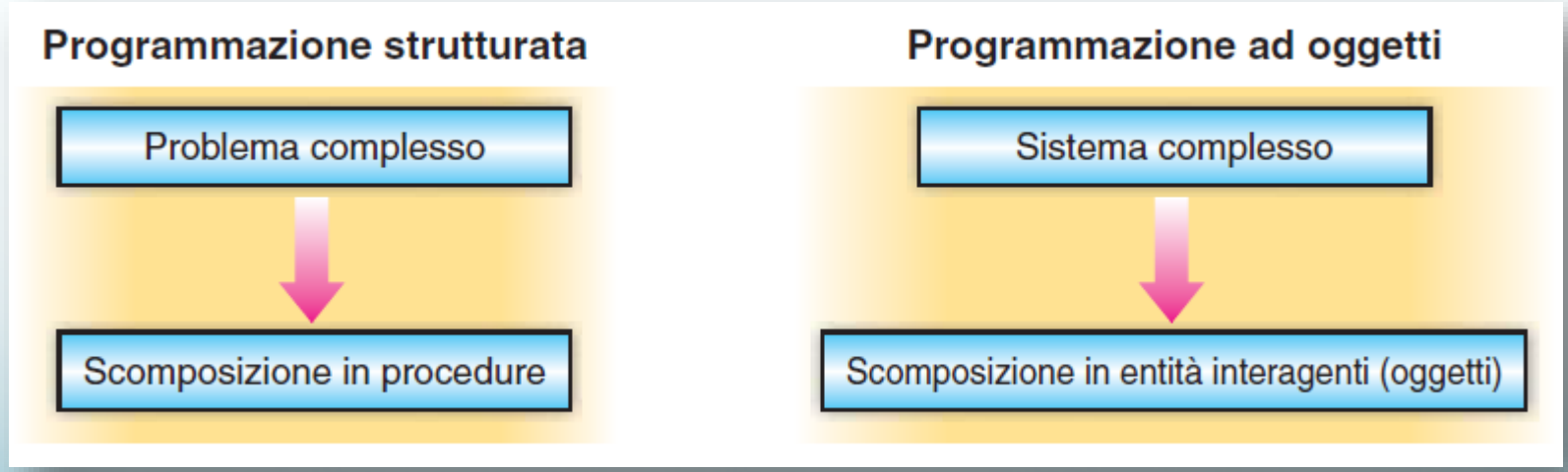
Classi e oggetti

A. Lorenzi, A. Rizzi
PRO.TECH Volume B
Atlas

© Istituto Italiano Edizioni Atlas

Oggetti

La programmazione orientata agli oggetti, **OOP** (*Object-Oriented Programming*), prende il nome dall'elemento su cui si basa, l'oggetto.



OOP

Vantaggi

- facilità di **lettura** e di **comprensione** del codice
- rapidità nella **manutenzione**
- **robustezza**
- **riusabilità** del codice

Oggetti

- Gli **oggetti** rappresentano le entità del problema o della realtà che si vuole automatizzare con l'informatica.
- Un oggetto è in grado di memorizzare le informazioni che riguardano il suo stato
- È possibile associare a un oggetto un insieme di operazioni che esso può compiere.

Attributi e metodi

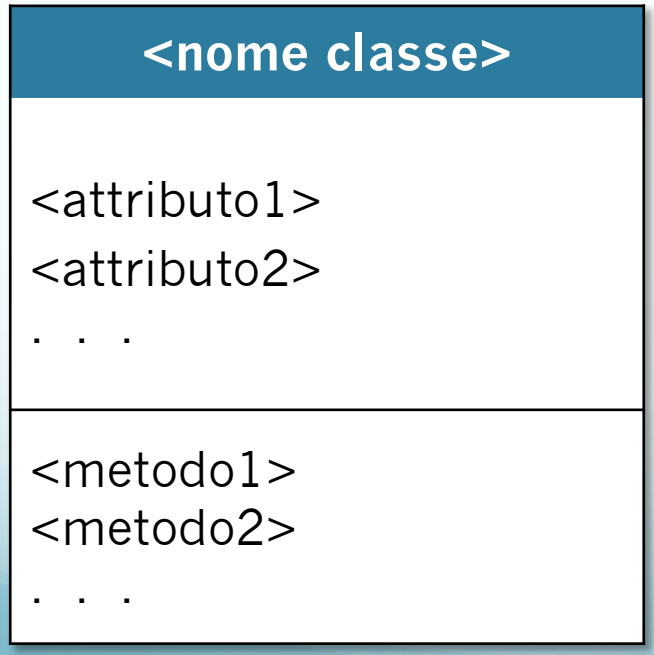
Gli **attributi** rappresentano gli elementi che caratterizzano l'oggetto, utili per descrivere le sue proprietà e definirne il suo stato.

