UML: Class diagram

Gianluca Pironato

Class diagram in UML

- È un diagramma utile a osservare alcuni aspetti strutturali, quindi statici, del sistema;
- In particolare, ci permette di cogliere al volo **classi e relazioni** essenziali del contesto che stiamo trattando;
- Il codice del SW in esame riflette ovviamente questa organizzazione a oggetti.

Elementi fondamentali

- **Classe**: rappresenta un'entità, raffigurata da un rettangolo diviso in tre sezioni.
  - sezione superiore: nome della classe (solitamente in maiuscolo o in corsivo se astratta)
  - sezione centrale: attributi (campi, variabili, proprietà)
  - sezione inferiore: metodi (operazioni, funzioni)
- Ogni istanza della classe è un oggetto.

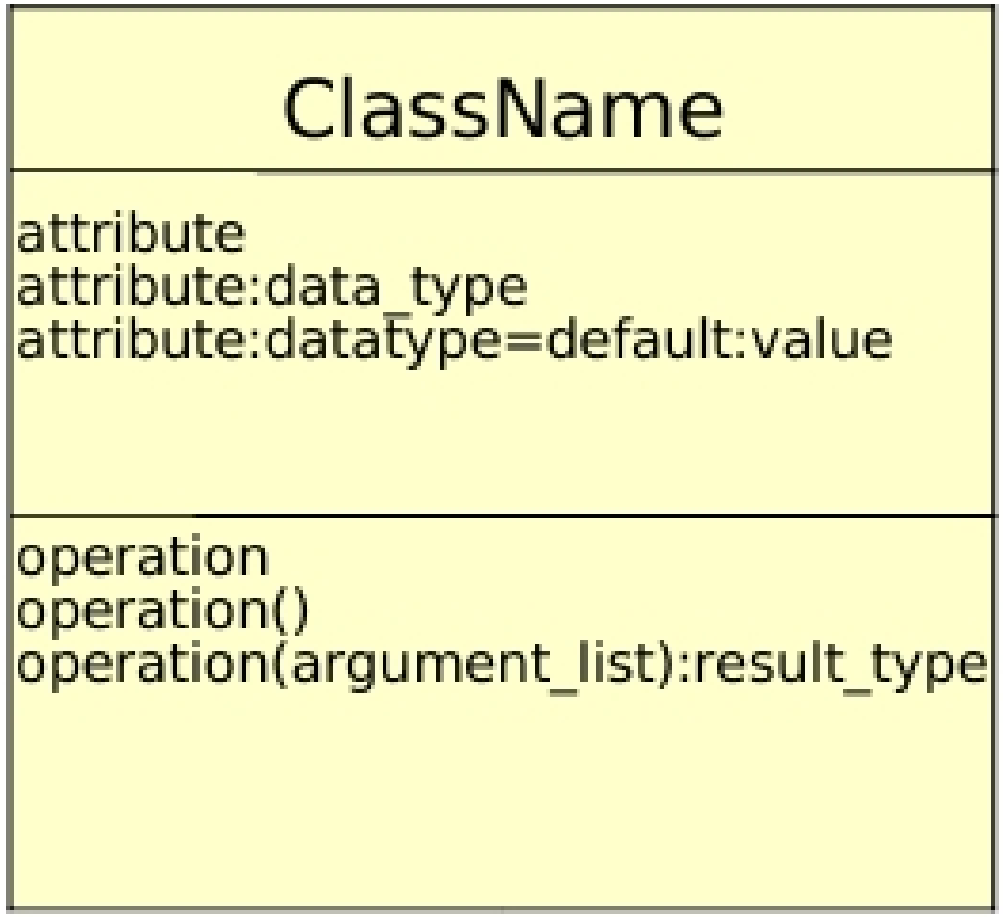
Elementi fondamentali

- **Attributi**: sono le proprietà significative che descrivono un'istanza di una classe; si indica dopo **":"** il tipo di variabile;
  - es. per una classe `Animal` si possono avere attributi come `name` (stringa), `id` (intero), `age` (intero).
- **Metodi**: azioni che descrivono il comportamento della classe; si indica dopo **":"** il tipo di valore restituito (se presente);
  - es. per `Animal` si possono definire metodi come `setName()` o `eat()`;
  - tra le tonde, posso indicare anche un parametro e dopo **":"** il tipo di questo.
- **N.B. sia attributi che metodi vengono preceduti da un simbolo di visibilità**.

