Diagrammi Delle Classi

Ingegneria Del Software

Università degli Studi di Padova
Dipartimento di Matematica
Corso di Laurea in Informatica

rcardin@math.unipd.it

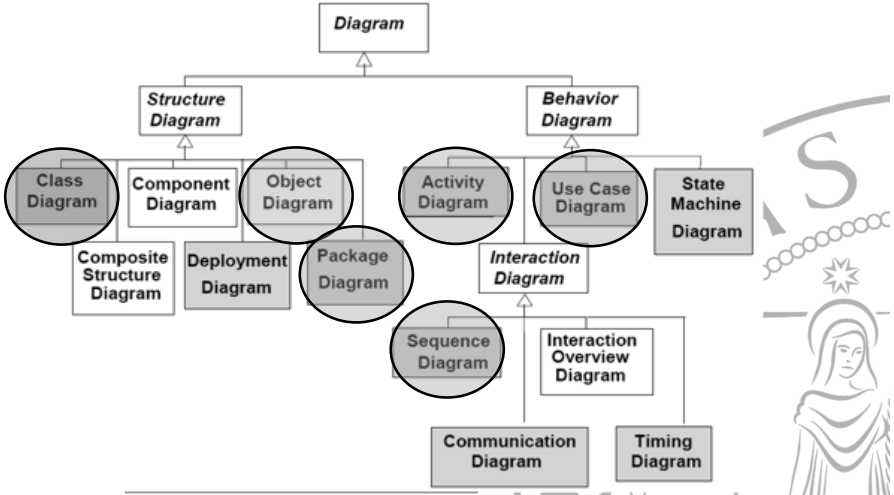
Sommario

- Introduzione
- Proprietà e Operazioni
- Concetti base e avanzati
- Diagrammi degli oggetti

Sommario

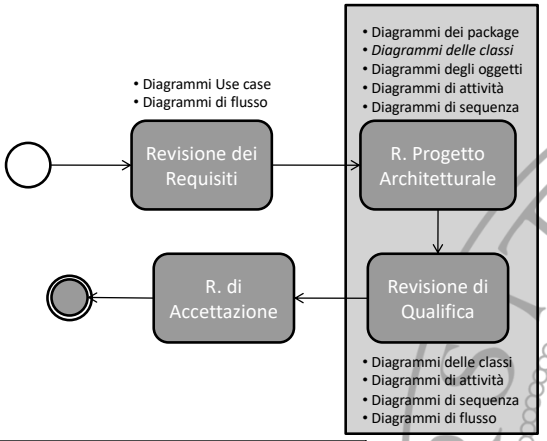
- Introduzione
- Proprietà e Operazioni
- Concetti base e avanzati
- Diagrammi degli oggetti