I **metodi** rappresentano le funzionalità che l'oggetto mette a disposizione.

Definizione delle classi

La **classe** è la descrizione astratta degli oggetti attraverso gli attributi e i metodi.

- Diagramma delle classi

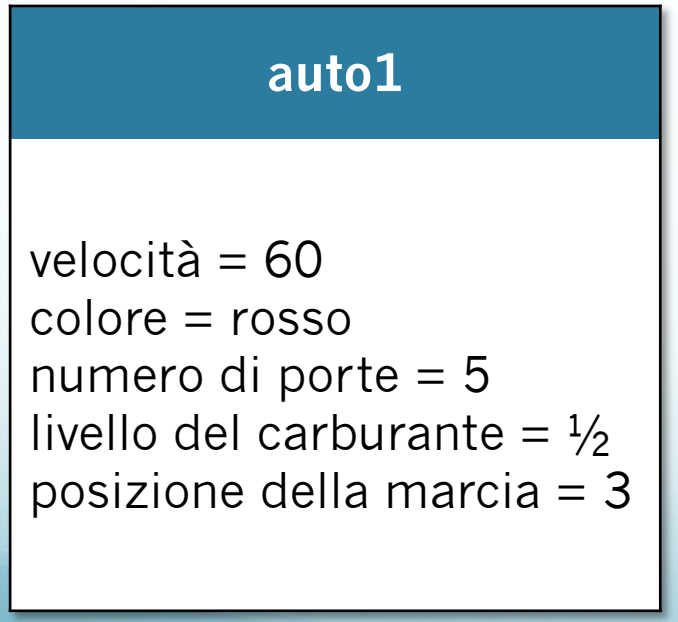


I simboli utilizzati negli schemi grafici rispettano gli standard del linguaggio **UML** (*Unified Modeling Language*)

Istanza

Per utilizzare un oggetto occorre crearlo come esemplare della classe, cioè come **istanza** di una classe.

- Diagramma degli oggetti



Ogni classe possiede un particolare metodo predefinito, detto **costruttore**, che viene attivato quando si crea un oggetto.

Incapsulamento

Il termine **incapsulamento** indica la proprietà degli oggetti di incorporare al loro interno sia gli attributi che i metodi, cioè le caratteristiche e i comportamenti dell'oggetto.

Dichiarazione di una classe

```
class NomeClasse
{
  // attributi
  // metodi
}
```

Dichiarazione di un oggetto

- **dichiarazione di un oggetto:**

```
NomeClasse nomeOggetto;
```

- **creazione dell'istanza**:

```
nomeOggetto = new NomeClasse();
```

Dichiarazione di un attributo

- dichiarazione di un attributo:

```
livelloDiVisibilità tipo nomeAttributo;
```

- Esempi:

```
private int x, y, z;

public double altezza = 15.76;
```

Livelli di visibilità

- **public**
  l'attributo è accessibile da qualsiasi altra classe
- **private**
  permette di nascondere l'attributo all'interno dell'oggetto
- **protected**
  è visto all'esterno solo dalle classi che appartengono alla stessa libreria oppure dalle sottoclassi della classe in cui è stato dichiarato l'attributo.

Altre caratteristiche

- **static**
  indica un attributo legato alla classe
- **final**
  se lo si vuole rendere una costante, in modo che possa assumere un unico valore.