Visibilità

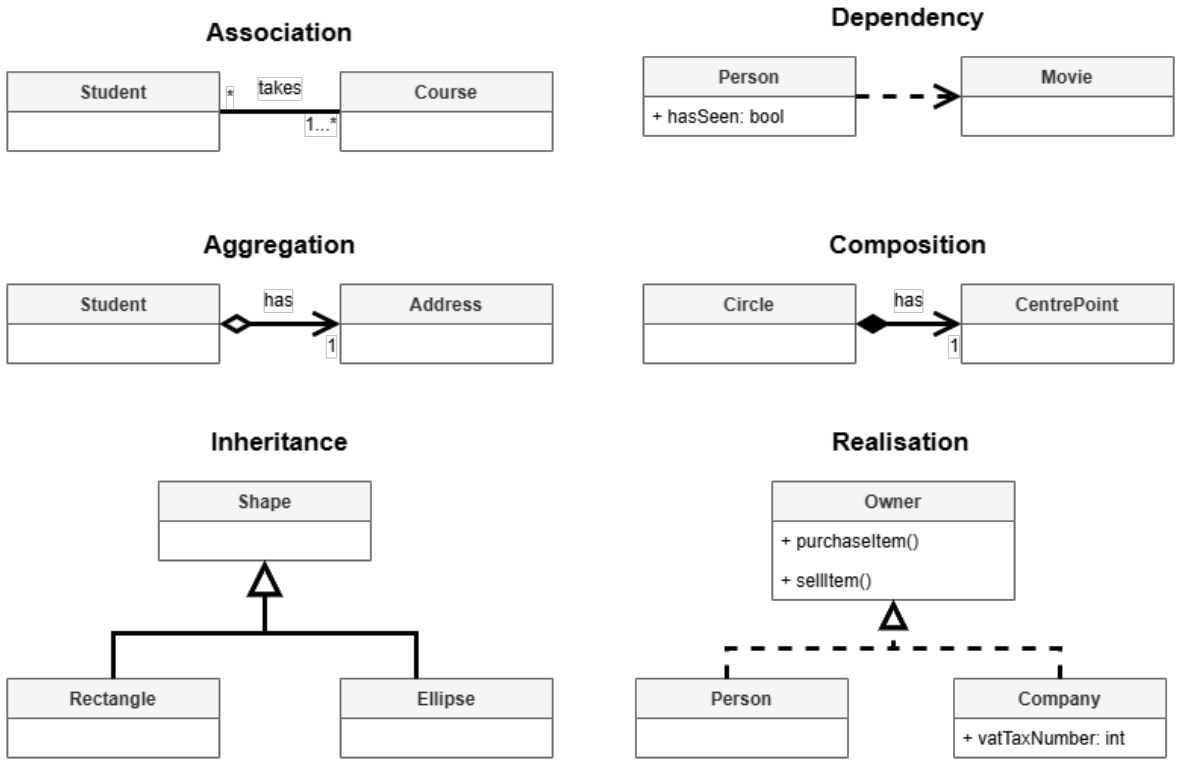
- La visibilità determina l'accesso ad attributi e metodi:
  - \- indica **private**: accessibile solo all'interno della classe;
  - \+ indica **public**: accessibile da qualsiasi altra classe;
  - \# indica **protected**: accessibile dalla classe e dalle sue sottoclassi;
  - ~ indica **package**: accessibile dalle classi dello stesso package.
- in genere, gli attributi sono dichiarati `private` o `protected`, mentre i metodi sono `public`;
- inoltre, per l'accesso agli attributi è bene predisporre un'interfaccia di metodi setter e getter.

