Java: Definire Classi e Creare Oggetti

Damiano Macedonio

Dipartimento di Informatica, Università degli Studi di Verona

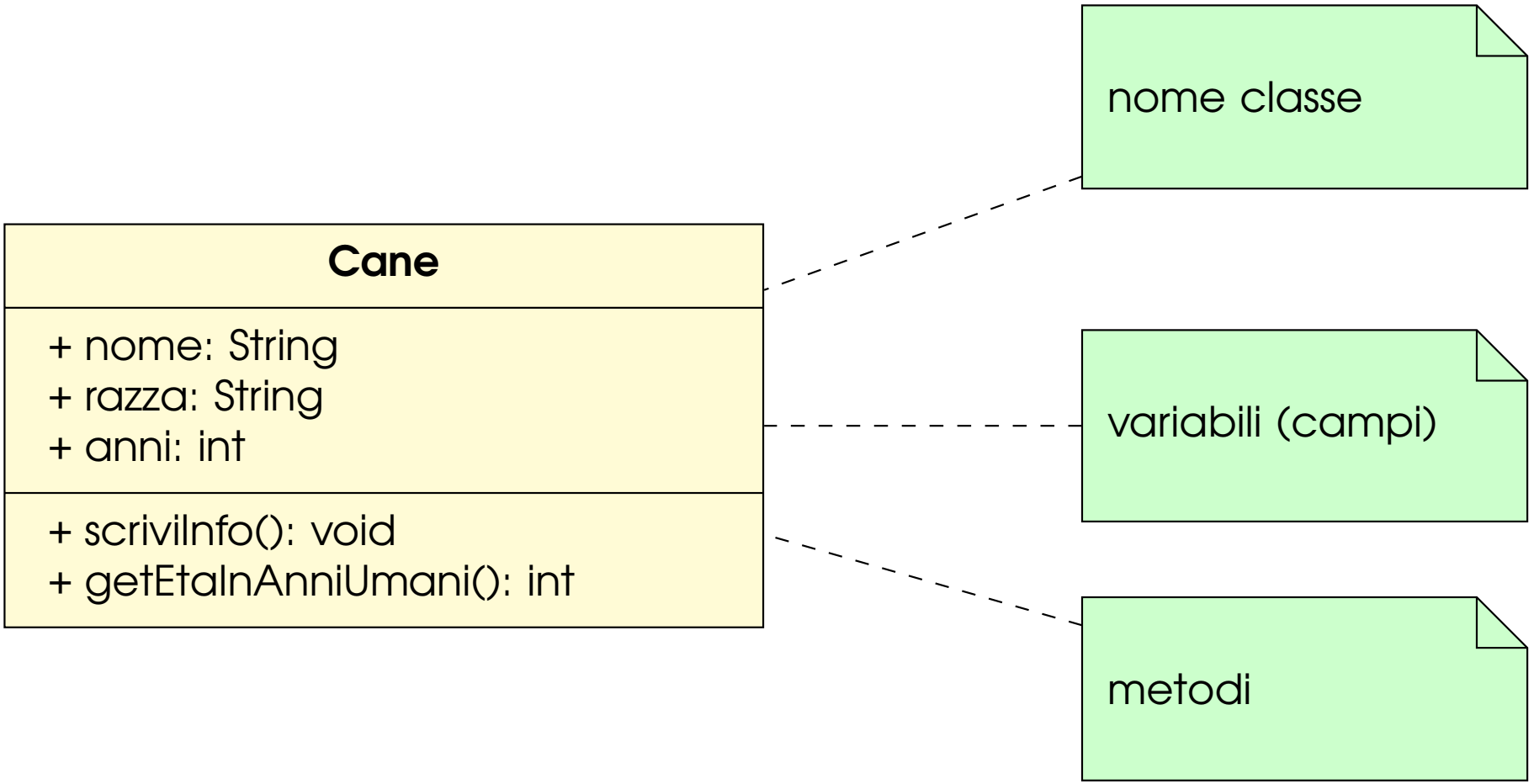
Corso di Programmazione per Bioformatica