Dichiarazione dei metodi

```
livelloDiVisibilità tipoRestituito nomeMetodo (parametri)
{
  // variabili
  // istruzioni
}
```

Esempio

- Il metodo che esegue la somma di due numeri interi passati come parametri:

```
public int addizione (int a, int b)
{
   int sum;

   sum = a+b;

   return sum;
}
```

Valore di ritorno

- Il **tipo del valore di ritorno** indica quale sarà il valore restituito al termine dell'esecuzione del metodo.
- Un metodo può anche non restituire alcun valore: parola chiave **void.**

```
public void setRaggio(double r)
{
  raggio = r;
}
```

Istruzione return

```
public int max(int a, int b)
{
  if (a > b)
  {
    return a;
  }
  else
  {
    return b;
  }
}
```

Parametri

- In Java, i parametri possono essere passati ai metodi solo **per valore**.
- Per quanto riguarda i **tipi riferimento**, cioè gli array e gli oggetti, viene creata una copia del riferimento e non dell'oggetto.

Livelli di visibilità

I livelli di visibilità di un metodo:

- **public**
  può essere richiamato da qualunque altra classe
- **private**
  non può essere richiamato esternamente alla classe in cui è dichiarato
- **protected**
  è visibile solo dalle classi che appartengono alla stessa libreria oppure dalle sottoclassi della classe in cui è dichiarato

Creazione di oggetti

- L'**allocazione dell'oggetto** riserva lo spazio per memorizzare gli attributi dell'oggetto e viene codificata con l'operatore **new** seguito dal nome della classe.

Viene attivato in modo implicito un particolare metodo predefinito, detto **costruttore** della classe, che consente di creare un oggetto.

```
Cerchio tavolo = new Cerchio();
```

Creazione di oggetti

Creare un'istanza della classe *Cerchio* usando il costruttore e indicando un valore di inizializzazione per l'attributo *raggio*.

```
Cerchio ruota = new Cerchio(0.41);
```

Utilizzo degli oggetti

- Accesso agli attributi di un oggetto con l'**operatore punto**:

```
nomeOggetto.attributo
```

- **Invocazione di un metodo:**

```
nomeOggetto.metodo(parametri)
```

- In ogni classe è implicitamente presente un oggetto speciale identificato dalla parola chiave **this**, per indicare l'oggetto a cui il costruttore si riferisce.

Information hiding

Il termine **information hiding** indica il mascheramento delle modalità di implementazione di un'oggetto, rendendone disponibile all'esterno solo le funzionalità.

Metodi per leggere e modificare il valore degli attributi:

- **get:** leggere il valore di un attributo e restituirlo
- **set:** modificare il valore di un attributo.

Esempio

```
class Automobile
{
  private final int POSIZIONE_MAX = 6;
  private int posMarcia;

  public Automobile()
  {
   posMarcia = 1;
  }

  public int getMarcia()
  {
   return posMarcia;
  }

  public void setMarcia(int m)
  {
   if ((m >= 1) && (m <= POSIZIONE_MAX))
   {
    posMarcia = m;
   }
  }
 }
```

Interfaccia

- il termine interfaccia riferito alle classi:

L'**interfaccia** di una classe indica l'elenco dei metodi pubblici, cioè l'insieme delle funzionalità utilizzabili dalle istanze della classe.

- per esempio, l'interfaccia della classe *Automobile*:

```
public Automobile()
public int getMarcia()
public void setMarcia(int m)
```

- I dettagli sulle caratteristiche e la struttura dell'oggetto sono nascosti all'interno, garantendo l'*information hiding* (mascheramento dell'informazione).