Notazione per classe



Elementi fondamentali - prosegue

- **Relazioni tra classi**: associazione, aggregazione, composizione, dipendenza, ereditarietà, implementazione, ecc.
- Ciascuna con notazione grafica diversa.

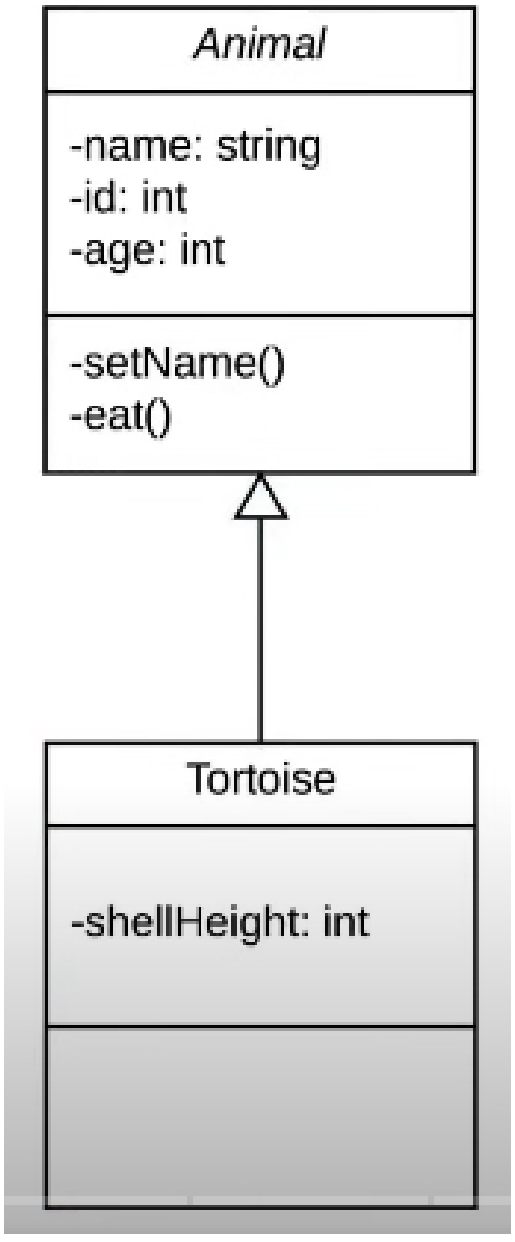


Ereditarietà

- Permette di definire una relazione tra una classe generica (superclasse, genitore) e classi specializzate (sottoclassi, figli).
- I figli ereditano attributi e metodi dal genitore.
- Si rappresenta con una freccia con "testa vuota" che punta dalla sottoclasse alla superclasse.
- es. le classi `Cat` e `Dog` possono ereditare dalla classe `Animal`.
- Una classe figlio, essendo più specifica, deve obbligatoriamente avere qualcosa in più rispetto alla classe genitore.

N.B. se il nome della classe genitore è scritto in corsivo, allora questa è da considerarsi una classe astratta (non viene mai direttamente istanziata nel codice).

