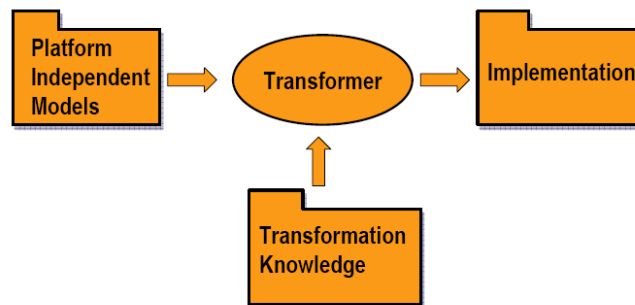


## Notazioni grafiche

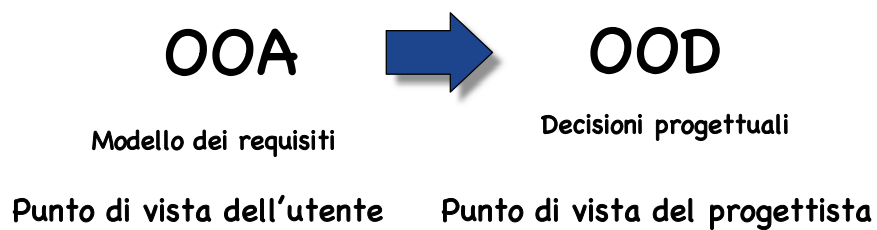
- Alcune discipline ingegneristiche dispongono di validi mezzi di rappresentazione (schemi, diagrammi di prestazioni e consumi, ...)
- Il software non dispone ancora di tecniche efficaci per descriverne la struttura, le funzionalità e le prestazioni
- UML cerca di rimediare a questa situazione
  - Standard OMG (Object Management Group)
  - Oggi siamo alla versione 2.0
  - Progettazione indipendente dal linguaggio di programmazione

## Model-driven Development



3

## Analisi vs. Design



- Transizione OOA -> OOD presenta spesso discontinuità: scelte architeturali

4

## UML

---

- **Diagrammi di struttura**
  - diagrammi delle classi, diagrammi degli oggetti, diagrammi dei componenti, diagrammi delle strutture composte, diagrammi dei package e i diagrammi di deployment
- **Diagrammi di comportamento**
  - diagrammi dei casi d'uso, diagrammi delle attività e diagrammi delle macchine a stati
- **Diagrammi di interazione**
  - diagrammi di sequenza, diagrammi di comunicazione, diagrammi di temporizzazione e diagrammi di interazione generale
- OCL (Object Constraint Language)

5

## Dettagli nella descrizione

---

- UML consente di esprimere graficamente livelli crescenti di dettaglio nella descrizione delle classi
- Questi livelli crescenti di dettaglio sono spesso inappropriati o addirittura completamente fuori luogo nella specifica dei requisiti
- Diventano invece essenziali nella descrizione dell'architettura della soluzione, dove le classi corrispondono esattamente alle classi della soluzione in Java

6

## UML ↔ Java

---



7

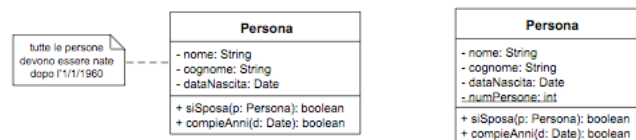
## Diagramma delle classi

---

Classi, interfacce e relazioni

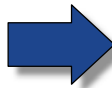
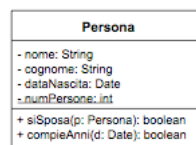
## Classe

- Composta da tre parti
  - Nome
  - Attributi (lo stato)
  - Metodi (il comportamento)
- **Attributo:** visibilità nome: tipo [molteplicità] = default {stringa di proprietà}
- **Metodo:** visibilità nome (lista parametri): tipo di ritorno {stringa di proprietà}
- **Visibilità:** + public, - private, # protected, ~ friendly
- **Parametro:** direzione nome: tipo = default



9

## Traduzione



```
class Persona {
    private String nome;
    private String cognome;
    private Date dataNascita;
    private static int numPersone;

    public boolean siSposa(Persona p) {
        ...
    }

    public boolean compieAnni(Date d) {
        ...
    }
}
```

