PROGRAMMAZIONE II

LAUREA TRIENNALE IN SCIENZE INFORMATICHE

Magliani Andrea
Perego Luca

Università degli studi di Milano-Bicocca

A.A. 2022/2023

INDICE

Classi, Oggetti & Istanze

1.1 Definizioni

Classe: Raccolta di proprietà comuni ad un insieme di istanze;

Istanza: Concretizzazione degli attributi e i comportamenti descritti da una classe, è sinonimo di oggetto;

1.2 Astrazione

L'astrazione è il processo concettuale tramite il quale si definisce un concetto generico partendo da una sua istanza specifica ed estraendo solamente gli aspetti generali.

Astrazioni Strutturali

2.1 Classificazione & Istanziazione

Si dice **istanziazione** l'azione tramite la quale si crea un oggetto rappresentato astrattamente da una classe.

Si dice **classificazione** l'azione tramite la quale si crea una classe rappresentando le caratteristiche e comportamenti di una particolare istanza.

La classificazione si indica con la keyword **istance_of**.

2.2 Generalizzazione

La generalizzazione lega una **superclasse** (classe padre) ad un o più **sottoclassi** (classi figlie). Ogni sottoclasse possiede tutte le caratteristiche della sua superclasse.

La generalizzazione si indica con la keyword **is_a**.

2.3 Associazione

L'associazione definisce una connessione logica tra 2 oggetti, senza definire un interpretazione.

La generalizzazione si indica con la keyword **has_a**.

2.4 Aggregazione

L'aggregazione è un caso particolare dell'associazione: lega un classe "aggregato" con un insieme di classi "parti".
Ogni aggregato è costituito da parti.

L'aggregazione si indica con la keyword **part_of**.

Formalismi di rappresentazione

3.1 ER [Entity-Relationship]

1. Entità
2. Relazioni

Il formalismo ER è nato nell'area "Basi di dati"

3.2 UML [Unified Modeling Language]

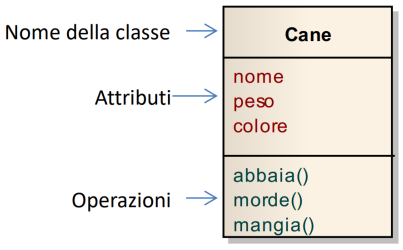
1. Classi
2. Proprietà
3. Relazioni
4. Comportamenti
5. …

Il formalismo UML è nato nell'area "Programmazione"

UML: diagramma delle classi

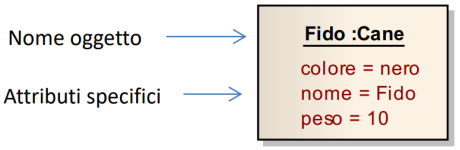
4.1 Classi

Identificazione delle classi e delle relazioni fra le stesse. Attività fondamentale in fase di **analisi** e di **progetto**.



4.2 Oggetti

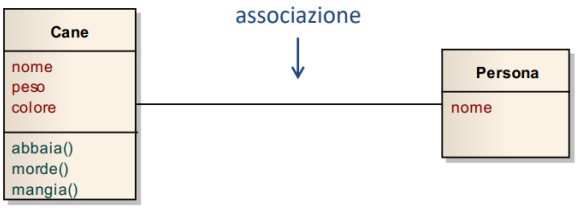
Identifica gli oggetti del sistema e i loro stati durante la loro vita.



Associazioni

5.1 Definizione

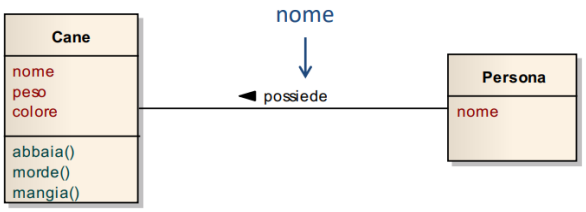
Un'associazione è una linea che collega le classi coinvolte.



5.2 Nome

Esprime il significato dell'associazione.