Diagrammi Delle Classi



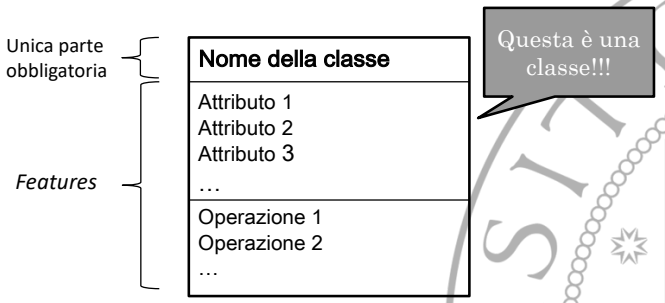
Diagrammi Delle Classi

- Specifica Tecnica, Definizione di Prodotto



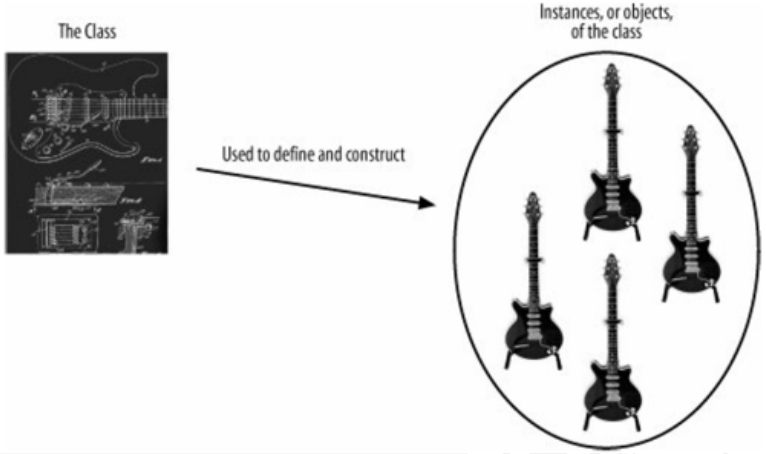
Diagrammi Delle Classi

- Definizione
  - Descrizione del tipo degli oggetti che fa parte di un sistema
    - Relazioni statiche fra i tipi degli oggetti



Diagrammi Delle Classi

- Definizione: esempio



Diagrammi Delle Classi

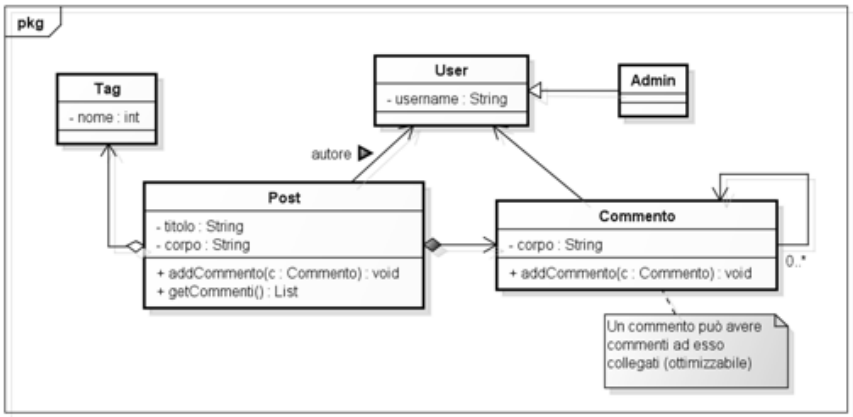
- Esempio principale

Esempio

È richiesto lo sviluppo di un'applicazione che permetta la gestione di un semplice blog. In particolare devono essere disponibili almeno tutte le funzionalità base di un blog: deve essere possibile per un utente inserire un nuovo post e successivamente per gli altri utenti deve essere possibile commentarlo. Queste due operazioni devono essere disponibili unicamente agli utenti registrati all'interno del sistema. La registrazione avviene scegliendo una username e una password. La username deve essere univoca all'interno del sistema.

Diagrammi Delle Classi

- Esempio principale



Sommario

- Introduzione
- Proprietà e Operazioni
- Concetti base e avanzati
- Diagrammi degli oggetti

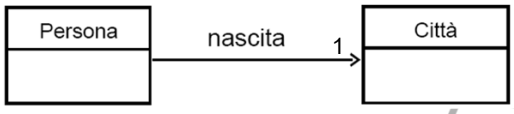
Proprietà

- Caratteristiche strutturali
  - Attributo

Definizione

```
Visibilità nome : tipo [molteplicità] = default {proprietà aggiuntive}
```

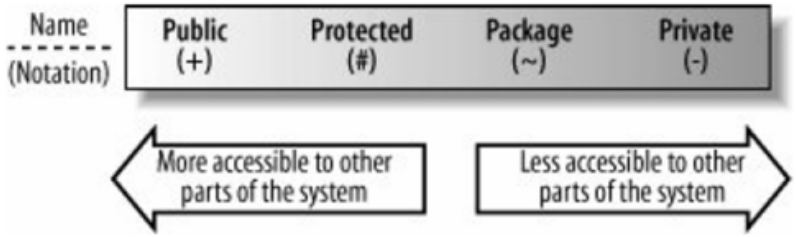
    - Visibilità: **+** pubblica, **-** privata, **#** protetta
  - Associazione
    - Linea continua e orientata fra due classi



    - Molteplicità
      - Quanti oggetti possono far parte dell'associazione
        - 1, 0..1, 0..*, *,…
    - Spesso interscambiabile con un attributo: quando usarla?

