

G

T

G

T

5

E

5

N

O

N

O

+

O

S

O

O

G

T

G

T

9

5

E

5

2

N

O

N

O

O

S

O

G

T

G

T

5

E

5

PROG 2

ASTRAZIONI STRUTTURALI

Degli oggetti che condividono **caratteristiche in comune** possono essere

classificati: vengono quindi accumulati da una **classe**

Il processo per il quale definisco caratteristiche generali condivise dagli oggetti è detto **ASTRAZIONE**

→ un oggetto di una classe è detta **"ISTANZA"** della classe

LA CLASSE quindi è una sorta di **TEMPLATE** dell'oggetto

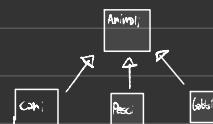
TIPI DI ASTRAZIONE

→ **CLASSIFICAZIONE**: crea la classe } **instance of**: controlla se un oggetto è un'istanza di una classe

- * lega le istanze alle classi
- * definisce le caratteristiche comuni tra gli oggetti
- * ogni oggetto ha quelle determinate proprietà della classe

→ **GENERALIZZAZIONE**: definisco una **superclasse** → i S-A

- * lega una classe genitore ad una o più classi figlie (sottoclasse) che ne sono ereditarie } esempio: Lupo è un cane, un cane è un mammifero (superclasse)
- * ogni classe figlia ha le proprietà del padre
- * ogni classe figlia appartiene anche alla superclasse (padre)



→ **AGGREGAZIONE**: definisco un'appartenenza ad una classe → **part-of**

- * "Aggregato" (classe) fa parte di classi "parti"
- * ogni esemplare di "aggregato" è costituito da esemplari di "parti"

* ESEMPIO:

Un automobile comprende RUOTE, MOTORI, ...

→ **ASSOCIAZIONE**: connessione logica fra oggetti di una classe e oggetti di un'altra classe → **has-a**

* esempio

Mario è padrone di Fido

RAPPRESENTAZIONE

UML

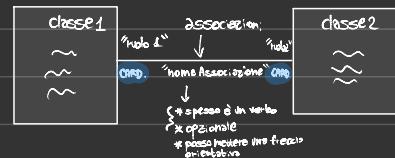
classe:

nome classe
~~ attributi
~~ operazioni

oggetto:



associazioni:



CARDINALITÀ ASSOCIAZIONE

Mi dice con quante istanze posso mettere in relazione con altre dell'altra classe

ATTENZIONE

Nell'UML la cardinalità segue il verso delle frecce dell'associazione



CARDINALITÀ DELLE ASSOCIAZIONI TRA Istanze:

0, 1

0 = partecipazione opzionale

1 = partecipazione in un'istanza

$1, *$ / associazione verso più istanze
ogni istanza è associata

ASSOCIAZIONE MULTPLA: più associazioni tra copie di classi



Tipi di relazione:

→ **AGGREGAZIONE**: le parti dell'associazione possono esistere → esempio: una stampante nata anche senza un computer
(poco forte) ↓ indipendentemente dal TUTTO, essa in alcuni casi può comunque esistere indipendentemente
diamente nato → alcune parti possono essere condivise tra aggregati



→ **COMPOSIZIONE**: le parti dipendono strettamente dal loro TUTTO che è unico
(molto forte) ↓ (solo uno per volta), sono gestite da lui; si può cedere la proprietà ad un altro TUTTO
diamente piano ↓ esempio: il motore appartiene ad una e una sola auto



Se il tutto viene distrutto devono essere distrutte anche le parti o cedute ad un altro oggetto

OGGETTO

Possiede 3 caratteristiche:

- * **Stato**: una delle possibili condizioni in cui può trovarsi, è dato dall'insieme dei suoi attributi
- * **comportamento**: come risponde alle richieste degli oggetti (metodi), definito dalle loro interface
- * **identità**: due oggetti anche se si trovano nello stesso stato sono comunque distinti → codice univoco [OID]

CLASSE IN JAVA

La classe è la rappresentazione di un oggetto: gli oggetti di una classe condividono proprietà e comportamenti, poiché la classe definisce le caratteristiche di ogni istanza

Interfaccia di una classe:

- Definisce comandi: nome, return e parametri formal
- Implementa: attributi, metodi interni e operazioni

INFORMATION HIDING

Sintassi
modificatore di visibilità [alti modificatori] class NomeClasse {

Caratteristica delle classi che permette di impostare visibilità dei suoi elementi, rendendoli invocabili dall'esterno o invisibili (quindi gioca sul private/public)

// Def. attributi di istanza e/o di classe

// Implementazione costruttore:

// Implementazione Metodi di istanza e/o di classe

GESTIONE DELLA MEMORIA

- Le dichiarazioni delle variabili sono salvate nello stack, hanno valore undefined
- le assegnazioni (valori) sono nella heap
- L'oggetto è dichiarato allo stesso modo, ma i valori sono quelli standard (int=0, char="")

Operatore di confronto (==):

- Variabili = controllo ugualanza contenuto
- Oggetti = controllo ugualanza celle di memoria

DOT NOTATION

attributo di un oggetto

oggetto.attributo

DEBUG

Tool che permette di controllare il codice



gli IDE permettono la selezione di breakpoint, punti in cui il sistema si ferma

permettendo di controllare un'istruzione alla volta



questo metodo consente di vedere identificativi e attributi in tempo reale (quindi tutti i loro stati)

ARRAY DI OGGETTI

Oggetto [] nomeArray;

contengono le referenze agli oggetti in questione (sono memorizzati gli indirizzi di memoria)

↓
array di reference

il valore default da inizializzazione è null.