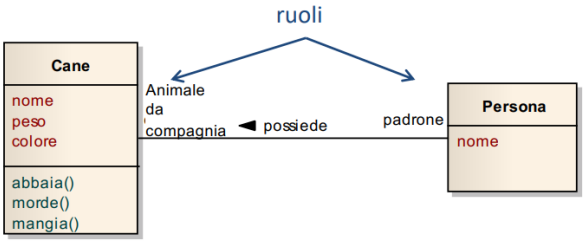
1. Spesso è un verbo
2. Opzionale [ma consigliato]



5.3 Ruolo

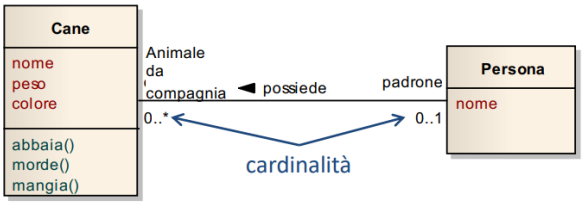
Esprime il ruolo giocato dal partner.

1. Spesso è un sostantivo o un aggettivo.
2. Opzionale [ma consigliato]



5.4 Cardinalità

Esprime quante istanze della classe possono essere associate all'altra classe.



5.5 Associazioni multiple

L'associazione multipla esplicita più associazioni tra una coppia di classi per arrichirne la descrizione.

Le associazioni tra una coppia di classi possono variare in:

1. Ruolo
2. Direzione
3. Nome

Teoricamente, ogni istanza è rappresentata da un'unica associazione tra quelle disponibili della coppia di classi, ma non è obbligatorio seguire questa norma.

5.6 Associazioni cappio

Esistono associazioni cappio, ovvero classi poste in relazione a loro stesse.

5.7 Navigabilità delle associazioni

Le associazioni hanno una proprietà detta navigabilità, se hanno una freccia che esplicita la modalità in cui viene realizzata.
Una classe con un'associazione in ingresso non ha accesso ai dati della classe a cui è collegata.
L'assenza di frecce implica un'associazione con **navigabilità bidirezionale**.

Aggregazioni

6.1 Definizione

Le aggregazioni sono relazioni **poco forti**:

1. Si rappresenta un diamante vuoto.
2. Le parti possono esistere indipendentemente dal tutto.
    a. *Ex. La stampante esiste anche senza computer*
3. Il tutto può in alcuni casi esistere indipendentemente dalle parti, ma in altri casi no.
    a. *Ex. Il computer può esistere anche senza stampante*
4. Il tutto è in qualche modo incompleto se mancano alcune delle sue parti.
    a. *Ex. Azienda senza lavoratori*
5. E’ possibile che più aggregati condividano una stessa parte.
    a. *Ex. Stampante condivisa*

Composizioni

7.1 Definizione

Le composizioni sono relazioni **molto forti**:

1. Si rappresenta un diamante pieno.

2. Ogni parte può appartenere ad un solo tutto per volta.
    a. *Ex. Il motore appartiene ad una sola auto*

3. Il tutto è l'unico responsabile di tutte le sue parti: questo vuol dire che è responsabile della loro creazione e distruzione.

4. Il tutto può rilasciare una sua parte, a patto che un altro oggetto si prenda la relativa responsabilità.
    a. *Ex. Il motore viene montato su un'altra auto*

5. Se il tutto viene distrutto, devo distruggere tutte le sue parti o cederne la responsabilità a qualche altro oggetto.
    a. *Ex. L'auto si rottama tutta, oppure si cedono le parti ad altre auto*

Generalizzazione

8.1 Definizione

La generalizzazione è rappresentata da triangolo vuoti (puntati verso la superclasse).

La superclasse è detta anche **classe padre**.
Le classi a lei collegate sono anche dette **classi figlie**.

Object Oriented Programming

9.1 OOP e Oggetti

Java è un linguaggio di programmazione ad oggetti.
Un oggetto è un contenitore che incapsula dati ed algoritmi.

I dati contenuti sono gli elementi che determinano lo stato dell'oggetto, mentre gli algoritmi contenuti sono incapsulati nei metodi della classe generatrice dell'oggetto

Tramite un oggetto è possibile rappresentare un'entità reale o astratta tramite un processo di astrazione.