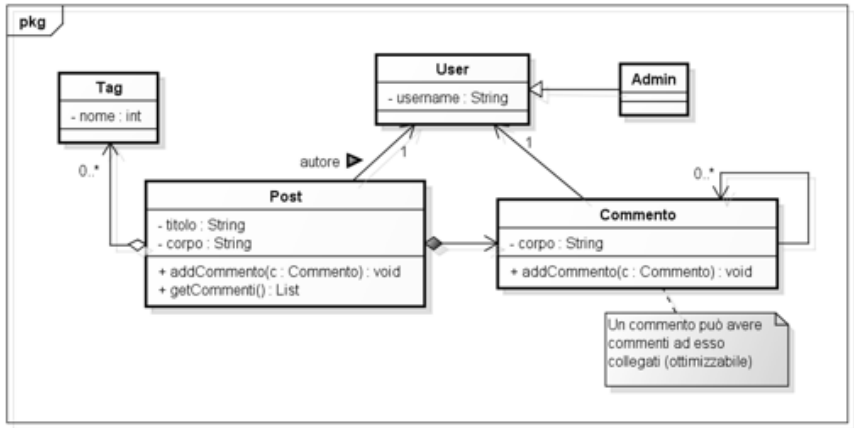
Proprietà

- Visibilità



PROPRIETÀ

- Esempio 1

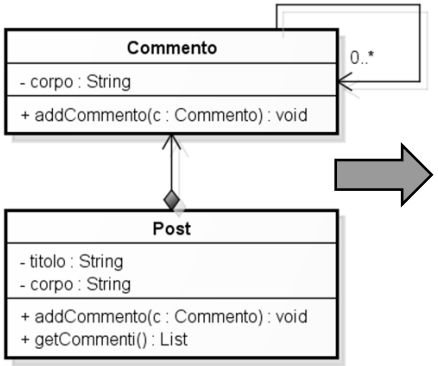


PROPRIETÀ

- …nel linguaggio di programmazione
  - Attributi
    - Membri di classe (privati, se possibile)
    - Proprietà aggiuntive
      - Se *ordered*: Array o vettori
      - Se *unordered*: insiemi
    - Convenzioni dei gruppi di programmazione
      - Esempio: *Getter* e *setter* per ogni attributo
  - Associazioni
    - Anche se etichettata con verbo, meglio renderla con un nome
    - Evitare le associazioni bidirezionali
      - Di chi è la responsabilità di aggiornare la relazione?

PROPRIETÀ

- Esempio 2



Java

```
public class Commento {
  private String corpo = null;
  private List<Commento> commenti =
    new ArrayList<Commento>();

  public void addCommento(Commento c) {
    commenti.add(c);
  }
}

public class Post {
  private List<Commento> commenti =
    new ArrayList<Commento>();

  // ...

  public List<Commento> getCommenti()
    {...}
}
```


OPERAZIONI

- Le azioni che la classe "sa eseguire"
  - Aspetto comportamentale
  - Servizio che può essere richiesto ad ogni istanza della classe

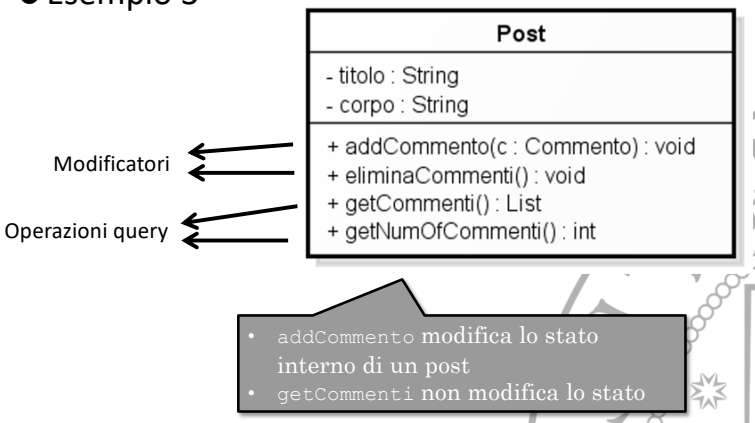
Definizione

```
Visibilità nome (lista-parametri) : tipo-ritorno {proprietà aggiuntive}
Lista-parametri := direzione nome : tipo = default
```