Ereditarietà



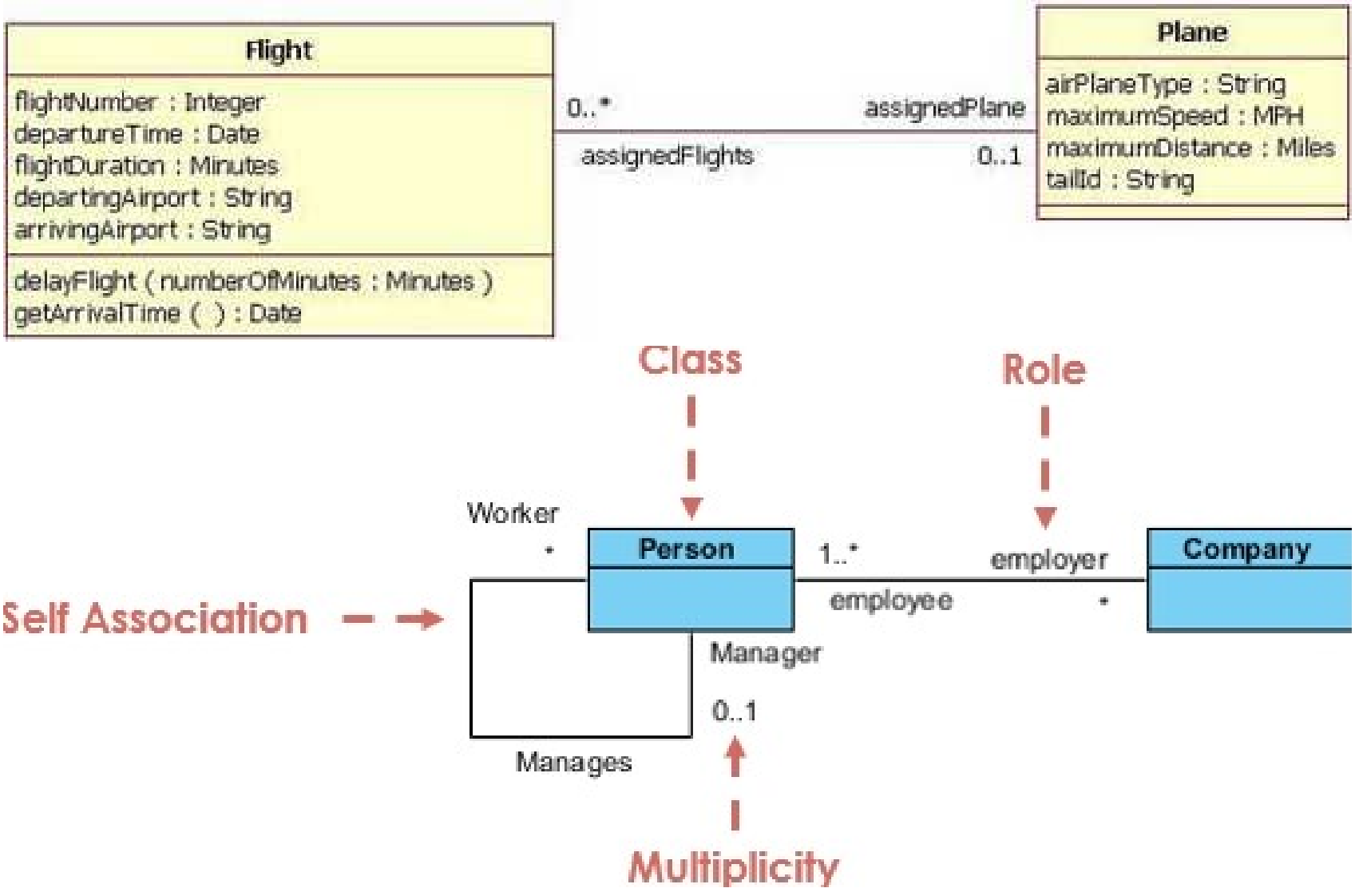
Associazione

- Permette di definire una relazione semplice tra due classi.
- es. `Cat` mangia `Mouse`
- Si rappresenta con una linea accompagnata da un verbo e/o dal ruolo e dalla cardinalità minima e massima (o molteplicità) su ogni lato.
- In genere, sono associazioni tutte le relazioni che non rendono una classe dipendente dall'altra.