10

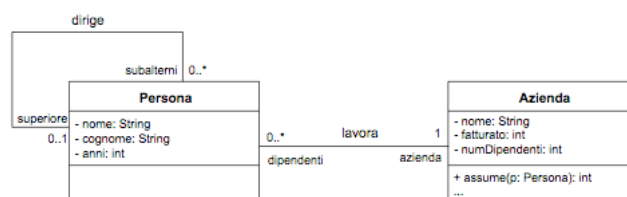
## Associazioni

- Un'associazione indica una relazione tra classi
  - ad esempio persona che lavora per azienda
- Un'associazione può avere
  - un nome (solitamente un verbo)
  - i ruoli svolti dalle classi nell'associazione
- Gli estremi di un'associazione
  - sono "attributi impliciti"
  - hanno visibilità come gli attributi normali
  - hanno una molteplicità
    - 1, 0..1, 1..\*, 4, 6-12

11

## Ruolo

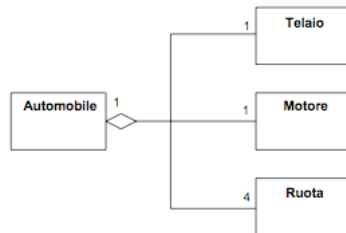
- Definisce il ruolo svolto nell'associazione



12

## Aggregazioni

- Le aggregazioni sono una forma particolare di associazione
- Una parte è in relazione con un oggetto (part-of)



13

## Composizioni

- Una relazione di composizione è un'aggregazione forte
  - Le parti componenti non esistono senza il contenitore
    - Creazione e distruzione avvengono nel contenitore
    - I componenti non sono parti di altri oggetti



- In Java aggregazioni e composizioni si traducono allo stesso modo
  - In C++ esistono modi differenti

14

## Esempio1

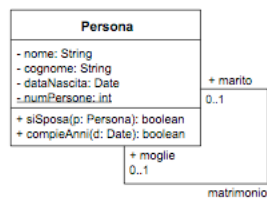


```
class Persona {
    ...
    private Casa casa;
    ...
}
```

```
class Casa {
    ...
    ArrayList<Persona> persone;
    ...
}
```

15

## Esempio2



```
class Persona {
    private String nome;
    private String cognome;
    private Date dataNascita;
    private static int numPersone;
    public Persona marito;
    public Persona moglie;

    public boolean siSposa(Persona p) {
        ...
    }

    public boolean compieAnni(Date d) {
        ...
    }
}
```

16



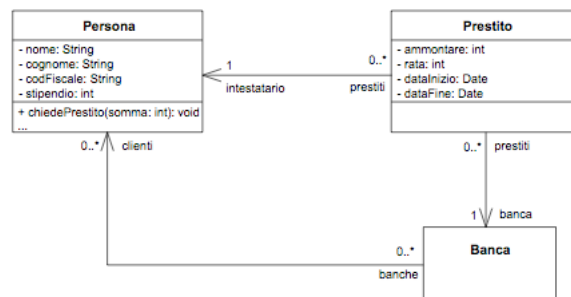
## Esempio3

```
class Persona {  
    private String nome;  
    private String cognome;  
    private String codFiscale;  
    private int stipendio;  
}  
  
class Banca {  
    private ArrayList<Persona> clienti;  
}
```

```
class Prestito {  
    private int ammontare;  
    private int rata;  
    private Date dataInizio;  
    private Date dataFine;  
    private Persona intestatario;  
    private Banca banca;  
}
```

17

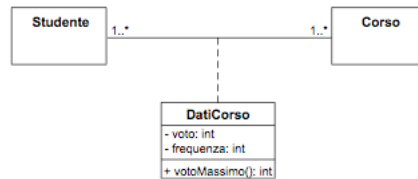
## Esempio3



18

## Classi di associazione

- Alcune proprietà potrebbero appartenere all'associazione e non alle parti coinvolte

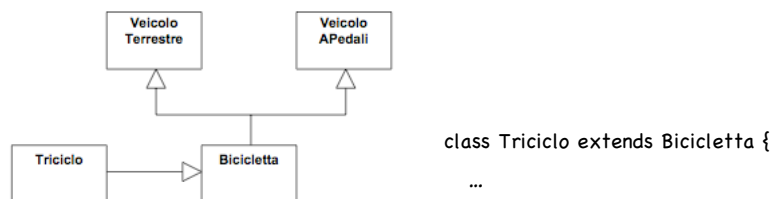


- In Java potremmo pensare a tre classi distinte

19

## Ereditarietà (Generalizzazione)

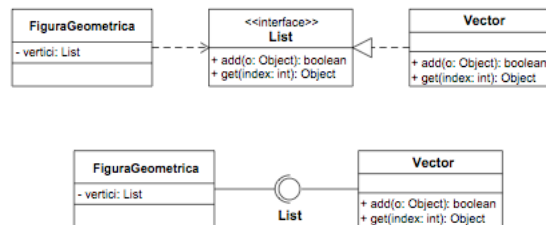
- Esplicita eventuali comportamenti comuni



