# Programmazione Orientata agli Oggetti in Linguaggio Java

## Classi e Oggetti: Metafora Parte a

versione 2.2



**G. Mecca – Università della Basilicata – mecca@unibas.it**



## Sommario

- Componenti
- Heap
- Classe
- Package
- Public e Private
- Invio di Messaggi ai Componenti
- Difetti della Soluzione Statica

# Componenti

- Componente
  - ⇨pezzo di software capace di svolgere compiti (responsabilità)
- Applicazione a oggetti
  - ⇨collezione di componenti
  - ⇨i componenti collaborano per raggiungere l'obiettivo applicativo
  - ⇨la collaborazione avviene attraverso lo scambio di messaggi





# Componenti

- Due tipi di componenti
  - ⇨classi
  - ⇨oggetti
- Ogni componente ha
  - ⇨un identificatore >> nome
  - ⇨delle proprietà
  - ⇨e dei metodi

# Componenti

- Proprietà
  - ⇨dati – costanti e variabili – conosciuti dal componente
  - ⇨rappresentano lo stato del componente
- Metodi
  - ⇨funzioni e procedure
  - ⇨rappresentano le operazioni eseguibili dal componente





# Componenti

- Metafora
  - ⇨ogni componente conosce delle informazioni
  - ⇨ed è in grado di svolgere dei compiti, cioè eseguire delle operazioni
  - ⇨i componenti si scambiano messaggi che corrispondono a richieste di servizio
  - ⇨in risposta ad una richiesta di servizio, i componenti eseguono le operazioni e rispondono con i risultati

# Heap

- Memoria per i componenti
  - ⇨all'interno dello "heap"
- Nella programmazione procedurale
  - ⇨tre zone di memoria per un programma
  - ⇨istruzioni (zona del codice)
  - ⇨pila di attivazione (zona dei dati) – dati "statici"
  - ⇨heap – dati allocati dinamicamente (ovvero manipolati attraverso i puntatori)





# Heap

- Utilizzo della memoria in Java
  - ⇨zona del codice: contiene il bytecode
  - ⇨pila di attivazione (stack): contiene i record di attivazione dei metodi (variabili locali e parametri)
- Lo heap
  - ⇨non viene utilizzato per i puntatori (che non esistono)
  - ⇨è riservato ai componenti

# Heap

- Heap (“Mucchio”)
  - ⇨zona di memoria in cui viene allocata la memoria per gli oggetti
  - ⇨zona separata rispetto allo stack
- Metafora
  - ⇨nello heap “vivono” i componenti
  - ⇨lo heap è la città dei robot





# Classi

- Un esempio: la calcolatrice statica
  - ⇨applicazione che consente di effettuare operazioni aritmetiche (+, -, *, /)
  - ⇨il risultato dell’ultima operazione resta in memoria e può essere utilizzato per effettuare operazioni con memoria
  - ⇨l’interfaccia è basata sulla console: i dati vengono letti dalla tastiera e visualizzati in forma testuale

# Classi

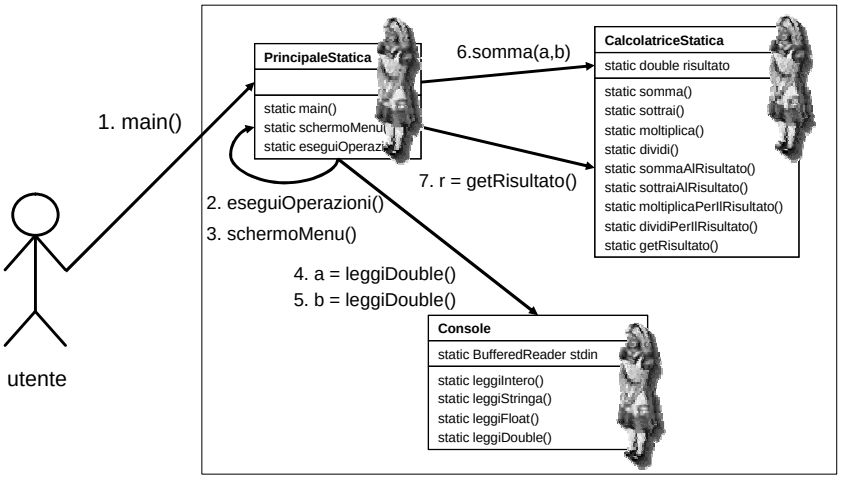
- I versione: tre componenti di tipo Classe
  - ⇨ogni componente ha precise responsabilità
- CalcolatriceStatica
  - ⇨effettua i calcoli
- PrincipaleStatica
  - ⇨gestisce l'interazione con l'utente e il controllo
- Console
  - ⇨legge i dati dalla tastiera