lezione del 21 marzo 2014

Programmare con gli Oggetti

- Un programma Java è costituito da oggetti di vario tipo che interagiscono tra loro.
- Gli oggetti hanno associati dati e possono eseguire azioni.
  - I dati sono contenuti nelle variabili di istanza (o campi).
  - Le azioni vengono definite dai metodi di istanza.
- Gli oggetti vengono assegnati a variabili di tipo classe.
  - Una classe è la definizione di un tipo di oggetto.
  - Una classe specifica il nome e il tipo delle variabili di istanza degli oggetti, ma non specifica il loro valore.
  - Una classe specifica i metodi dei suoi oggetti.
- Un oggetto di una classe è una istanza della classe.
  - Il valore delle variabili di istanza è specifico delle singole istanze (ogni istanza possiede una propria copia).
  - Tutte le istanze di una determinata classe hanno gli stessi metodi.

Diagramma UML (Universal Modelling Language)



Implementazione

```java
// Cane.java
public class Cane {
  // variabili di istanza (non e' una buona idea definirle "public")
  public String nome;
  public String razza;
  public int anni;

  // metodi di istanza
  public void scriviInfo() {
    System.out.println("Nome: " + nome);
    System.out.println("Razza: " + razza);
    System.out.println("Eta': " + anni);
  }
  public int getEtaInAnniUmani() {
    if (anni <= 2)
      return anni * 11;
    else
      return 22 + ((anni - 2) * 5);
  }
}
```

Attenzione: Per semplificare la descrizione, stiamo violando alcuni importanti principi di progettazione!

Istanze della classe **Cane**

caneDiShaggy

nome: Scooby-Doo
razza: alano
anni: 9

caneDiTopolino

nome: Pluto
razza: bracco
anni: 7

caneRandagio

nome: Balto
razza: husky mezzo-lupo
anni: 8

```java
// TestCane.java
public class TestCane {
  public static void main(String[] args) {
    Cane caneDiShaggy; // crea variabile di tipo Cane
    caneDiShaggy = new Cane(); // assegna un nuovo oggetto alla varibile
    caneDiShaggy.nome = "Scooby-Doo";
    caneDiShaggy.razza = "alano";
    caneDiShaggy.anni = 9;

    caneDiShaggy.scriviInfo();

    Cane caneDiTopolino = new Cane(); // crea variabile assegnando un nuovo oggetto
    caneDiTopolino.nome = "Pluto";
    caneDiTopolino.razza = "bracco";
    caneDiTopolino.anni = 7;

    System.out.println(caneDiTopolino.nome + "e' un " + caneDiTopolino.razza +
        " di " + caneDiTopolino.getEtaInAnniUmani() + " anni umani.");
  }
}
```

```
$ java TestCane
Nome: Scooby-Doo
Razza: alano
Eta': 9
Pluto e' un bracco di 47 anni umani.
```

Creare istanze: la parola chiave `new`

Creare un oggetto di tipo `Cane`:

```
caneDiShaggy = new Cane();
```

1. Viene creato un oggetto della classe `Cane`
2. Vengono predisposte tutte le variabili di istanza all'interno dell'oggetto creato
3. Viene restituito l'indirizzo di memoria dell'oggetto creato
4. L'istruzione = assegna l'indirizzo dell'oggetto alla variabile `caneDiShaggy`

Valori di Default

- Quando viene istanziato un oggetto, le sue variabili di istanza sono automaticamente inizializzate a valori di default.
- I valori di default dipendono dal tipo della variabile.
  - int: 0
  - bool: `false`
  - String: `null`
- Ricordiamo che le variabili locali ai metodi rimangono indefinite finché non viene effettuato un assegnamento esplicito.

Invocazione

```
caneDiShaggy.scriviInfo();
```

- I metodi di istanza possono essere invocati solo su oggetti della classe in cui sono definiti.
- Un metodo di una classe **C** è accessibile a tutti gli oggetti della classe **C** creati.
- L'invocazione di un metodo comporta l'esecuzione delle operazioni in esso definite.
- Un metodo di istanza viene invocato su un oggetto e può manipolare lo stato (le variabili di istanza) dell'oggetto stesso.