Array di oggetti

La dichiarazione e l'allocazione di un **array di oggetti:**

- la dichiarazione di un array di 10 cerchi

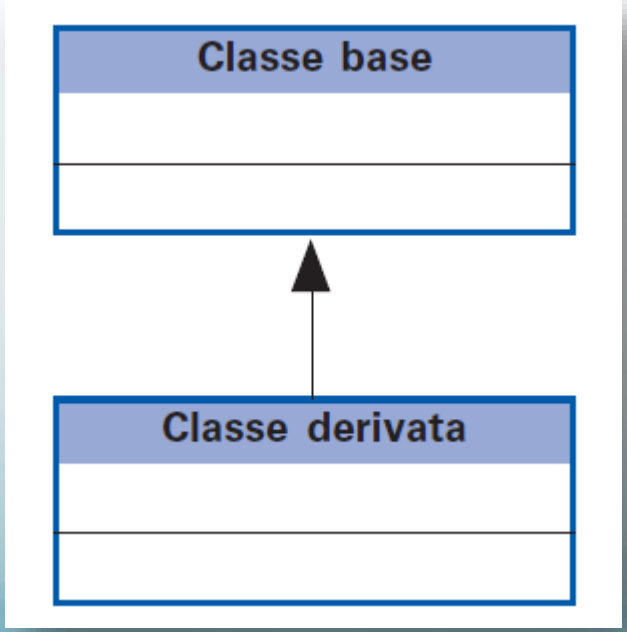
```
Cerchio collezione[] = new Cerchio[10];
```

- allocare i singoli oggetti

```
collezione[0] = new Cerchio(0.58);
collezione[1] = new Cerchio(4.3);
```

Ereditarietà

L'**ereditarietà** rappresenta la possibilità di creare nuove classi a partire da una classe già esistente (**classe base**). La nuova classe eredita tutti gli attributi e i metodi della classe base e può essere arricchita con nuovi attributi e nuovi metodi. La classe così ottenuta si chiama **classe derivata**..