    - Direzione: **in**, **out**, **inout** (*default* **in**)
    - Visibilità: **+** pubblica, **-** privata, **#** protetta
  - *Query*
  - Modificatori
  - Operazione ≠ metodo
    - Concetti differenti in presenza di polimorfismo

Operazioni

- Esempio 3

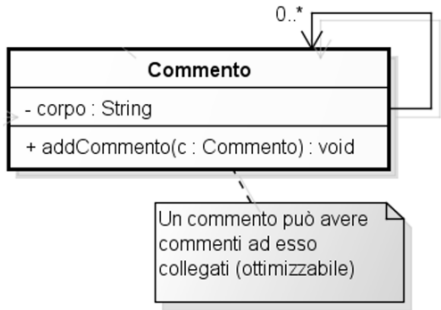


Sommario

- Introduzione
- Proprietà e Operazioni
- Concetti base e avanzati
- Diagrammi degli oggetti

Commenti e Note

- Informazioni aggiuntive
  - Singole e solitarie
  - Legate a qualsiasi elemento grafico
    - Linea tratteggiata
  - Esempio 5



Relazione di Dipendenza

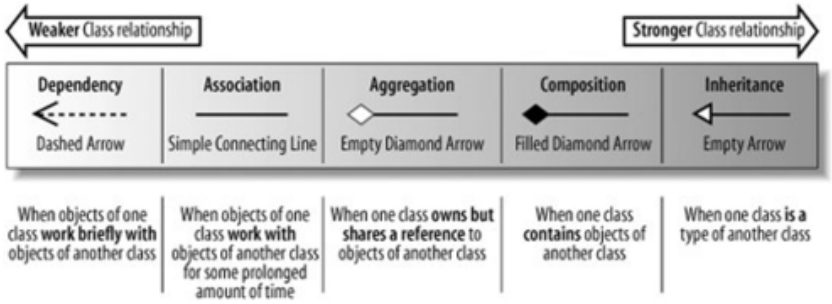
- Definizione

> Si ha dipendenza tra due elementi di un diagramma se la modifica alla definizione del primo (fornitore) può cambiare la definizione del secondo (client)

- UML permettere di modellare ogni sorta di dipendenza
  - Non è una proprietà transitiva!
- Le dipendenze vanno minimizzate!
  - *Loose coupling*
- Da inserire solo quando danno valore aggiunto
  - Troppe dipendenze creano confusione nel diagramma

Relazione di Dipendenza

- Definizione



Relazione di Dipendenza

- Definizione

Maggiore è la quantità di codice condiviso fra due tipi, maggiore è la dipendenza fra essi.

- La dipendenza tra due tipi è direttamente proporzionale alla probabilità di modificare entrambi

$$\delta_{A \to B} \propto P(B_{mod} | A_{mod})$$

- La dipendenza è quindi una funzione di numero SLOC condivise e di ampiezza dello scope del codice condiviso

$$\delta_{A \to B} = \frac{\varphi_{S_{A|B}}}{\varphi_{S_{tot_B}}} \varepsilon_{A \to B}$$