Definizione

L'oggetto su cui viene invocato il metodo si dice receiver.

Corpo

- L'intestazione dei metodi di istanza non ha il modificatore `static`.

  ```
  public void scriviInfo() {
  ```

- Il corpo può contenere istruzioni che fanno riferimento a variabili di istanza

  ```
  System.out.println("Nome: " + nome);
  ```

- Le variabili di istanza a cui il metodo si riferisce sono quelle proprie del receiver

La Parola Chiave `this`

- Al di fuori della definizione della classe, il nome delle variabili di istanza è composto dal dal nome dell'oggetto della classe seguito dal punto e dal nome della variabile di istanza

```
caneDiShaggy.nome = "Scooby-Doo";
```

- All'interno della definizione di un metodo che risiede nella stessa classe della variabile di istanza, basta utilizzare il nome della variabile di istanza, senza il nome di alcun oggetto o punto.

```
public void scriviInfo() {
  System.out.println("Nome: " + nome);
```

La Parola Chiave `this`

- Quando si definiscono i metodi di istanza si sottintende l'oggetto, omettendone il nome. Il nome sottinteso è `this`.
- Il nome dell'oggetto si può anche includere:

```
public void scriviInfo() {
  System.out.println("Nome: " + this.nome);
  System.out.println("Razza: " + this.razza);
  System.out.println("Eta': " + this.anni);
}
```

- La parola chiave `this` rappresenta il receiver. Per esempio l'invocazione del metodo `caneDiShaggy.scriviInfo()` è equivalente a

```
1 System.out.println("Nome: " + caneDiShaggy.nome);
2 System.out.println("Razza: " + caneDiShaggy.razza);
3 System.out.println("Eta': " + caneDiShaggy.anni);
```

- Java permette di omettere la parola chiave `this`. Sarà necessario usarlo solo in alcuni casi.

Information Hiding

Quando usiamo un metodo definito da un altro programmatore, non ci interessa conoscerne i dettagli: ci interessa sapere cosa fa il metodo e non come lo fa.

I metodi dovrebbero essere unità complete, progettate senza tener conto dei dettagli implementativi degli altri metodi e dei programmi che ne fanno uso.

L'information hiding è una pratica che consente di progettare un metodo in modo che possa essere usato senza alcun bisogno di comprendere i dettagli del codice che lo implementa. Un po' come se il corpo del metodo fosse *nascosto*.

Encapsulation

L'incapsulamento è una forma di information hiding: la definizione di una classe deve essere tale per cui un programmatore possa usarla senza conoscerne i dettagli implementativi.

Separa la definizione di una classe in due parti.

Interfaccia

Indica ai programmatori ciò di cui hanno bisogno per usare la classe nei loro programmi.

Implementazione

Tutto ció che serve al programmatore che sta effettivamente scrivendo il codice della classe.

Commenti con Precondizioni e Postcondizioni

Sono commenti specifici per descrivere le finalità di un metodo.

Precondizione

Descrive le condizioni che devono sussistere prima che il metodo sia invocato. Se la precondizione non è soddisfatta, il metodo non dovrebbe essere invocato e comunque non ci si può aspettare che restituisca il risultato atteso.

Postcondizione

Descrive tutti gli effetti prodotti dall'invocazione del metodo. Essa indica cosa varrà dopo che il metodo è stato eseguito, sempre se la precondizione era valida prima dell'esecuzione del metodo stesso. Se il metodo restituisce un valore, la postcondizione include anche una descrizione del valore restituito.

Esempi

```
/**
 * Precondizione: Le variabili di istanza del receiver
 * devono avere dei valori
 * Postcondizione: Stampa a video i valori delle
 * variabili di istanza del receiver
 */
public void scriviInfo() {
....

/**
 * Precondizione: La variabile di istanza anni deve
 * avere un valore
 * @return il corrispettivo umano dell'eta' del cane
 */
public int getEtaInAnniUmani() {
```

Modificatori di accesso: `public` / `private`

`public`

Applicato a una classe, a un metodo o a una variabile di istanza, indica che qualsiasi altra classe li può usare direttamente.