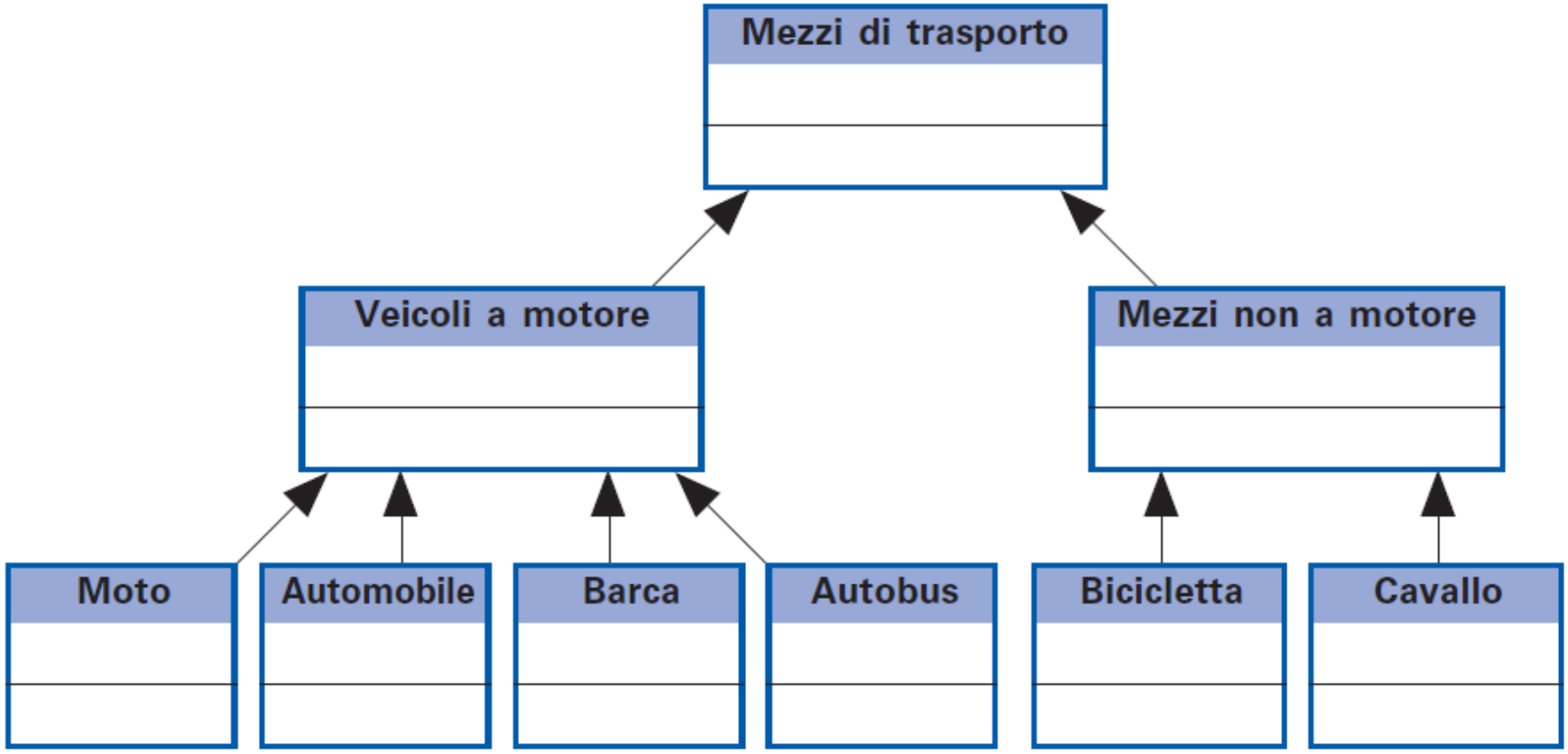


Sottoclasse e superclasse

La classe che è stata derivata da un'altra usando l'ereditarietà prende il nome di **sottoclasse.**

La classe generatrice di una sottoclasse si chiama **superclasse** o *sopraclasse*.

Grafo di gerarchia



Classe derivata

La nuova classe si differenzia dalla sopraclasse:

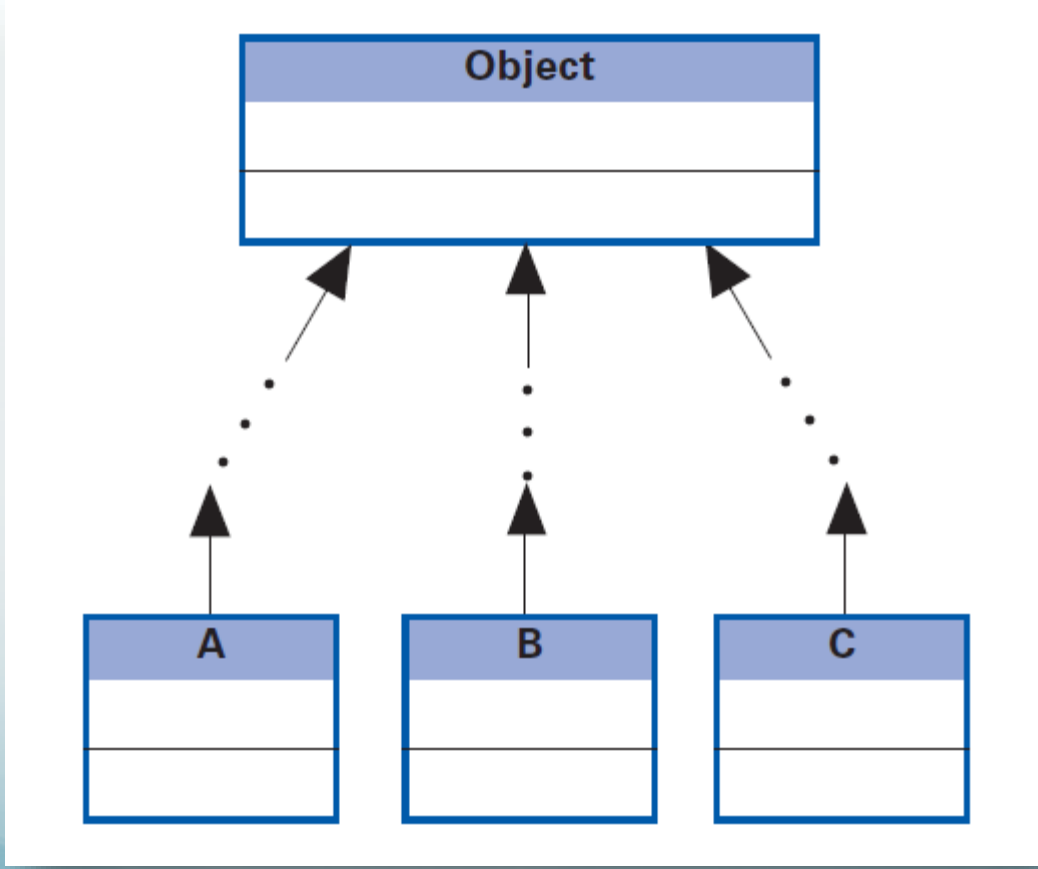
- per **estensione:** la sottoclasse aggiunge nuovi attributi e metodi
- per **ridefinizione:** la sottoclasse ridefinisce i metodi ereditati (**overriding** del metodo)