9.2 Caratteristiche di un oggetto

Stato:

Una delle condizioni in cui può trovarsi l'oggetto.
Lo stato di un oggetto è definito dalle sue variabili e attributi.
Gli attributi di un oggetto possono avere qualsiasi tipo, primitivo o non primitivo.

Comportamento:

Determina la risposta di un oggetto a una determinata richiesta.

Il comportamento di un oggetto è il modo in cui risponde alle richieste esterne.
Gli oggetti comunicano scambiandosi chiamate attraverso le loro interfacce.

Le chiamate attivano i metodi, i quali infine determinano il comportamento dell'oggetto stesso.

Identità:

Gli oggetti dispongono di un’identità univoca: due oggetti che si trovano nello stesso stato e hanno lo stesso comportamento sono comunque 2 entità distinte.

Ogni oggetto dispone di un Object Identifier [OID], un codice univoco e invariante nel tempo.

9.3 Classi

Tutti gli oggetti che condividono proprietà e comportamenti possono essere classificati assieme.

Una classe rappresenta una e una sola astrazione dalla quale è possibile creare oggetti.

La classe è una struttura che definisce le caratteristiche delle sue istanze.

Una classe specifica l’interfaccia che ogni sua istanza offre agli altri elementi con cui interagisce, essa:

- Definisce comandi: nome, return e parametri formali
- Implementa: attributi, metodi interni e operazioni di questi

Information Hiding

Le classi dispongono anche del principio detto information hiding: questa caratteristica permette di impostare la visibilità dei suoi elementi, rendendoli invocabili dall’esterno o invisibili.

Questa tecnica permette modularità nella programmazione e inoltre lascia cambiare l’implementazione di una classe senza farlo notare agli elementi che interagiscono esternamente.

Gestione della memoria

10.1 Gestione di memoria

Le dichiarazioni delle variabili vengono salvate nella **memoria stack** e inizializzate con valore undefined.
Le assegnazioni vengono salvate nella **memoria heap**.
Le dichiarazioni di un oggetto viene gestita come le variabili, ma l'inizializzazione di questo avviene con i valori standard di ogni tipo (int = 0, char = "",..),
In Java l'operatore di confronto permette di verificare l'**identità** di due elementi:

1. Nel caso di variabili primitive, l'identità è uguale se entrambe contengono lo stesso valore.

2. Nel caso degli oggetti l'identità confronta l'**area di memoria** legata alla dichiarazione di questi.

10.2 Dot notation

Per accedere ad un attributo di un oggetto, si utilizza la scrittura: oggetto.attributo.

Questo metodo è detto **dot notation** e permette di confrontare gli attributi degli oggetti invece dei loro indirizzi di memoria.

10.3 Debug

Il sistema di debug permette di controllare il codice senza scrivere output al suo interno.

Tramite gli IDE è possibile posizionare **breakpoint**, ovvero punti in cui il sistema di debug si ferma permettendo di controllare il codice un'istruzione alla volta.

Il debug permette di visualizzare identificativo e attributi di un oggetto, aggiornando il loro stato in tempo reale.
Tramite gli identificativi è possibile capire se due istanze fanno riferimento a istanze diverse.

10.4 Array di oggetti

Gli array di oggetti contengono le **reference** agli oggetti in questione, in quanto vengono memorizzati gli indirizzi di memoria degli oggetti.

Gli array di oggetti si dichiarano nel seguente modo:
`Oggetto [] nomeArray;`

Questi array sono di fatto array di reference.
I valori di inizializzazione di ogni cella è null (in quanto è il valore di inizializzazione degli oggetti).

Ereditarietà

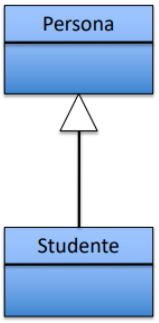
11.1 Definizione

L'ereditarietà è il processo per cui una classe specializzata è creata partendo dalle caratteristiche di una classe generalizzata.

11.2 Caratteristiche

La classe derivata ha tutte le variabili e i metodi della classe base, oltre a poter istanziarne di nuove.

La classe derivata ha tutte le funzionalità della classe generalizzata.