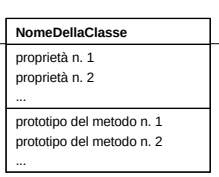
>> calcolatricestatica





# Classi






# Classi

- Rappresentazione grafica di una classe
  - ⇨convenzione di UML (“Uniform Modeling Language”)
- Nota
  - ⇨a volte utilizzeremo delle semplificazioni
- Esempi
  - ⇨spesso ometteremo il valore di ritorno dei metodi
  - ⇨spesso ometteremo l’elenco dei parametri
  - ⇨spesso ometteremo del tutto proprietà e metodi





# Classi

- Sintatticamente
  - ⇨la parola chiave che contraddistingue le proprietà e i metodi di una classe è static
- Esempio

```
public class CalcolatriceStatica {
    private static double risultato;
    public static void somma (double a, double b) {
        CalcolatriceStatica.risultato = a + b;
    }
    ...
}
```

# Classi

array di stringhe
parametri per il main
forniti dalla riga di comando

- Un metodo statico particolare
  - ⇨il metodo main
  - ⇨serve ad avviare l'esecuzione dell'applic.
- Esempio

```
public class PrincipaleStatica {
   public static void main (String[] args) {
      PrincipaleStatica.eseguiOperazioni();
   }
}
```





# Package

- Organizzazione dei componenti
  - ⇨i componenti sono organizzate in "package"
- Package
  - ⇨gruppo di componenti affini
- Due funzioni principali
  - ⇨serve a completare il riferimento alle classi
  - ⇨stabilisce legami più stretti tra i componenti di un package

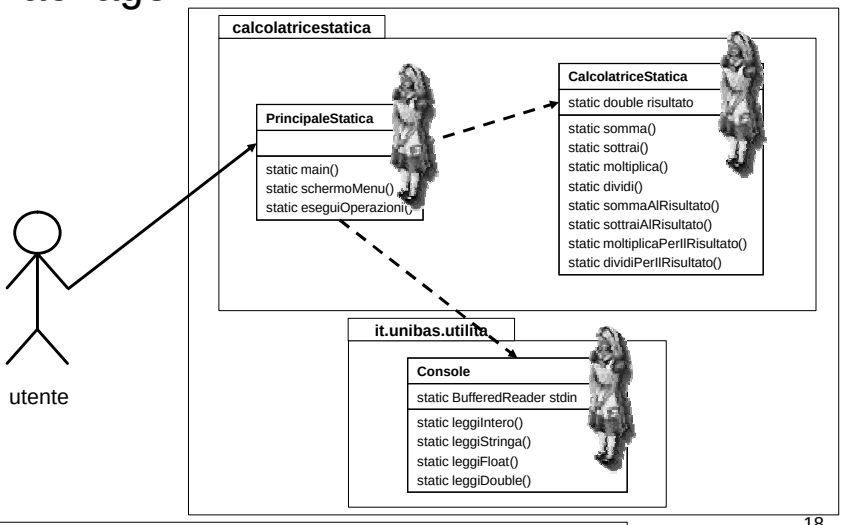
# Package

- Metafora
  - ⇨il package è un condominio
  - ⇨con un indirizzo
  - ⇨tutti i componenti appartenenti ad uno stesso package abitano nello stesso condominio
  - ⇨hanno lo stesso indirizzo
  - ⇨si conoscono bene tra di loro
- Per mandare messaggi ad una classe
  - ⇨bisogna specificare nome e indirizzo