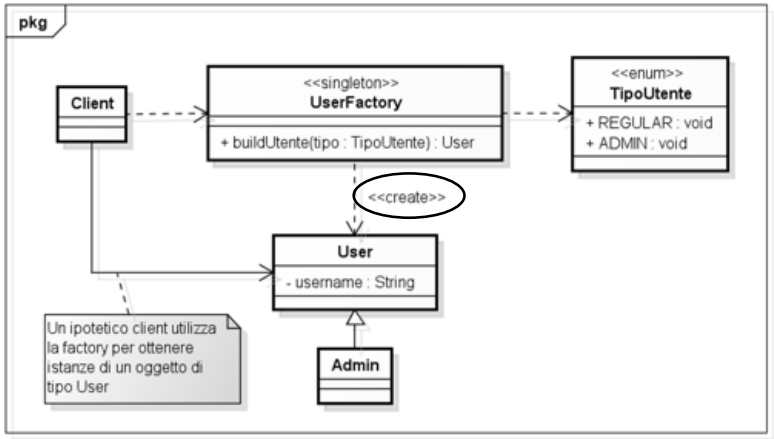
Relazione di Dipendenza

- Dipendenze UML

Parola chiave	Significato
«call»	La sorgente invoca un'operazione della classe destinazione.
«create»	La sorgente crea istanze della classe destinazione.
«derive»	La sorgente è derivata dalla classe destinazione
«instantiate»	La sorgente è una istanza della classe destinazione (meta-classe)
«permit»	La classe destinazione permette alla sorgente di accedere ai suoi campi privati.
«realize»	La sorgente è un'implementazione di una specifica o di una interfaccia definita dalla sorgente
«refine»	Raffinamento tra differenti livelli semantici.
«substitute»	La sorgente è sostituibile alla destinazione.
«trace»	Tiene traccia dei requisiti o di come i cambiamenti di una parte di modello si colleghino ad altre
«use»	La sorgente richiede la destinazione per la sua implementazione.

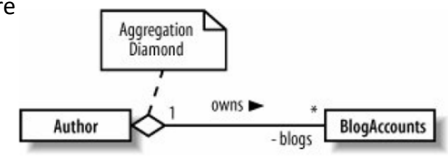
Relazione di Dipendenza

- Esempio 6

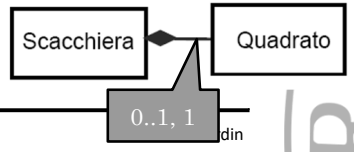


Aggregazione e Composizione

- Aggregazione
  - Relazione di tipo "parte di" *(part of)*
    - Gli aggregati possono essere condivisi

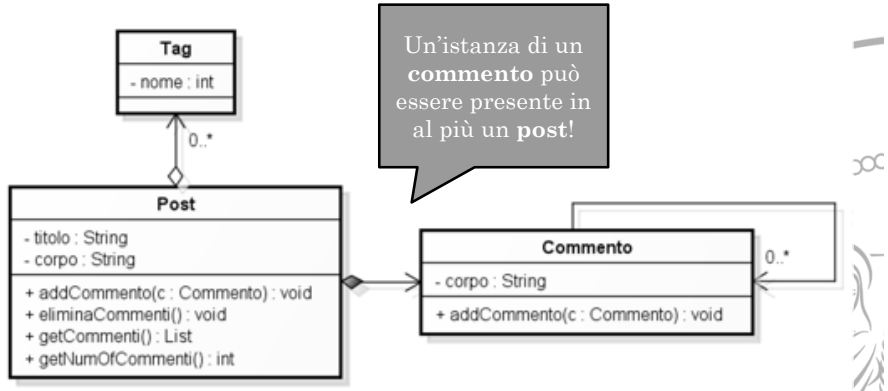


- Composizione
  - Come aggregazione, ma:
    - Gli aggregati appartengono ad un solo aggregato
    - Solo l'oggetto intero può creare e distruggere le sue parti