Molteplicità

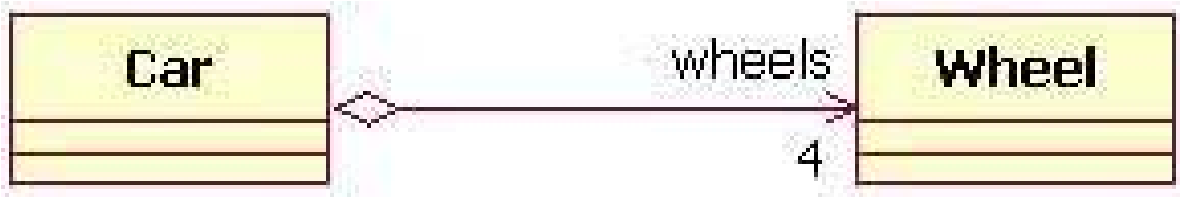
- Specifica quanti oggetti di una classe possono essere associati a un oggetto di un'altra classe.
- esempi:
  - 1: esiste esattamente un oggetto (1 a 1);
  - 0..1: opzionale, al massimo uno;
  - 1..*: almeno uno, ma possono essercene molti;
  - 0..*: opzionale e possono essercene molti.
- La molteplicità viene letta e interpretata in base al ruolo scritto vicino.

Associazione



Aggregazione

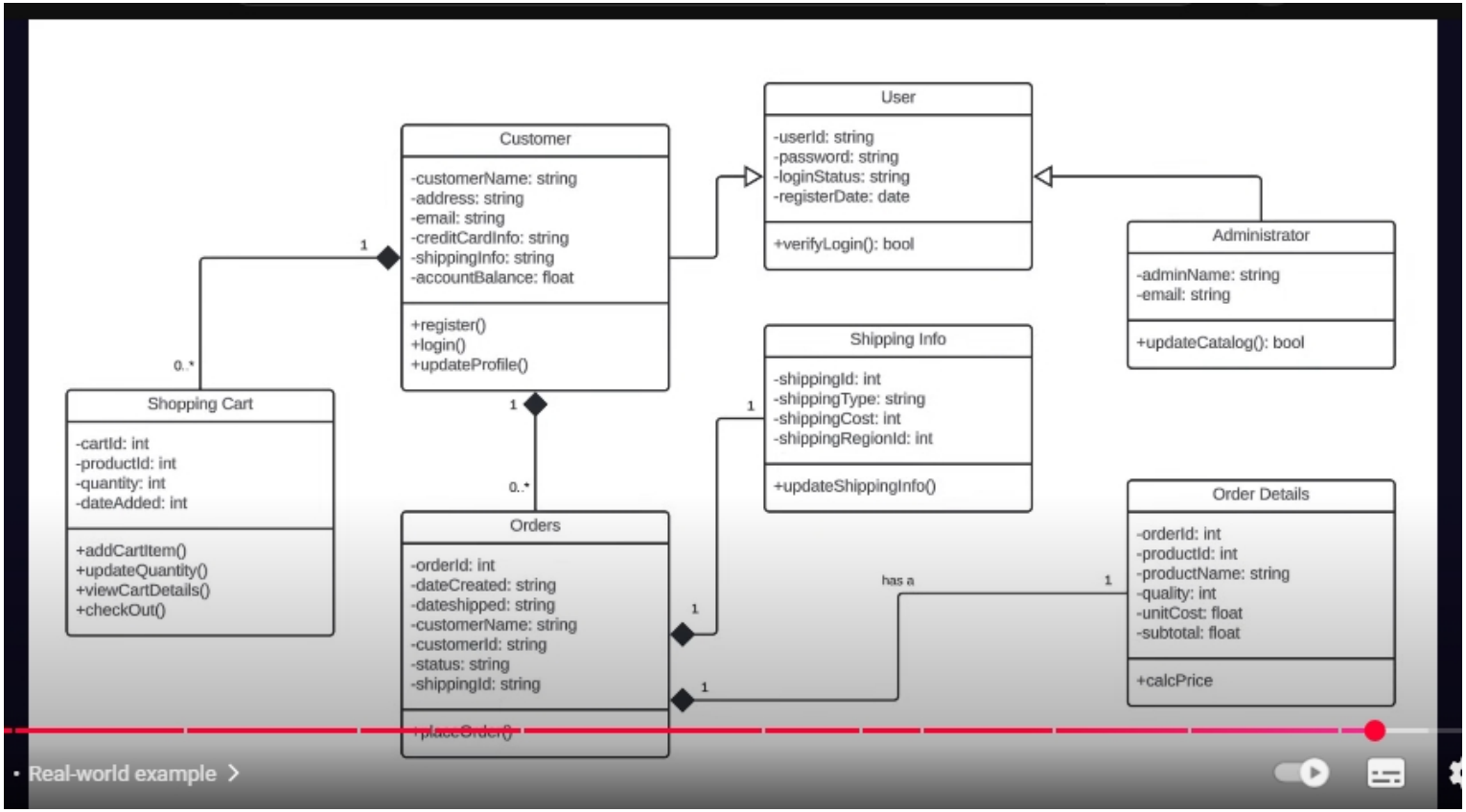
- Trattasi di un tipo di associazione utile a definire un'entità e le sue parti.
- **L'aggregazione delle parti non è però essenziale per la loro esistenza**; ovvero, le parti possono esistere indipendentemente.
- Si rappresenta con una linea che culmina in un rombo vuoto sul lato della classe "aggregante".