# Package



# Package

- Nome di un package
  - ⇨può essere semplice (es: calcolatricestatica)
  - ⇨oppure composto da più parti separate da . (es: it.unibas.utilita)
- Nome completo di una classe
  - ⇨“nome completamente qualificato”
  - ⇨*nomePackage.nomeClasse*
  - ⇨es: calcolatricestatica.PrincipaleStatica, it.unibas.utilita.Console





# Package

- Vantaggi
  - ⇨in questo modo possono esistere due classi distinte chiamate allo stesso modo
  - ⇨due classi Calcolatrice in package diversi
  - ⇨di conseguenza, per distinguerle è necessario specificare oltre al nome della classe anche il nome del package a cui ciascuna appartiene

# Package

- Nota
  - ⇨le scelte sui nomi in Java si ripercuotono sulla struttura fisica dei file
- Regole sulle classi
  - ⇨una classe chiamata CalcolatriceStatica deve essere contenuta in un file CalcolatriceStatica.java
  - ⇨attenzione a minuscole e maiuscole





# Package

- Regole sui package
  - ⇨i file .java delle classi di del package calcolatricestatica devono essere contenuti in una cartella del disco chiamata calcolatricestatica
  - ⇨attenzione a maiuscole e minuscole
- Attenzione
  - ⇨queste regole si ripercuotono sul processo di compilazione ed esecuzione (>>)

# Public e Private

- Visibilità
  - ⇨non tutte le proprietà e tutti i metodi di un componente sono visibili all'esterno
  - ⇨il linguaggio consente di utilizzare modificatori di visibilità
- Modificatori di Visibilità
  - ⇨stabiliscono a che livello è visibile una risorsa (es: proprietà, metodo o l'intero componente)
  - ⇨principali valori: public e private





# Public e Private

- public
  - ⇨tutti gli altri componenti possono utilizzare la risorsa (inviare messaggi)
- private
  - ⇨la risorsa può essere utilizzata solo all'interno del codice del componente stesso
  - ⇨è inaccessibile all'esterno

# Invio di Messaggi ai Componenti

- Per inviare un messaggio ad un componente
  - ⇨bisogna conoscerne il nome
- Nel caso delle classi
  - ⇨in generale bisogna specificare il nome completamente qualificato della classe
  - ⇨tra classi dello stesso package è possibile omettere il nome del package





# Invio di Messaggi ai Componenti

- Esempi:
  - ⇨in PrincipaleStatica.java
  - ⇨invio di un messaggio a Console:
    es: scelta = it.unibas.utilita.Console.leggiIntero()
  - ⇨invio di un messaggio a CalcolatriceStatica
    es: risultato = CalcolatriceStatica.somma(a, b);

# Invio di Messaggi ai Componenti

- Tipologie di messaggi
  - ne esistono due
  - tipicamente i messaggi vengono inviati per richiedere l'esecuzione di un metodo
  - ma è possibile anche inviare messaggi per richiedere l'accesso ad una proprietà





# Invio di Messaggi ai Componenti

- I forma: esecuzione di un metodo
  - bisogna specificare nome della classe, nome del metodo, argomenti
  - *NomeClasse.nomeMetodo(argomenti);*
  - es: scelta = it.unibas.utilita.Console.leggiIntero()
  - es: risultato = CalcolatriceStatica.somma(a, b);
- Un metodo molto utilizzato
  - System.out.println(<stringa>);
  - stampa sullo standard output

# Invio di Messaggi ai Componenti

- Esecuzione del metodo metodo
  - ⇨semantica usuale dell'esecuzione di sottoprogramma
  - ⇨interruzione del metodo chiamante
  - ⇨passaggio dei parametri (>>)
  - ⇨esecuzione del codice del metodo chiamato
  - ⇨ritorno ed eventuale restituzione del risultato