`private`

- Applicato ad una variabile di istanza, il nome di tale variabile non è accessibile al di fuori della definizione della sua classe. La variabile può essere usata solo all'interno dei metodi definiti nella classe.
- Applicato ad un metodo, questo non può essere invocato al di fuori della definizione della classe. Tuttavia può essere invocato all'interno da un altro metodo appartenente alla stessa classe.

Una buona regola: rendere private le variabili di istanza

```
1  // Cane.java modificata
2  public class Cane {
3    // variabili di istanza
4    private String nome;
5    private String razza;
6    private int anni;
7
8    // metodi di istanza
9    public void scriviInfo() {
10     System.out.println("Nome: " + nome); // ha accesso
11     System.out.println("Razza: " + razza); // ha accesso
12     System.out.println("Eta': " + anni); // ha accesso
13   }
14   public int getEtaInAnniUmani() {
15     // ha accesso alla variabile di istanza anni
16     if (anni <= 2)
17       return anni * 11;
18     else
19       return 22 + ((anni -2) * 5);
20   }
21 }
```

```java
// TestCane.java
public class TestCane {
  public static void main(String[] args) {
    Cane caneDiShaggy; // crea variabile di tipo Cane
    caneDiShaggy = new Cane(); // assegna un nuovo oggetto alla variabile
    caneDiShaggy.nome = "Scooby-Doo"; // NON piu' valida!!
    caneDiShaggy.razza = "alano"; // NON piu' valida!!
    caneDiShaggy.anni = 9; // NON piu' valida!!

    caneDiShaggy.scriviInfo(); // VALIDA, ma poco informativa...

    Cane caneDiTopolino = new Cane(); // crea variabile assegnando un nuovo oggetto
    caneDiTopolino.nome = "Pluto"; // NON piu' valida!!
    caneDiTopolino.razza = "bracco"; // NON piu' valida!!
    caneDiTopolino.anni = 7; // NON piu' valida!!

    // e' valida solo l'invocazione a getEtaInAnniUmani,
    // ma anni non e' inizializzata
    System.out.println(caneDiTopolino.nome + "e' un " + caneDiTopolino.razza +
        " di " + caneDiTopolino.getEtaInAnniUmani() + " anni umani.");
  }
}
```

Come possiamo dare dei valori alle variabili di istanza? Con i metodi!

Metodi `get` / `set`

Rendere private tutte le variabili di istanza di una classe permette di avere controllo totale su di esse. Se necessario, bisogna però fornire dei metodi di accesso.

Metodo `get`

Permette di osservare quali sono i dati contenuti in una variabile di istanza.

Metodo `set`

Permette di modificare i dati memorizzati nelle variabili di istanza private. Puó verificare se un cambiamento è appropriato prima di apportare le modifiche richieste.

Ridefiniamo...

Un segno + significa pubblico. Un segno - significa privato.

Cane
- nome: String - razza: String - anni: int
+ setNome(nome: String): void + setRazza(razza: String): void + setAnni(anni: int): void + setCane(nome: String, razza: String, anni: int): void + getNome(): String + getRazza(): String + getAnni(): int + scriviInfo(): void + getEtaInAnniUmani(): int

```
1  // arricchiamo la classe Cane.java
2
3  public void setNome(String nome) {
4    this.nome = nome;
5  }
6  public void setRazza(String razza) {
7    this.razza = razza;
8  }
9  public void setAnni(int anni) {
10   if (anni >= 0)
11     this.anni = anni;
12 }
13 public void setCane(String nome, String razza, int anni) {
14   setNome(nome);
15   setRazza(razza);
16   setAnni(anni);
17 }
18  public String getNome() {
19   return nome;
20 }
21 public String getRazza() {
22   return razza;
23 }
24 public int getAnni() {
25   return anni;
26 }
```

Osservazioni (1)

All'interno di un blocco possono coesistere una variabile locale e una variabile di istanza con lo stesso nome. In tal caso si usa la parola chiave `this` per riferirsi alla variabile di istanza.

```
public void setNome(String nome) {
  this.nome = nome;
}
```

Invece, un assegnamento di questo tipo

```
public void setNome(String nome) {
  nome = nome;
}
```

È perfettamente lecito, ma non ha alcun effetto sullo stato dell'oggetto: assegna alla variabile `nome` locale al metodo il valore della variabile stessa. Non modifica la variabile di istanza!