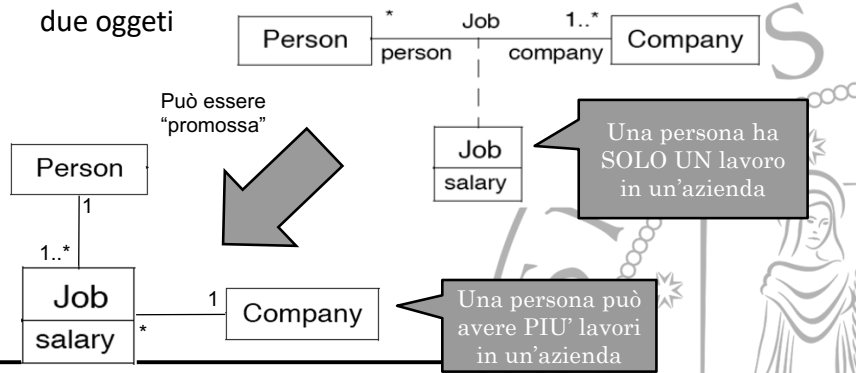
Aggregazione e Composizione

- Esempio 7



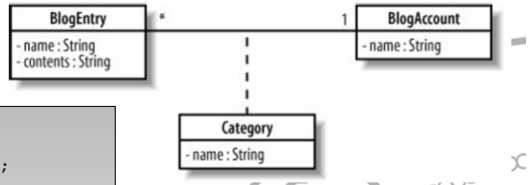
Classi di Associazione

- Aggiungono attributi e operazioni alle associazioni
  - Esiste solo una istanza della classe associazione fra i due oggetti



Classi di Associazione

- Traduzione in linguaggio di programmazione



Java

```
public class BlogAccount {
   private String name = null;
   private Category[] categories;
   private BlogEntry[] entries;
}

public class Category {
   private String name;
}

public class BlogEntry {
   private String name;
   private Category[] categories
}
```

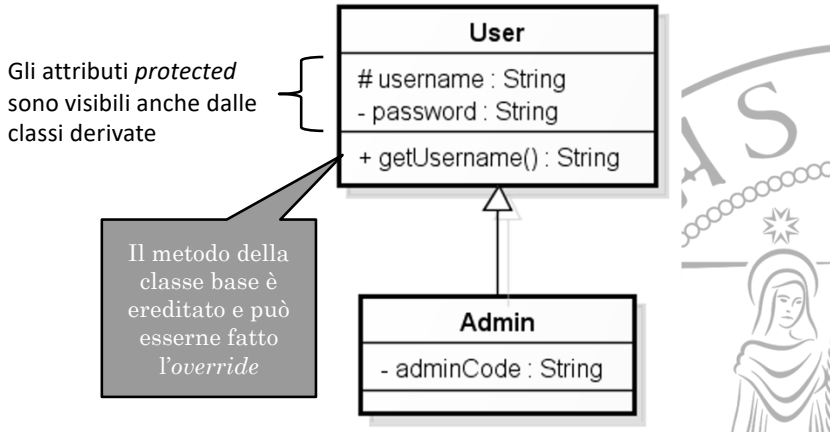
Ovviamente, è necessario imporre che esista solo un'istanza

GENERALIZZAZIONE

- A generalizza B, se ogni oggetto di B è anche un oggetto di A
  - Equivale all'ereditarietà dei linguaggi di programmazione
    - Ereditarietà multipla supportata, ma da **NON USARE**!
  - Le proprietà della superclasse non si riportano nel diagramma della sottoclasse
    - A meno di *override*
  - Sostituibilità
    - Sottotipo ≠ sottoclasse
      - Interfacce / implementazione