Composizione

- Trattasi di una forma più forte di aggregazione.
- **La composizione delle parti è essenziale per la loro esistenza**; ovvero, le parti non possono esistere indipendentemente.
- Si rappresenta con una linea che culmina in un rombo pieno sul lato della classe "composta".

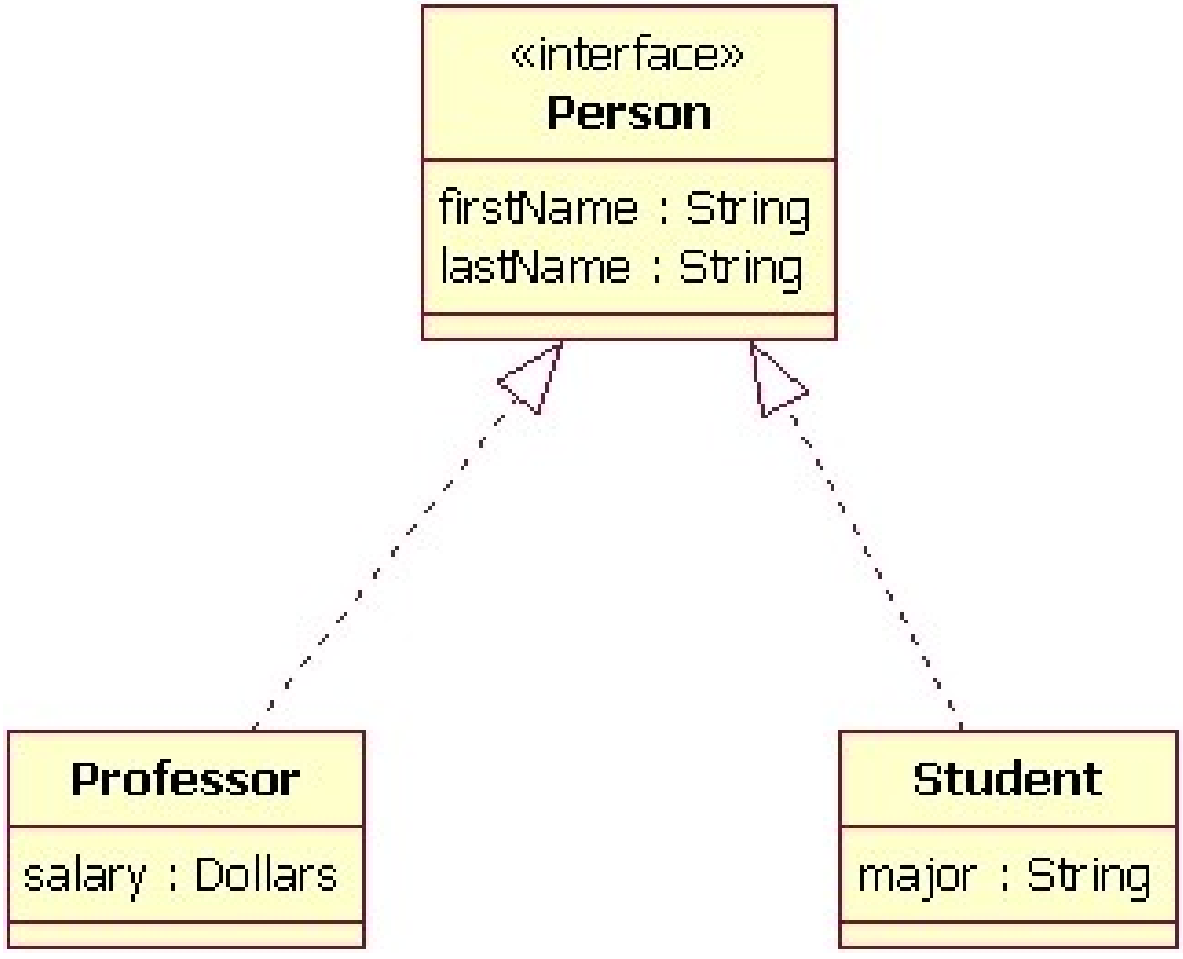
Esempio riassuntivo



Realizzazione

- È una relazione che sussiste tra un'interfaccia e le classi che la implementano.
- Da tenere presente sempre che un'interfaccia non può essere direttamente istanziata,
- Solitamente si utilizza per tenere traccia di classi che assumono "ruoli" descritti, appunto, nell'interfaccia (che racchiude dichiarazioni di attributi e di metodi, poi definiti in concreto dalle classi in relazione).