Osservazioni (2)

Il corpo di un metodo può contenere l'invocazione di un altro metodo. Se il metodo invocato si trova all'interno della stessa classe, non serve invocarlo scrivendo il nome dell'oggetto.

```
public void setCane(String nome, String razza, int anni) {
  setNome(nome);
  setRazza(razza);
  setAnni(anni);
}
```

Si potrebbe anche usare la parola chiave `this`

```
public void setCane(String nome, String razza, int anni) {
  this.setNome(nome);
  this.setRazza(razza);
  this.setAnni(anni);
}
```

```java
// TestCane.java
// Uso corretto della classe Cane.java
public class TestCane {
  public static void main(String[] args) {
    Cane caneDiShaggy = new Cane();
    caneDiShaggy.setNome("Scooby-Doo");
    caneDiShaggy.setRazza("alano");
    caneDiShaggy.setAnni(9);

    caneDiShaggy.scriviInfo();

    Cane caneDiTopolino;
    caneDiTopolino = new Cane();
    caneDiTopolino.setCane("Pluto", "bracco", 7);

    System.out.println(caneDiTopolino.getNome() +
        " e' un " + caneDiTopolino.getRazza() +
        " di " + caneDiTopolino.getEtaInAnniUmani() + " anni umani.");
  }
}
```

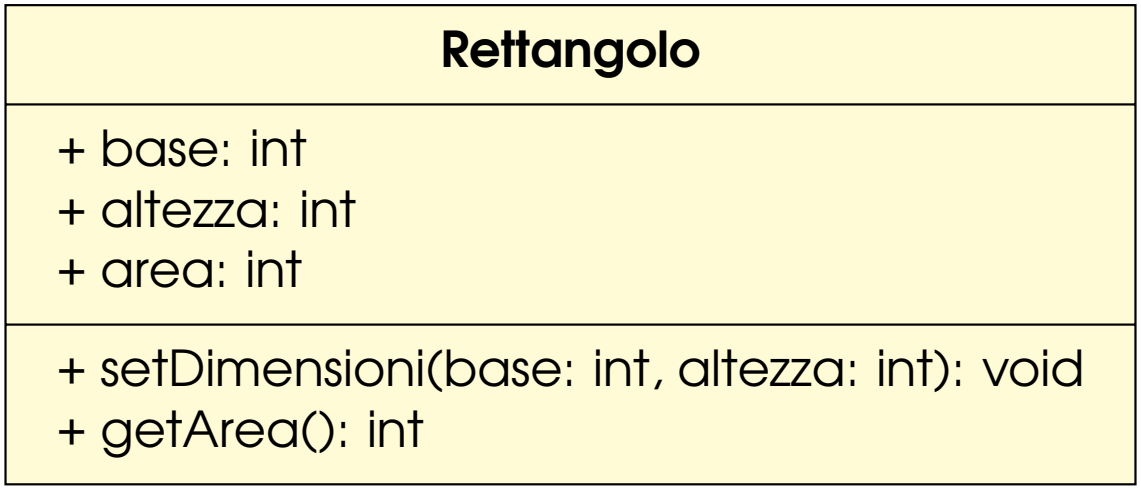
```
$ java TestCane
Nome: Scooby-Doo
Razza: alano
Eta': 9
Pluto e' un bracco di 47 anni umani.
```

Public o Private?

Normalmente tutti i metodi sono `public`. Se un metodo deve essere usato *solo* dagli altri metodi della sua classe, allora dovrebbe essere reso privato.