# Invio di Messaggi ai Componenti

- II forma: accesso alle proprietà
  - ⇨un componente chiede ad un altro di utilizzare la proprietà specificata (es: modificare o conoscere il valore)
  - ⇨es: in PrincipaleStatica

```
CalcolatriceStatica.risultato++;
CalcolatriceStatica.risultato = 0;
double risultato = CalcolatriceStatica.risultato;
```

# Invio di Messaggi ai Componenti

- Attenzione
  - ⇨la possibilità di inviare questo messaggio dipende dal modificatore di visibilità
  - ⇨tipicamente le proprietà sono dichiarate private
  - ⇨es: in CalcolatriceStatica
    private static double risultato;
  - ⇨se le proprietà sono private, è possibile esclusivamente inviare messaggi che chiedono l'esecuzione di metodi





```
package calcolatricestatica;
public class CalcolatriceStatica {

    private static double risultato;

    public static double getRisultato() {
        return CalcolatriceStatica.risultato;
    }

    public static void somma(double a, double b) {
        CalcolatriceStatica.risultato = a + b;
    }

    public static void sottrai(double a, double b) {
        CalcolatriceStatica.risultato = a - b;
    }

    public static void moltiplica(double a, double b) {
        CalcolatriceStatica.risultato = a * b;
    }
    ...
```

```
package calcolatricestatica;
public class PrincipaleStatica {

    public static void main(String args[]) {
        PrincipaleStatica.eseguiOperazioni();
    }

    public static void eseguiOperazioni() {
        boolean continua = true;
        while (continua) {
            double a, b, risultato = 0;
            int scelta = PrincipaleStatica.schermoMenu();
            if (scelta == 0) {
                continua = false;
            } else {
                System.out.print("   Inserisci gli argomenti:   --> ");
                a = it.unibas.utilita.Console.leggiDouble();
                b = it.unibas.utilita.Console.leggiDouble();
                if (scelta == 1) {
                    CalcolatriceStatica.somma(a, b);
                    risultato = CalcolatriceStatica.getRisultato();
                }
            ...
```





```
    ...
    private static int schermoMenu() {
        int scelta;
        System.out.println("--------------------------------");
        System.out.println("        Calcolatrice            ");
        System.out.println("--------------------------------");
        System.out.println();
        System.out.println("   1. Esegui somma");
        System.out.println("   2. Esegui differenza");
        System.out.println("   3. Esegui moltiplicazione");
        System.out.println("   4. Esegui divisione");
        System.out.println();
        System.out.println("   0. Esci");
        System.out.println();
        System.out.print("   Scelta ----> ");
        scelta = it.unibas.utilita.Console.leggiIntero();
        while ((scelta < 0) || (scelta > 4)) {
            System.out.println("   Errore. Ripeti ----> ");
            scelta = it.unibas.utilita.Console.leggiIntero();
        }
        return scelta;
    }
}
```

# Difetti della Soluzione Statica

- Pregi dell'applicazione vista
  - ⇨buona organizzazione: le responsabilità dei componenti sono ben individuate
  - ⇨semplicità: viene utilizzato il minimo numero di componenti
- Difetti dell'applicazione
  - ⇨difetto n.1: utilizzo di componenti "globali"
  - ⇨difetto n.2: è possibile avere un'unica calcolatrice





# Difetti della Soluzione Statica

- Difetto n.1
  - ⇨le classi sono componenti con visibilità globale
  - ⇨qualsiasi componente può chiedere ad una classe l'esecuzione di un metodo
  - ⇨questo aumenta decisamente l'accoppiamento tra i componenti
  - ⇨es: da Console potrei cambiare lo stato della calcolatrice chiamando il metodo somma

# Difetti della Soluzione Statica

- In generale
  - ⇨è preferibile limitare i componenti a visibilità globale
  - ⇨valgono le stesse regole di base della programmazione procedurale
- Una soluzione migliore
  - ⇨basata su componenti di tipo “oggetto”





# Riassumendo

- Componenti
- Heap
- Classe
- Package
- Public e Private
- Invio di Messaggi ai Componenti
- Difetti della Soluzione Statica

# Termini della Licenza