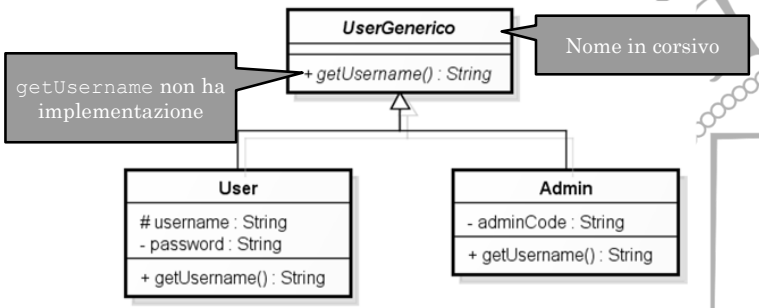
GENERALIZZAZIONE

- Esempio 4



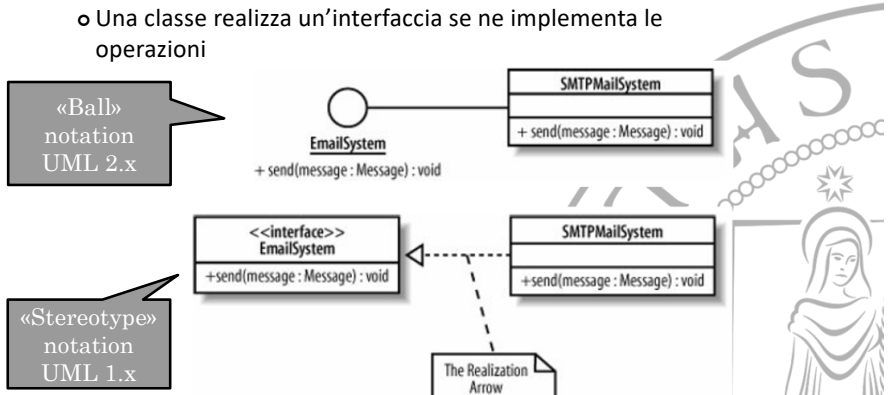
CLASSI ASTRATTE

- Classe Astratta `{abstract}`
  - Classe che non può essere istanziata
    - Operazione astratta non ha implementazione
    - Altre operazioni possono avere implementazione



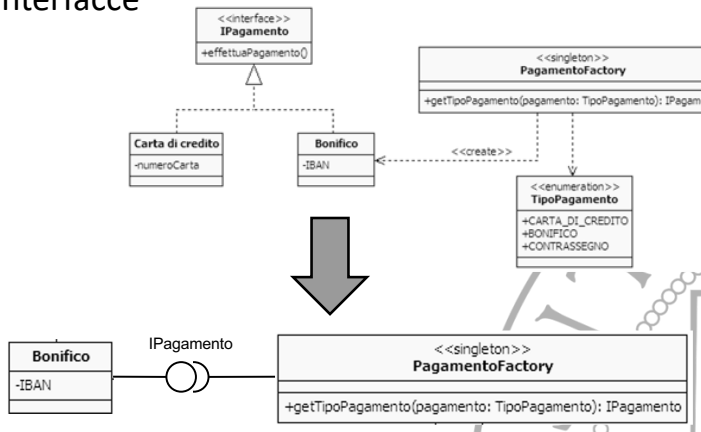
INTERFACCE

- Interfaccia `«interface»`
  - Classe priva di implementazione
    - Una classe realizza un'interfaccia se ne implementa le operazioni



Interfacce

- Interfacce



Classificazione e Generalizzazione

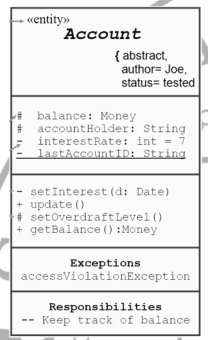
- Sottotipo ≠ "è un" (IS A)

Generalizzazione	Classificazione
• *Un Border Collie è un cane* • *I cani sono animali* • *I cani sono una specie*	• *Shep è un Border Collie* • *Border Collie è una razza*

- Generalizzazione
  - Proprietà transitiva
    - La classificazione non lo è!
- Classificazione
  - Dipendenza «`instantiate`»

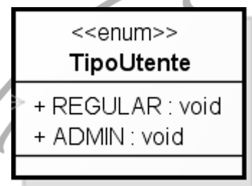
Caratteristiche Varie

- Operazioni e attributi statici
  - Applicabili alla classe, non all'oggetto
    - Sottolineati sul diagramma
- Parole chiave
  - Estensione della semantica UML
    - Costrutto simile + parola chiave!
      - «`interface`»
      - `{abstract}`
- Responsabilità
  - Funzionalità offerte
  - Aggiunta alla classe con commento