- Si rappresenta con una linea tratteggiata con culmina con una punta vuota.

Realizzazione



Dipendenza

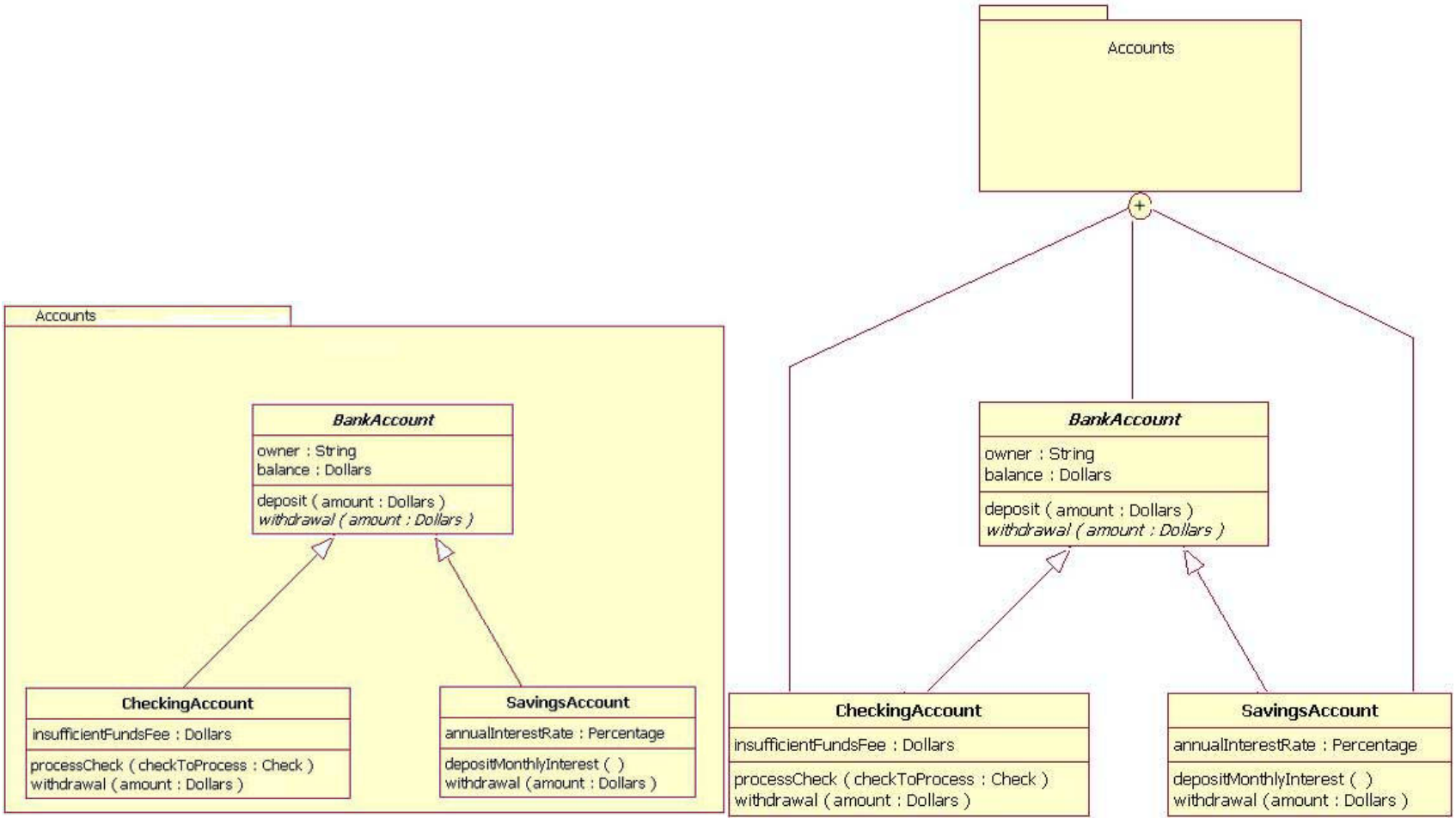
- Si presta a modellare relazioni con un parte client e una parte supplier, in cui la prima viene influenzata da modifiche operate sulla seconda.
- Un esempio calzante è quello del carrello su un sito di e-commerce: l'importo totale da pagare varia se il prezzo di un singolo prodotto aumenta.
- Si rappresenta con una freccia tratteggiata che punta dalla classe dipendente alla classe le cui "variazioni" hanno ricadute su terzi.

