# L5 - Progettazione e UML - Avanzati

Java · ereditarieta', interfacce e associazioni nel diagramma

*Materiale didattico di Moreno Sardella · morenosardella.com*

**Prima di iniziare.** Questi esercizi sono piu' impegnativi: combinano piu' concetti e si avvicinano al livello della verifica. Progetta prima su carta (classi, metodi, dati), poi scrivi il codice. Le soluzioni in fondo mostrano *una* via corretta: la tua puo' differire, conta che la logica funzioni e che il codice compili.

## Esercizi

### Esercizio 1 — Gerarchia di ereditarieta'

**GENERALIZZAZIONE**

Disegna l'UML di una gerarchia **Veicolo** (superclasse, attributo targa) con due sottoclassi **Auto** e **Moto**. Indica la freccia di generalizzazione (triangolo vuoto) verso Veicolo, poi scrivi gli scheletri Java.

```
// Auto e Moto extends Veicolo
```

### Esercizio 2 — Realizzazione di interfaccia

**INTERFACCIA IN UML**

Data l'interfaccia **Pagabile** con *+getImporto():double*, disegna la classe **Fattura** che la realizza (freccia tratteggiata con triangolo vuoto) e scrivi il codice di Fattura.

```
// Fattura implements Pagabile
```

### Esercizio 3 — Associazione con molteplicita'

**1 A MOLTI**

Modella **Ordine** che contiene piu' **Articolo** (1 Ordine -> * Articolo). Indica la molteplicita' sul diagramma e traduci l'associazione in un attributo ArrayList nel codice di Ordine.

```
private ArrayList&lt;Articolo&gt; articoli;
```

### Esercizio 4 — Progetto completo dal testo

**ANALISI -> UML -> CODICE**

*"Una scuola ha docenti e studenti. Un docente insegna una materia. Uno studente ha una media."* Individua le classi, disegna l'UML con le associazioni e scrivi gli scheletri delle classi (solo attributi e firme).

# Soluzioni

*Confronta solo dopo aver provato da solo. Esistono soluzioni equivalenti: conta la logica, non la copia esatta.*

## Soluzione 1

```
class Veicolo {
    protected String targa;
    Veicolo(String t) { targa = t; }
}
class Auto extends Veicolo {
    Auto(String t) { super(t); }
}
class Moto extends Veicolo {
    Moto(String t) { super(t); }
}
```

## Soluzione 2

```
interface Pagabile { double getImporto(); }
class Fattura implements Pagabile {
    private double prezzo; private int quantita;
    public double getImporto() { return prezzo * quantita; }
}
```

## Soluzione 3

```
class Ordine {
    private ArrayList&lt;Articolo&gt; articoli = new ArrayList&lt;&gt;();
    public void aggiungi(Articolo a) { articoli.add(a); }
    public double totale() {
        double t=0; for (Articolo a : articoli) t += a.getPrezzo(); return t;
    }
}
```

## Soluzione 4

**Classi**: Scuola, Docente, Studente. **Docente**: nome, materia. **Studente**: nome, media. **Scuola**: liste di Docente e Studente. Associazioni: Scuola 1->* Docente, Scuola 1->* Studente.

```
class Docente { String nome, materia; }
class Studente { String nome; double media; }
class Scuola {
    ArrayList&lt;Docente&gt; docenti;
    ArrayList&lt;Studente&gt; studenti;
}
```