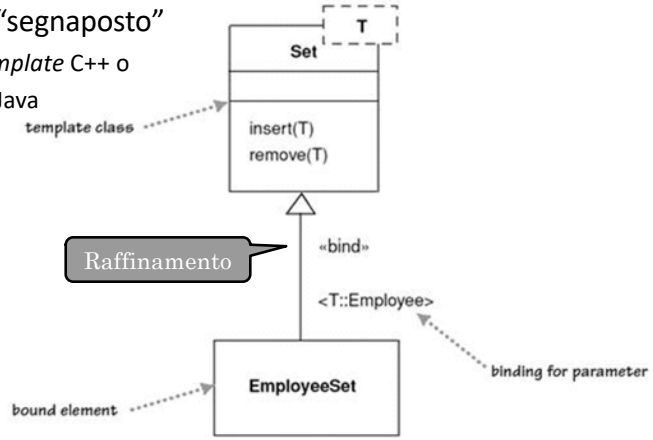
Caratteristiche Varie

- Proprietà derivate
  - Possono essere calcolate a partire da altri valori
    - Definiscono un vincolo fra valori
    - Si indicano con "/" che precede il nome della proprietà
- Proprietà *read only* e *frozen*
  - `{readOnly}`
    - Non vengono forniti i servizi di scrittura
  - `{frozen}`
    - Immutabile, non può variare nel suo ciclo di vita
- Enumerazioni
  - Insiemi di valori che non hanno altre proprietà oltre il valore simbolico
  - «`enumeration`»



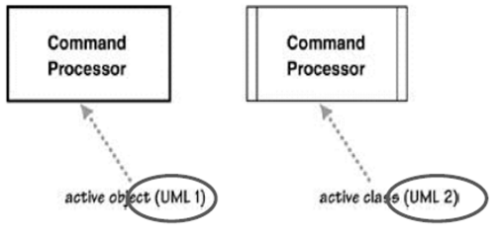
Caratteristiche Varie

- Classi Parametriche
  - T è detto "segnaposto"
    - Come *template* C++ o *generics* Java



Caratteristiche Varie

- Classi Attive
  - Eseguono e controllano il proprio *thread*



Consigli Utili

- Diagrammi molto ricchi di concetti
  - Non cercare di utilizzare tutte le notazioni disponibili
    - Cominciare dapprima con i concetti semplici
  - Una prospettiva concettuale permette di esplorare il linguaggio di un particolare *business*
    - Mantenere la notazione semplice e non introdurre concetti legati al *software*
  - Concentrarsi nel disegno dei diagrammi delle parti più importanti
    - Disegnare ogni cosa è sinonimo di diagrammi non fondamentali che diventano obsoleti molto presto!

Sommario

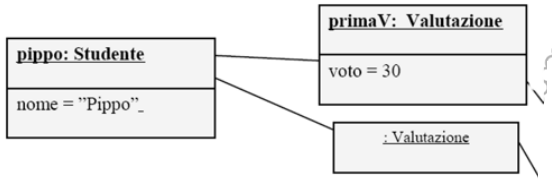
- Introduzione
- Proprietà e Operazioni
- Concetti base e avanzati
- Diagrammi degli oggetti

Diagrammi Degli Oggetti

- Grafo delle istanze, comprensivo di associazioni e valori delle proprietà

nome dell'istanza : nome della classe

  - Fotografia degli oggetti che compongono un sistema
  - Non ci sono parti obbligatorie
  - Specifica di istanza
    - Anche di classi astratte, omissione dei metodi, ecc…



Riferimenti

- OMG Homepage – www.omg.org
- UML Homepage – www.uml.org
- UML Distilled, Martin Fowler, 2004, Pearson (Addison Wesley)
- Learning UML 2.0, Kim Hamilton, Russell Miles, O'Reilly, 2006
- Dependency - http://rcardin.github.io/programming/oop/software-engineering/2017/04/10/dependency-dot.html

GitHub Repository

https://github.com/rcardin/swe