PT. 2 - PARZIALE

EREDITARIETÀ → processo attraverso cui una classe (derivata) viene creata a partire dal codice di un'altra classe (base)

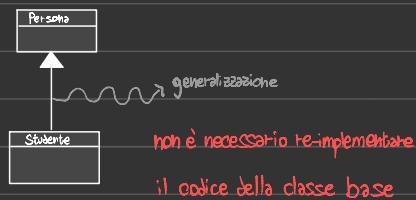
Per evitare attributi ripetuti possiamo usare il modello is-a

↓ con quale costruttore?

Extend → sottoclasse "eredita" il codice della classe

Studente extends Persona

↓ nel costruttore bisognerà richiamare il costruttore superiore



Garantisce inoltre consistenza (se la superclasse cambia automaticamente si aggiornano ed estendibili) (si possono implementare tante sottoclassi gradualmente e senza riscrivere)

OVERRIDING

Si possono pensare a comportamenti diversi per metodi della superclasse in base

alla sottoclasse grazie all'uso dell'override (esempio: `toString()` di `Persona` stampa anagrafica, override di `toString()` `Studente`

↓
② **override** { * permette al compilatore di verificare la correttezza dell'override
* migliora documentazione codice

final impedisce l'override

N.B!! L'override sono diversi metodi con stesso nome ma forme diverse
→ OVERRIDE riaccende il comportamento di un metodo

COSTRUTTORI

La sottoclasse deve richiamare nel suo costruttore il costruttore base, con tutti gli attributi base + attributi dell'eredità

↓

`super();` se omissa la invocazione è implicita senza parametri

CLASSE OBJECT

È una generalizzazione di classi, qualunque cosa sia (ogni classe Java eredita definizioni e metodi di `Object`)

Metodi di `Object`:

* `equals` (restituisce true se `this==obj`)

* `toString` (`nameclasse@hashcode`)

VISIBILITÀ TRA CLASSI: limitata allo stesso package
↓ estendibile tramite import di classi

PACKAGES

"collezione di classi correlate a cui viene assegnato un nome"

→ consente un'organizzazione di classi

→ evita conflitti tra i nomi delle classi (due classi con stesso nome possono esistere in 2 package diversi)

VISIBILITÀ ATTRIBUTI/METODI

* `public`

* `private`: solo alla classe

* `protected`: solo nello stesso package o sottoclasse

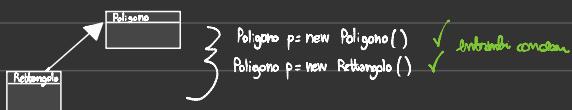
* `package-wide`: solo package

POLIMORFISMO

Proprietà del codice di comportarsi in modo diverso in base al contesto di esecuzione



realizzato tramite il concetto di superclasse e di eredi (reference alle sotto classi dinamica)



DYNAMIC BINDING: il vincolo è che un oggetto x può essere di tipo y oppure di un suo sottotipo (o antitipo) y.
(o late binding)

il legame tra definizione e invocazione di un metodo è dinamico, avviene in run-time e non prima:

il compilatore di Java riconosce il tipo effettivo ed esegue il metodo relativo

IL BINDING DINAMICO NON SI APPLICA PER:

* metodi statici: - metodi associati a classi, non ad oggetti, quindi la loro invocazione non avviene per reference
Figurateometrica numeroFigureCreate()

* metodi final: non possono essere riferimenti nelle sottoclassi

* Overloading: il binding è statico sulla firma dei metodi; è dinamico solo nella scelta dei tipi di una gerarchia (overriding)

COME VENGONO RISOLTI OVERLOADING E OVERRIDE

→ OVERLOADING (compile-time): decisa in compilazione in base all'argomento quale metodo con firma compatibile sceglie

→ OVERRIDE (run-time): riferito tipo dinamico e segue la procedura in runtime

CASTING

↑ verso l'alto: sicuro e funzionante, la sottoclasse contiene ogni caratteristica della super

↓ verso il basso: non sicuro e potenzialmente non funzionante, responsabile il programmatore

{
ERROR! RUN-TIME: cast riscontrato da un metodo non previsto in runtime
ERROR! COMPILE-TIME: gerarchia non ragionevole (tipi incompatibili)}

OBJECT: metodo instanceof (boolean)

Verifica se una reference è instance di una classe x

OBJECT consente sia up che down casting senza problemi

OBJECT: metodo getClass()

→ final, restituisce il tipo dinamico dell'oggetto (confronto con == e !=)

