Relazioni fra classi

Quando si progetta un programma, torna utile documentare le relazioni fra le classi; così facendo, si ottengono una serie di vantaggi. Per esempio, se si trovano classi caratterizzate da un comportamento comune, si può risparmiare un po' di fatica collocando il comportamento comune in una superclasse.
Se si sà che certe classi *non* sono correlate tra loro, si può assegnare a programmatori diversi il compito di implementare ciascuna di esse, senza avere la preoccupazione che uno di loro debba aspettare l'altro.

L'***ereditarietà*** è una relazione fra classi molto importante, ma non è l'unica relazione utile e si può anche correre il rischio di abusarne.

L'ereditarietà è una relazione fra una classe più generale (la superclasse) e una classe più specializzata (la sottoclasse). In questi casi si usa dire che si tratta di una relazione *è-un*. Ogni autocarro *è un* veicolo. Ogni conto corrente *è un* conto bancario. Ogni cerchio *è una* ellisse (con altezza e larghezza uguali).

L'ereditarietà (la relazione "*è–un*") viene a volte usata a sproposito, quando invece una relazione "*ha–un*" sarebbe più opportuna.

Tuttavia, spesso si abusa dell'ereditarietà.
Per esempio, si prenda in considerazione la classe Tire, che descrive un pneumatico di automobile.
Dovremmo considerare la classe Tire come una sottoclasse di Circle?
A prima vista sembra comodo, perché vi sono diversi metodi utili nella classe Circle: per esempio, la classe Tire erediterebbe i metodi che calcolano il raggio, la circonferenza e il punto centrale. Tutte cose che potrebbero venir buone quando si disegnano forme di pneumatici.
Eppure, anche se potrebbe essere comodo per il programmatore, questa impostazione non ha senso dal punto di vista concettuale. Non è vero che ogni pneumatico è un cerchio. I pneumatici sono componenti delle automobili, mentre i cerchi sono oggetti geometrici.
Esiste tuttavia una relazione fra pneumatici e cerchi: un pneumatico *ha un* cerchio come suo perimetro esterno. Java ci consente di fare un modello anche di tale relazione, **utilizzando una variabile istanza**:

```
class Tire {
        ...
        private String rating;
        private Circle boundary;
}
```

Il termine tecnico per questa relazione è ***associazione***. Ogni oggetto di tipo Tire è associato a un oggetto di tipo Circle.

Ecco un altro esempio: ogni automobile *è un* veicolo; ogni automobile *ha un* pneumatico (in realtà ne ha quattro, cinque se contate anche la ruota di scorta). Di conseguenza, si utilizzerà l'ereditarietà dalla classe Vehicle e l'associazione con gli oggetti Tire:

```
class Car extends Vehicle {
        ...
        private Tire[] tires;
}
```

Utilizzando la notazione UML per i diagrammi delle classi si indica la relazione di ereditarietà mediante una freccia con un triangolo aperto che punta alla superclasse. Nella notazione UML, l'associazione viene indicata mediante una linea a tratto continuo con una freccia aperta (se associazione *diretta* con navigazione unidirezionale). La Figura mostra un diagramma di classi con una relazione di ereditarietà e un'associazione.

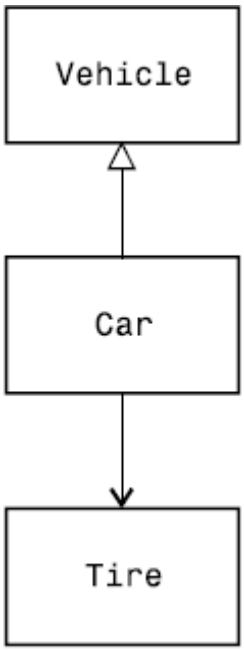


Figura:
Notazione UML
per **ereditarietà**
e **associazione**

Una classe è associata a un'altra se è possibile *navigare* da oggetti della prima classe a oggetti dell'altra classe. Ad esempio, dato un oggetto di tipo Car, potete giungere a oggetti di tipo Tire accedendo semplicemente alla variabile istanza tires.