Package

Terminata la panoramica riguardante le classi, è bene dedicarsi brevemente ai package previsti in UML.

- Quando si modella un sistema complesso, composto da molte classi e perciò difficile da gestire, si può optare per un elemento organizzativo chiamato package.
- Questo raggruppa le classi in namespace, simili a cartelle in un sistema di archiviazione, semplificando la comprensione del modello e favorendone la suddivisione in parti più specifiche e tematiche.
- Ci sono due modi di rappresentare i package e la scelta dipende unicamente da questioni di leggibilità.

Package


Accounts
BankAccount
owner : String
balance : Dollars
deposit (amount : Dollars)
withdrawal (amount : Dollars)
CheckingAccount
insufficientFundsFee : Dollars
processCheck (checkToProcess : Check)
withdrawal (amount : Dollars)
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal (amount : Dollars)
Accounts
BankAccount
owner : String
balance : Dollars
deposit (amount : Dollars)
withdrawal (amount : Dollars)
CheckingAccount
insufficientFundsFee : Dollars
processCheck (checkToProcess : Check)
withdrawal (amount : Dollars)
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal (amount : Dollars)

Conclusioni

- Le basi del Class diagram sono sufficienti ad acquisire la corretta chiave di lettura per tutti i cosiddetti *structure diagram*, rappresentanti gli aspetti statici del sistema modellato.
- L'utilità del Class diagram è vasta: dal design alla documentazione di codice, così come per fare design validation o per comunicare con aree e settori privi di conoscenze di programmazione.
- Al di là delle classificazioni, è facile cogliere la centralità di questo diagramma per tutto UML: anche i sequence e use-case diagram offrono viste assolutamente cariche di riferimenti alle classi modellate mediante un Class diagram.

Fonti

- UML 2 Class diagram, `https://developer.ibm.com/articles/the-class-diagram/`