Tutte le variabili di istanza dovrebbero essere dichiarate `private`. In questo modo si costringe chi usa la classe ad accedere alle variabili di istanza *solo* attraverso i metodi della classe. Questo permette alla classe di controllare tutte le attività di lettura e scrittura dei valori delle variabili di istanza.

Esempio - variabili di istanza pubbliche



```java
// Rettangolo.java

public class Rettangolo {

  public int base;
  public int altezza;
  public int area;

  public void setDimensioni(int base, int altezza) {
    this.base = base;
    this.altezza = altezza;
    area = base * altezza;
  }

  public int getArea() {
    return area;
  }
}
```

Usiamo la Classe **Rettangolo**

```
Rettangolo box = new Rettangolo();
box.setDimensioni(10, 5);
System.out.println("L'area del rettangolo e' " +
    box.getArea());
```

Output: `L'area del rettangolo e' 50`

```
box.larghezza = 6;
System.out.println("L'area del rettangolo e' " +
    box.getArea());
```

Output: `L'area del rettangolo e' 50`

La possibilità di cambiare i valori alle variabili di istanza può portare a dati incoerenti all'interno di un oggetto!

Esempio - modifichiamo la classe **Rettangolo**

Rettangolo
- base: int - altezza: int - area: int
+ setDimensioni(base: int, altezza: int): void + getArea(): int

L'accesso alle variabili di istanza è regolato dai metodi.

Implementazione (1)

Rendiamo `private` le variabili di istanza.

```
1  // Rettangolo1.java
2
3  public class Rettangolo1 {
4
5    private int base;
6    private int altezza;
7    private int area;
8
9    public void setDimensioni(int base, int altezza) {
10     this.base = base;
11     this.altezza = altezza;
12     area = base * altezza;
13   }
14
15   public int getArea() {
16     return area;
17   }
18 }
```

Implementazione (2)

Non salviamo più l'area in una variabile di istanza, ma la calcoliamo solo quando è invocato il metodo `getArea`.

```
1  // Rettangolo2.java
2
3  public class Rettangolo2 {
4
5    private int base;
6    private int altezza;
7
8    public void setDimensioni(int base, int altezza) {
9      this.base = base;
10     this.altezza = altezza;
11   }
12
13   public int getArea() {
14     return base * altezza;
15   }
16 }
```

Implementazione e Comportamento

Le classi **Rettangolo1** e **Rettangolo2** sono implementate in maniera diversa, ma i loro metodi hanno lo stesso comportamento: le due classi fanno la stessa *cosa* ma in *modo* diverso.

Quale è la migliore?

Rettangolo1 usa più memoria, ha una variabile di istanza in più.

Rettangolo2 usa più tempo per calcolare l'area.

Dipende dall'utilizzo che se ne fa!

Encapsulation, in sintesi

Interfaccia

Indica ai programmatori ciò di cui hanno bisogno per usare la classe nei loro programmi. Consiste nell'intestazione dei metodi pubblici e le costanti pubbliche, insieme ai commenti che indicano al programmatore come usare metodi e costanti.

Implementazione

Tutto ciò che serve al programmatore che sta effettivamente scrivendo il codice della classe. Consiste in tutti gli elementi privati della classe, principalmente le variabili di istanza private e le definizioni dei metodi pubblici e privati.

Interfaccia e Implementazione non sono separate nel codice.

Una classe ben incapsulata

- Ha un commento (con lo stile `/**...*/`) prima della sua definizione che descrive cosa rappresenta la classe, senza descrivere come lo fa.
- Dichiara tutte le variabili di istanza come `private`
- Fornisce metodi pubblici `get` e `set` per gestire i dati di un oggetto.
- Ha un commento (con lo stile `/**...*/`) prima di ogni intestazione di metodo pubblico per spiegare come usare il metodo.
- Rende privati i metodi ausiliari.
- Presenta commenti (con lo stile `//...`) all'interno della classe per descrivere i dettagli implementativi.
- Si possono modificare i suoi dettagli implementativi senza dover modificare alcun programma che la usa.