Quando una classe ha una variabile istanza il cui tipo è quello di un'altra classe, fra le due classi esiste un'associazione.

Una classe è associata a un'altra se potete spostarvi da suoi oggetti a oggetti dell'altra classe, solitamente seguendo un riferimento a oggetto.

La relazione di associazione si ricollega alla relazione di ***dipendenza***.
Una classe dipende da un'altra se uno dei suoi metodi ***usa*** un oggetto di tale classe in qualche modo.
Per esempio, tutte le classi applet dipendono dalla classe Graphics, perché ricevono un oggetto di tipo Graphics nel metodo paint e lo usano per disegnare varie forme.
Le applicazioni di console dipendono dalla classe System, perché usano la variabile statica System.out.

La dipendenza è un altro nome per la relazione "usa".

L'associazione è una forma più forte di dipendenza. Se una classe è associata a un'altra, ne dipende anche.
Tuttavia, il contrario non è vero. Se una classe è associata a un'altra, i suoi oggetti possono localizzare oggetti della classe associata, solitamente perché ne memorizzano riferimenti. Se una classe dipende da un'altra, ciò significa che viene in contatto con oggetti di tale classe in qualche modo, ma non necessariamente attraverso la navigazione.
Ad esempio, un applet dipende dalla classe Graphics, ma non è associato alla classe Graphics.
Dato un oggetto di tipo Applet, non potete navigare fino a un oggetto di tipo Graphics: dovete aspettare che il metodo paint fornisca un oggetto di tipo Graphics come parametro.

Nella notazione UML la dipendenza è rappresentata mediante una linea tratteggiata e la navigabilità con una freccia aperta che punta alla classe dipendente.

Le frecce usate nella notazione UML possono confondere; la tabella seguente riassume i quattro simboli usati nella notazione UML per rappresentare le relazioni descritte:

Relazione	Simbolo	Tratto	Punta della freccia
Ereditarietà		Continuo	Chiusa
Implementazione		Tratteggio	Chiusa
Associazione		Continuo	Aperta
Dipendenza		Tratteggio	Aperta

Si deve essere in grado di distinguere le notazioni UML per l'ereditarietà, l'implementazione, l'associazione e la dipendenza.

NB: si può non elencare come attributo ciò che si rappresenta graficamente come associazione. Ad esempio, se si indica con un'associazione il fatto che un oggetto di tipo Car contiene oggetti di tipo Tire, non si aggiunge l'attributo tires alla classe Car.

Associazione, aggregazione e composizione

La relazione di associazione è la relazione più complessa nella notazione UML, ed è anche quella meno standard. Se leggete altri libri e osservate diagrammi di classi prodotti da colleghi programmatori, potete riscontrare alcuni diversi stili di rappresentazione della relazione di associazione.

La relazione di associazione che si è usata viene detta associazione *diretta*: essa implica che sia possibile navigare da una classe a un'altra, ma non viceversa.

Ad esempio, dato un oggetto di tipo Car, si può navigare fino a raggiungere oggetti di tipo Tire. Ma se si ha un oggetto di tipo Tire, non esiste alcuna indicazione di quale sia l'automobile al quale appartiene. Ovviamente, un oggetto di tipo Tire può contenere un riferimento all'oggetto di tipo Car a cui appartiene, in modo che si possa navigare a ritroso dal pneumatico fino all'automobile: in questo caso l'associazione è bidirezionale. Per automobili e pneumatici, questa implementazione sarebbe poco probabile, ma si consideri l'esempio di oggetti di tipo Person e Company.

Una ditta può conservare un elenco delle persone che vi lavorano, e ciascun oggetto che rappresenta una persona può avere un riferimento al proprio datore di lavoro.

Secondo lo standard UML, un'*associazione bidirezionale* si rappresenta con una linea a tratto continuo *senza alcuna freccia* (si osservi la Figura seguente per questa e altre varianti della notazione per l'associazione), pur se alcuni progettisti interpretano un'associazione senza frecce come un'associazione "indecisa", nella quale non è ancora chiaro in quale direzione possa avvenire la navigazione.

Ad alcuni progettisti piace aggiungere *ornamenti* alle relazioni di associazione.