- Possibilità di ereditare da più classi
  - Vietato in Java
- Può portare a conflitti fra attributi o servizi con lo stesso nome ereditati da classi diverse

20

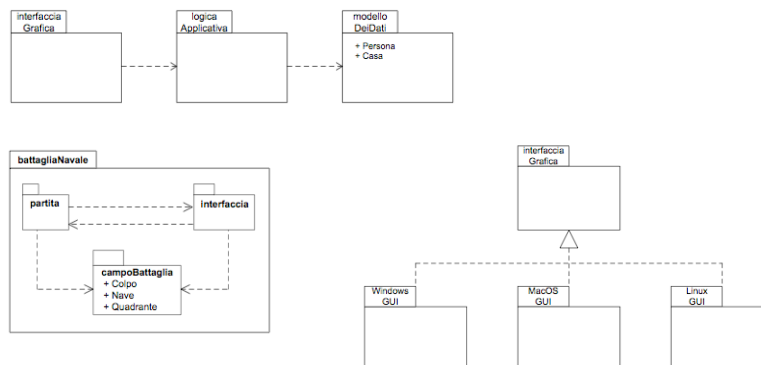
## Interfacce



21

## Package

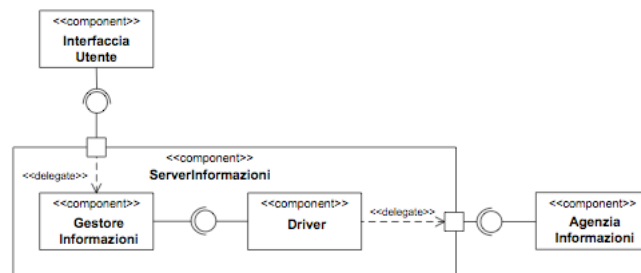
- Decomposizione gerarchica e dipendenze tra package
- In Java esiste un concetto simile



22

## Componenti

- Utili per “decomporre” il sistema in esame



23

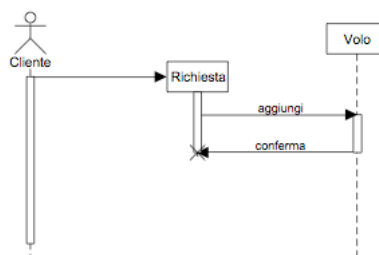
## Diagrammi di sequenza

## Utili per

- I diagrammi di sequenza rappresentano interazioni tra oggetti
  - Materializzazione di scenari specifici
- Sono utili per
  - Evidenziare le interazioni tra oggetti e quindi i metodi da associare alle diverse classi
  - Provare l'efficacia dei metodi identificati

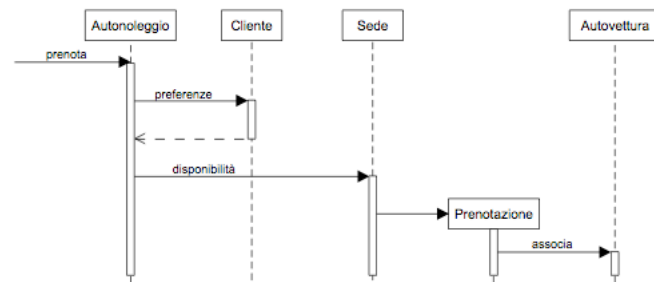
25

## Il caso più semplice



26

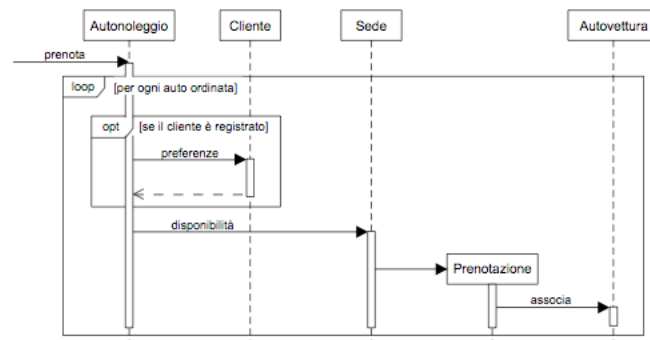
## Cosa succede se ...



27

## Frame di interazione

- ref
- alt
- opt
- loop
- par
- neg
- ...



28