = "Ciao";  
  
Scanner input = new Scanner(System.in);  
String str2 = input.nextLine();  
input.close();  
  
boolean uguali = str1.equals(str2);  
System.out.println(uguali);
```

## Concatenazione di stringhe

Normalmente due o più stringhe vengono concatenate utilizzando **l'operatore +**.

```
Scanner input = new Scanner(System.in);
String nome = input.nextLine();
input.close();

System.out.println("Ciao , " + nome + " !");
```

Se si concatenano stringhe con **valori numerici**, questi vengono **automaticamente convertiti in stringhe**.

```
int i = 5;
String str1 = "Il numero è: " + i;
```

## Blocco di testo

È possibile spezzare una stringa su più righe usando la sintassi seguente:

```
String testo = "{\n" +  
    " \"nome\": \"Mario Rossi\", \n" +  
    " \"eta\": 45, \n" +  
    " \"indirizzo\": \"via dei Meli 42, Java Town\"\n" +  
    " }";  
System.out.println(testo);
```

**Output:**

```
{  
    "nome": "Mario Rossi",  
    "eta": 45,  
    "indirizzo": "via dei Meli 42, Java Town"  
}
```

## Blocco di testo

A partire da Java 17, è possibile utilizzare dei **blocchi di testo, definiti con tre virgolette**. Le escape sequences '\n' e '\"' non sono più necessarie.

```
String testo = """
{
    "nome": "Mario Rossi",
    "eta": 45,
    "indirizzo": "via dei Meli 42, Java Town"
}
""";
System.out.println(testo);
```

L'output è identico a quello mostrato nella slide precedente.

## Blocco di testo

**Attenzione:** la posizione delle tre virgolette di chiusura identificano il margine sinistro del blocco di testo.

```
String blocco = """  
    Blocco di testo  
    che si estende  
    su più righe  
""";
```

```
System.out.println("123456");  
System.out.println(blocco);
```



```
String blocco = """  
    Blocco di testo  
    che si estende  
    su più righe  
""";
```

```
System.out.println("123456");  
System.out.println(blocco);
```

**Output:**

```
123456  
Blocco di testo  
che si estende  
su più righe
```

**Output:**



```
123456  
Blocco di testo  
che si estende  
su più righe
```

## Blocco di testo

Se le tre virgolette di chiusura sono spostate verso destra, il primo carattere (non spazio) nel blocco di testo identifica il margine sinistro.

```
String blocco = """  
    Blocco di testo  
        che si estende  
        su più righe  
    """;  
  
System.out.println("123456");  
System.out.println(blocco);
```

**Output:**

```
123456  
Blocco di testo  
    che si estende  
    su più righe
```

## Conversioni di tipi primitivi in stringhe

Qualsiasi tipo di Java può essere **convertito nel tipo String** e viceversa. Per questo tipo di conversioni esistono funzioni specifiche.

Ad esempio, si potrebbe volere convertire:

- la stringa "10" nel valore numerico intero 10, oppure
- la stringa "17.42e-2" nel valore double 0.1742.

Per convertire una stringa in un valore di tipo int si utilizza la funzione:

```
String str = "10";
int num = Integer.parseInt(str);
```

Per convertire una stringa in un double si utilizza la funzione:

```
String str = "17.42e-2";
double num = Double.parseDouble(str);
```

## Conversioni di stringhe, esempi

Conversione di un valore numerico in una stringa:

```
int i = 42;
String s1 = "" + i;
String s2 = String.valueOf(i);
String s3 = Integer.toString(i);
```

Conversione di una stringa in un valore numerico:

```
int i1 = Integer.parseInt("17030075");
int i2 = Integer.valueOf("17030075");
```

## Operazioni sulle stringhe

Gli oggetti di tipo stringa mettono a disposizione diverse **funzioni** che permettono di interrogarne le proprietà e di eseguire operazioni sulle stringhe:

- `s1.length()` per conoscere la lunghezza della stringa `s1`,
- `s1.charAt(i)` per ottenere il carattere alla posizione `i`,
- `s1.substring(i, j)` per ottenere una nuova stringa formata dai caratteri presenti a partire dalla posizione `i` (compresa) fino alla posizione `j` (esclusa),
- `s1.indexOf(ch)` e `s1.indexOf(s2)` per ottenere l'indice (posizione) del carattere o della stringa ricercata all'interno di `s1`,
- `s1.toLowerCase()` per ottenere una copia di `s1` tutta in minuscolo,
- `s1.toUpperCase()` per ottenere una copia di `s1` tutta in maiuscolo,
- `s1.equalsIgnoreCase(s2)` per confrontare le stringhe `s1` e `s2` ignorando la differenza tra maiuscole e minuscole,
- `s1.trim()` per ottenere una copia di `s1` con rimosso gli spazi all'inizio e alla fine della stringa.

## Stringhe, esempi

```
public class TestStringhe {  
    public static void main(String[] args) {  
        String testStr = "Ciao, Mondo!";  
        String newStr = "ciao, Mamma!";  
        int i = 0, j = 4;  
        System.out.println("La prima stringa è " + testStr);  
        System.out.println("La seconda stringa è " + newStr);  
        System.out.println("È " + testStr.equals(newStr)  
            + " che le due stringhe sono uguali");  
        System.out.print("Ignorando la differenza tra "  
            + "minuscole e maiuscole, allora ");  
        System.out.println(testStr.equalsIgnoreCase(newStr));  
        System.out.println("La prima stringa è lunga "  
            + testStr.length() + " caratteri");  
        System.out.println("Il primo carattere della prima stringa è "  
            + testStr.charAt(0));  
        System.out.println("La sottostringa compresa tra " + i + " e "  
            + j + " è " + testStr.substring(i, j));  
        System.out.println("La prima stringa in maiuscolo è: "  
            + testStr.toUpperCase());  
    }  
}
```

# Riepilogo

- Sottoprogrammi: procedure e funzioni
- Classi e sottoprogrammi
- Classi e costanti
- Classi come tipi di dato
- Import delle classi
- Procedure e funzioni per l'output
- Procedure e funzioni per l'input
- "Anomalia" dell'input
- La classe String
- Operazioni sulle stringhe