Un'associazione può avere un nome, dei ruoli o delle molteplicità. Un **nome** descrive la natura della relazione. Gli **ornamenti** che indicano i ruoli esprimono in modo specifico il ruolo che le classi associate svolgono l'una rispetto all'altra.

Le **molteplicità** indicano quanti oggetti si possono raggiungere navigando lungo la relazione di associazione.

L'esempio di Figura (il quarto diagramma) esprime il fatto che ogni pneumatico (oggetto della classe Tire) è associato a un numero di automobili uguale a 0 o 1, mentre ogni automobile (oggetto della classe Car) deve avere 4 o più pneumatici.

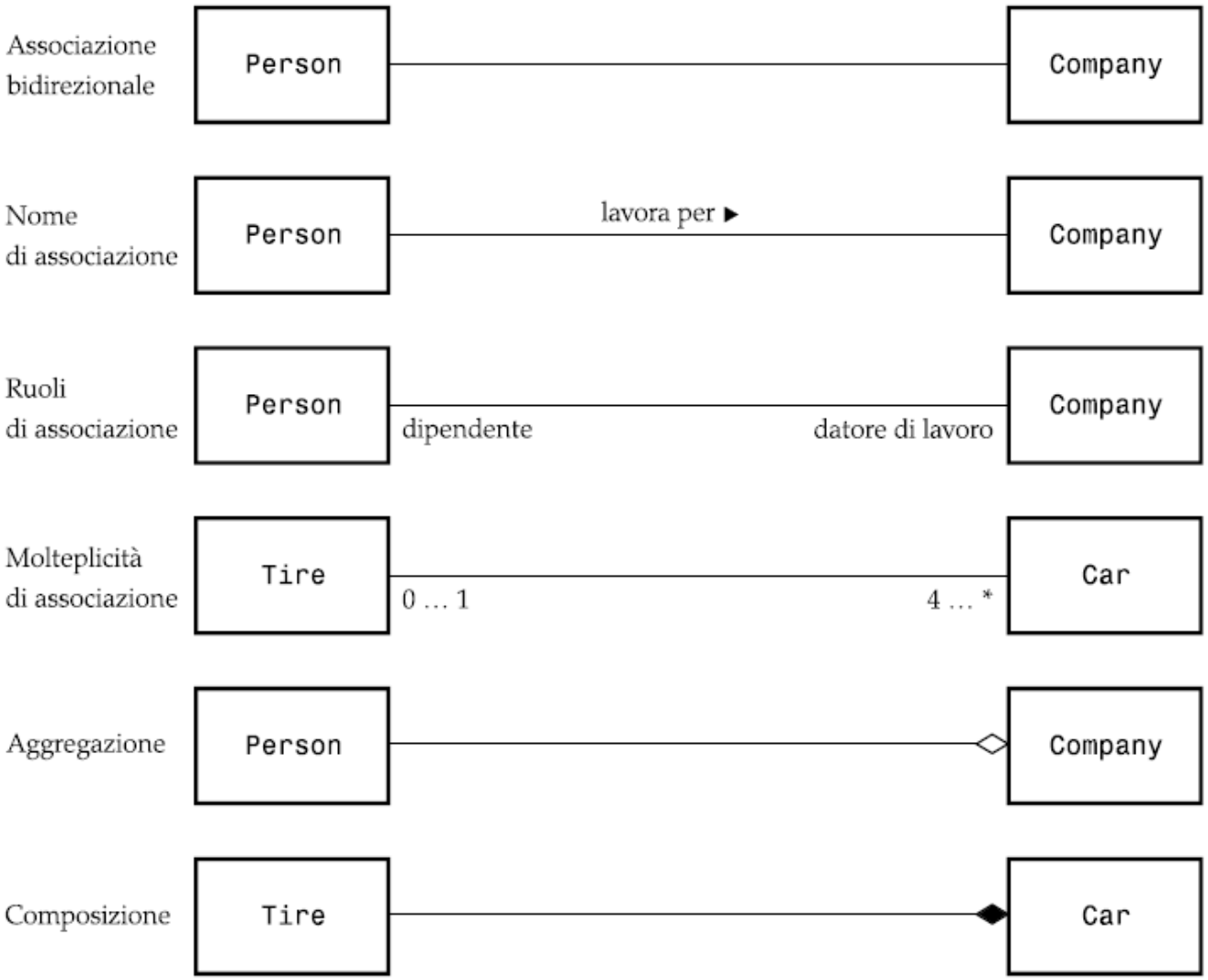


Figura
Varianti della notazione di associazione
e forme più forti (aggregazione, composizione)