Dichiarazione di una sottoclasse

```
class NomeSottoClasse extends NomeSopraClasse
{
  . . .
}
```

Classe Object

Classe da cui vengono derivate tutte le altre



Polimorfismo

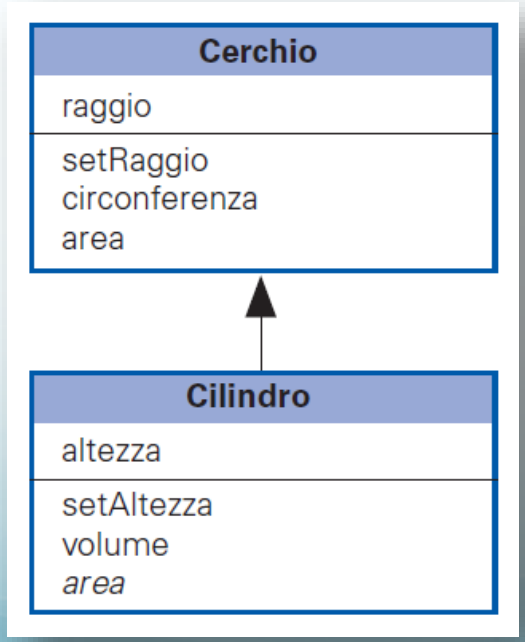
Il **polimorfismo** indica la possibilità per i metodi di assumere forme, cioè implementazioni, diverse all'interno della gerarchia delle classi.

Due tipi di polimorfismo:

- l'**overriding**, o *sovrapposizione* dei metodi,
- l'**overloading**, o *sovraccarico* dei metodi.

Overriding

- **Overriding:** ridefinire, nella classe derivata, un metodo ereditato con lo scopo di modificarne il comportamento. Il nuovo metodo deve avere lo stesso nome e gli stessi parametri del metodo che viene sovrascritto.



Overloading

- L'**overloading** di un metodo è la possibilità di utilizzare lo stesso nome per compiere operazioni diverse.
- Solitamente si applica ai metodi della stessa classe che si presentano con lo stesso nome, ma con un numero o un tipo diverso di parametri.

Librerie

Le **librerie** sono un insieme di classi già compilate che possono essere richiamate e usate all'interno dei programmi. Java usa il termine **package.**

Esempi:

- **java.applet**
- **java.awt**
- **java.io**
- **java.lang**
- **java.net**
- **java.util**

Librerie

Per riferirsi a una classe contenuta in un particolare package:

```
NomePackage.NomeClasse
```

Per evitare di ripetere più volte il nome del package: comando **import**

```
import java.awt.Button
Button b;
b = new Button();
;
```

Librerie

Per importare tutte le classi contenute nel package **java.io**:

```
import java.io.*;;
```

Il package **java.lang** contiene la dichiarazione delle classi di base del linguaggio Java.

Classe String

Inclusa nel package *java.lang*.

- Per usare le stringhe in Java si deve creare un oggetto di classe *String*.

```
String nome = new String("Elena");
```

- altro modo per allocare le stringhe:

```
String nome = "Laura";
```

Classe String