11.3 Vantaggi

L'ereditarietà permette di **riutilizzare** codice senza riscriverlo, mantenendo funzionamento e modifiche **consistenti** in ogni sua istanza.

11.4 Overloading & Overriding

Overloading: metodi con lo stesso nome ma firme differenti;

Overriding: riscrittura di un metodo derivante da una superclasse;

11.5 Object

Ogni classe creata estende Object implicitamente ed in quanto tale, ogni classe Java eredita tutti i metodi e definizioni di Object. Alcuni metodi di Object sono: **equals**, **toString**, **getClass**, **istanceOf**.

Java Packages

12.1 Introduzione

Un package è una collezione di classi correlate e raggruppate sotto il nome del package.
I package permettono di organizzare le classi in modo logico ed evitare conflitti tra nomi in package diversi.

12.2 Visibilità

Le classi possono avere diverse visibilità, che ne influenzano l'accesso in base alla posizione nella cartella src:

Public: senza restrizioni;

Private: accessibili solo nella classe di definizione;

Protected: non accessibile all'esterno, fatta eccezione per le classi nello stesso package e le classi derivate;

Package-Wide: accessibile a tutte le classi nello stesso package;

Polimorfismo

13.1 Introduzione

Il **polimorfismo** è una caratteristica del codice Object Oriented che permette di riferirsi ad un oggetto dinamico tramite una reference.

13.2 Tipo Dinamico

Con **oggetto dinamico** si intende un oggetto la cui reference può avere tipo generalizzato. Il tipo dinamico è limitato solo alle gerarchie di tipi definite tramite ereditarietà.

13.3 Dynamic Binding

Il **Binding Dinamico** permette di definire e invocare un metodo dinamicamente, ovvero durante il runtime.

13.4 Casting

Come per i tipi primitivi, il casting permette di cambiare tipo ad un'istanza. Esistono 2 tipi di casting nella programmazione OOP:

Down-casting: aumento della specificità di una dichiarazione;

Up-casting: riduzione della specificità di una dichiarazione;

Classi Astratte & Interfacce

14.1 Classi Astratte

Una classe astratta è utilizzata per rappresentare un concetto non concreto ed in quanto tali, non possono essere istanziate.

Le classi astratte sono utili per imporre degli elementi comuni a tutte le sotto classi, senza la necessità che la loro superclasse debba essere istanziata o evocata.

14.2 Metodi astratti

Come la classe, anche i metodi astratti sono realizzati per essere adattati alle superclassi. I metodi astratti hanno la caratteristica peculiare di non avere il body.

14.3 Interfacce

Un'interfaccia è utilizzata per raggruppare l'insieme delle operazioni e dei dati pubblici di una classe.

Operazioni e dati privati non fanno parte dell'interfaccia, così come i suoi metodi.

Le interfacce si dichiarano con la parola chiave *interface* e si implementano in una classe utilizzando la congiunzione *implements*.

A differenza delle classi astratte, un'interfaccia definisce l'insieme dei metodi pubblici di una classe, indicando cosa fa, senza specificare il metodo di esecuzione.

Eccezioni

15.1 Introduzione

Molte procedure sono parziali, ovvero hanno un comportamento specifico solo per un sottoinsieme molto ristretto di argomenti.

Le eccezioni permettono di segnalare i casi non implementati gestendole e personalizzandole.

15.2 Throw

La parola chiave throw permette di lanciare l'eccezione seguita dalla keyword.
Questa termina l'esecuzione del metodo e propaga l'eccezione.

15.3 Try-Catch

Le eccezioni possono anche essere catturate e gestite tramite la struttura *try-catch*.
Tramite questa struttura è anche possibile legare più blocchi catch ad un solo try.

La struttura Try-Catch può avere un ramo *finally*, che viene eseguito qualunque sia il risultato dei primi rami.

15.4 Gestione delle Eccezioni

Le eccezioni possono essere gestite tramite 3 tecniche:

Masking: Viene gestita l'eccezione e l'esecuzione prosegue;

Forwarding: L'eccezione non può essere gestita, quindi si prosegue;

Re-Throwing: L'eccezione è catturata ma non può essere gestita, si prosegue generando un'eccezione differente;