L'***aggregazione*** è una forma più forte di associazione. Una classe ne aggrega un'altra se esiste tra le due classi una relazione "**intero-parte**". Ad esempio, la classe Company aggrega la classe Person, perché una ditta ("intero") è composta di persone ("parte"), in particolare i suoi dipendenti e i suoi clienti. La classe BankAccount, invece, non aggrega la classe Person, anche se può darsi che sia possibile navigare da un oggetto che rappresenta un conto bancario fino a un oggetto che rappresenta una persona, il proprietario del conto. Dal punto di vista concettuale, una persona non fa parte di un conto bancario.

La ***composizione*** è una forma di aggregazione ancora più forte, che indica che una "parte" può appartenere a un solo "intero" in un certo istante di tempo. Ad esempio, un pneumatico può far parte di una sola automobile in un certo istante, mentre una persona può lavorare contemporaneamente per due ditte.

Sinceramente, le differenze fra associazione, aggregazione e composizione possono confondere anche i progettisti esperti. Se si pensa che queste distinzioni siano utili, si usino senza dubbio, ma non si perda tempo pensando alle sottili differenze fra questi concetti.

Dal punto di vista pratico di un programmatore **Java**, è utile sapere se una classe **conserva un riferimento a un'altra classe**, e l'***associazione diretta*** descrive accuratamente questo fatto.

Diagrammi UML e consigli pratici

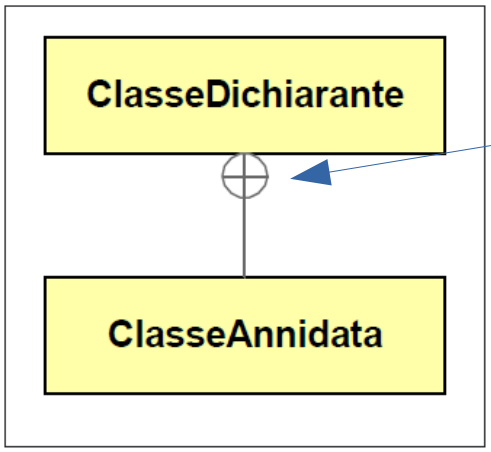
Prima di scrivere il codice per un problema complesso, è necessario averne progettato una soluzione.

La metodologia presentata suggerisce di seguire un procedimento di progettazione composto dalle seguenti fasi:

1. Identificate le classi.
2. Determinate le responsabilità di ciascuna classe.
3. Descrivete le relazioni tra le classi.

A volte si privilegia progettare delle classi annidate (***nested class***) o interne (***inner class***) cioè definite all'interno della definizione di un'altra classe.

Rappresentazione di classi annidate.



Se si vuole rappresentare tale relazione in diagrammi UML in modo meno generico si può utilizzare un' icona ad ancora come illustrato in figura

icona ad ancora (cerchio crociato)

indica che la classe che si trova di fianco al cerchio crociato **contiene** la classe che si trova all'altra estremità della linea

Esercizio:

- In un sistema per la gestione di contatti on-line, gli utenti sono identificati da nome, cognome, login e password. Gli utenti usano il sistema per gestire delle agende telefoniche. Le agende sono organizzate per contatti e, per ogni contatto, è possibile memorizzare un nominativo e uno o più numeri telefonici associati.

- Le operazioni che più frequentemente gli utenti effettuano sulle agende di propria competenza riguardano l'inserimento e la cancellazione di un nuovo contatto o numero di telefono, la modifica di un contatto o dei relativi numeri di telefono, e la ricerca dei numeri di telefono associati ad un particolare contatto.

UML nella descrizione di relazioni tra le classi: