
# UML e Java

# UML

- 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 alle versione 2.X
  - Progettazione indipendente dal linguaggio di programmazione

# Alcuni diagrammi

- **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)

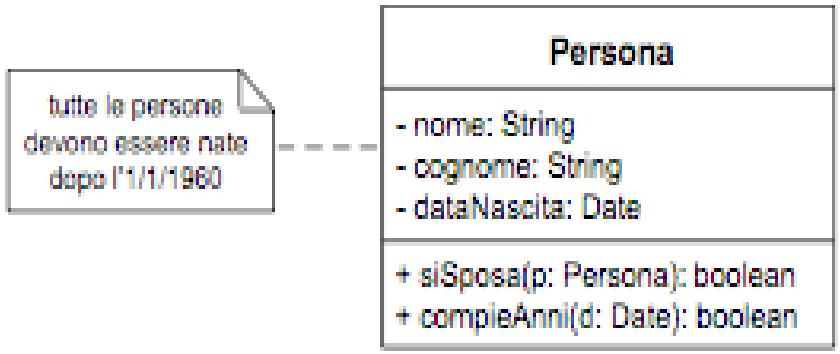
# Dettagli nella descrizione

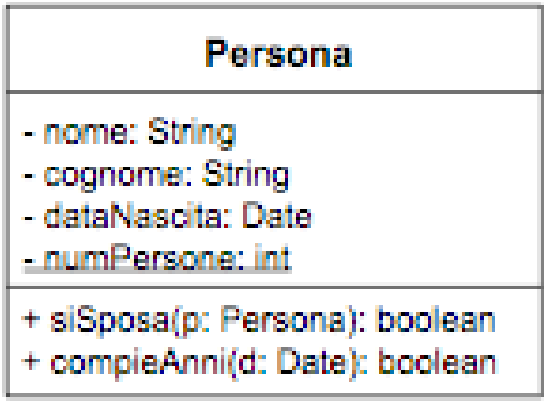
- UML consente di esprimere graficamente livelli crescenti di dettaglio nella descrizione delle classi
- Questi livelli crescenti di dettaglio sono spesso inappropriati o addirittura completamente fuori luogo nella specifica dei requisiti
- Diventano invece essenziali nella descrizione dell'architettura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

# Classe

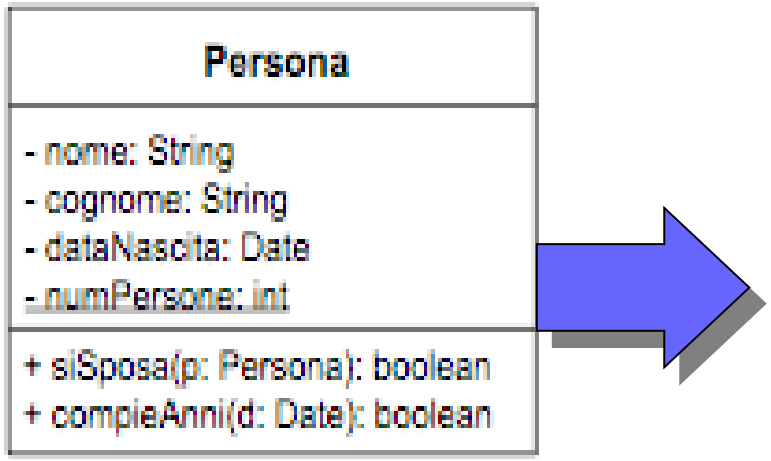
- Composta da tre parti
  - Nome
  - Attributi (lo stato)
  - Metodi (il comportamento)

- 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



# Traduzione

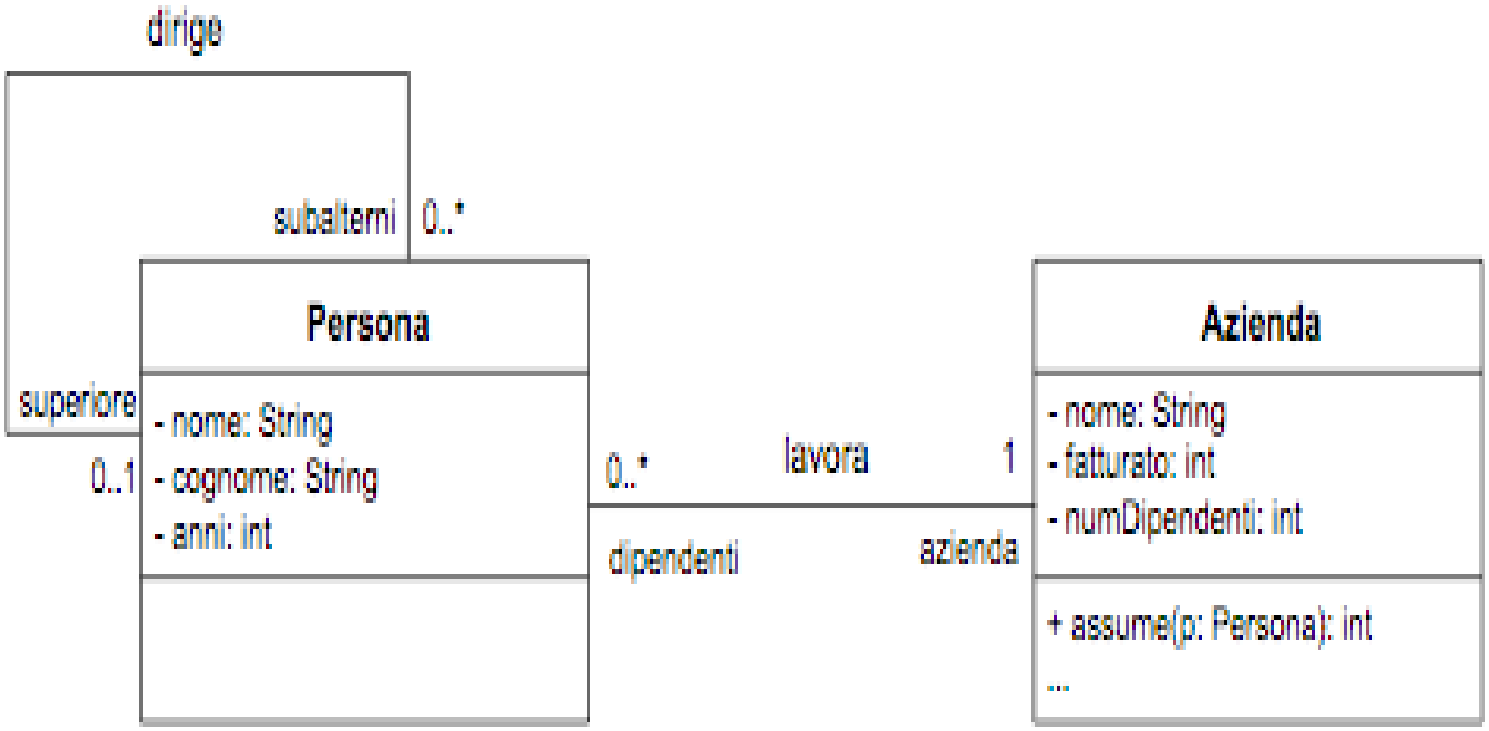



```
public class Persona {
  private String nome;
  private String cognome;
  private Date dataNascita;
  private static int numPersone;

  public boolean siSposa(Persona p) {
     …
  }

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

# Associazioni

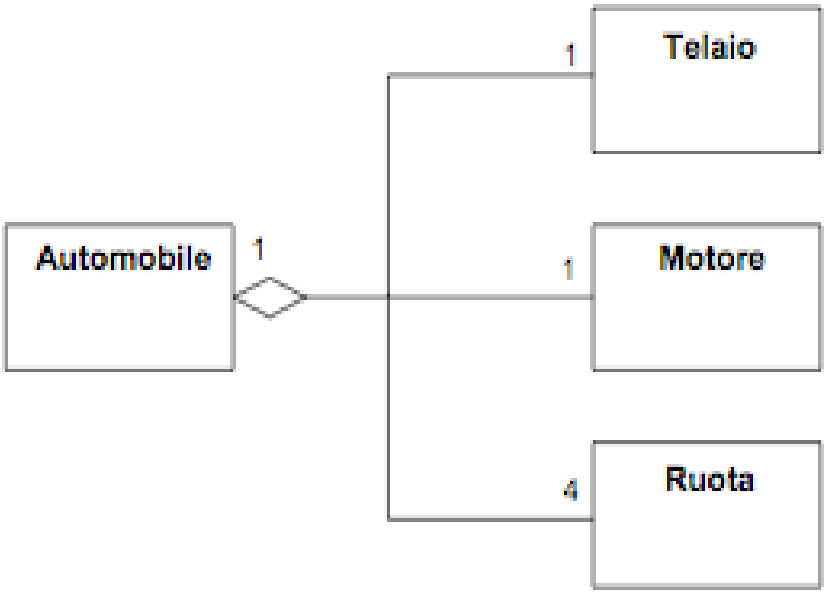
- Un’associazione indica una relazione tra classi
  - ad esempio persona che lavora per azienda
- Un’associazione può avere
  - un nome (solitamente un verbo)
  - i ruoli svolti dalle classi nell’associazione
- Gli estremi di un’associazione
  - sono “attributi impliciti”
  - hanno visibilità come gli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

# Diagramma delle classi



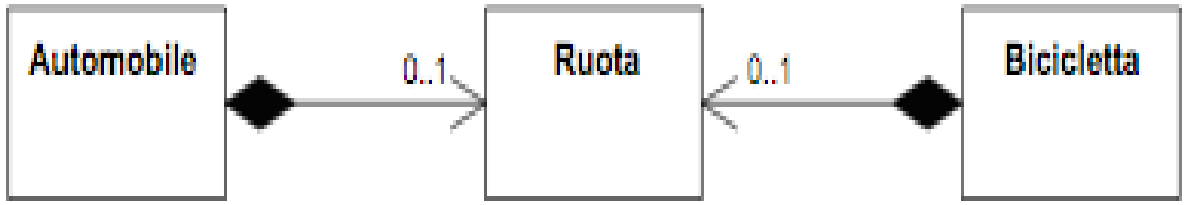
# Aggregazioni

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



# 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



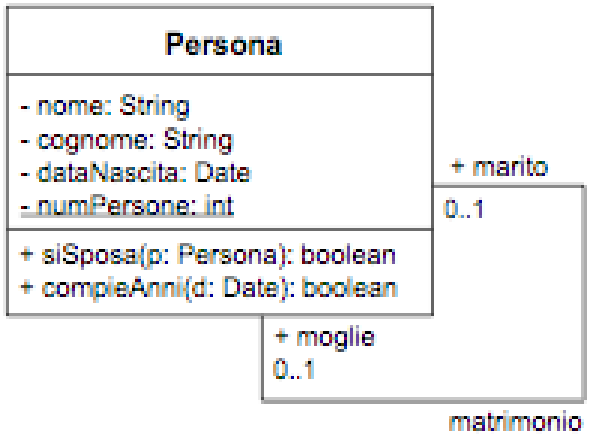

- In Java aggregazioni e composizioni si traducono allo stesso modo
  - In C++ esistono modi differenti

# Esempio1


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

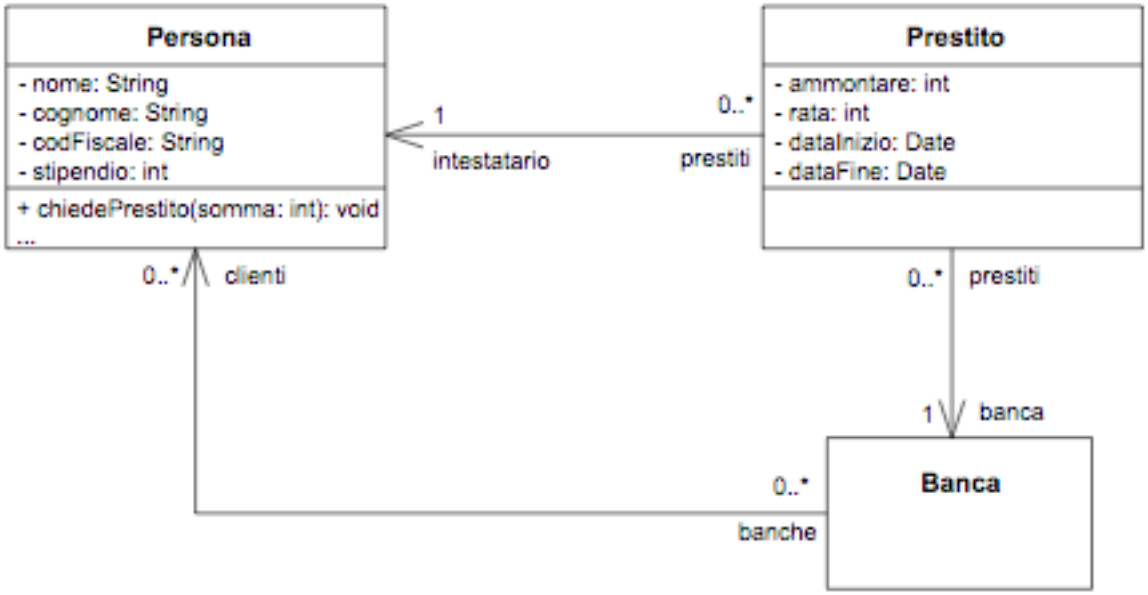
```
class Casa {
    …
    private Persona[] persone;
    …
}
```

# 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) {
      …
   }
}
```

# Esempio3


Persona
- nome: String
- cognome: String
- codFiscale: String
- stipendio: int
+ chiedePrestito(somma: int): void
...
Prestito
- ammontare: int
- rata: int
- dataInizio: Date
- dataFine: Date
1
intestatario
0..*
prestiti
0..*
clienti
0..*
prestiti
1
banca
0..*
banche
Banca

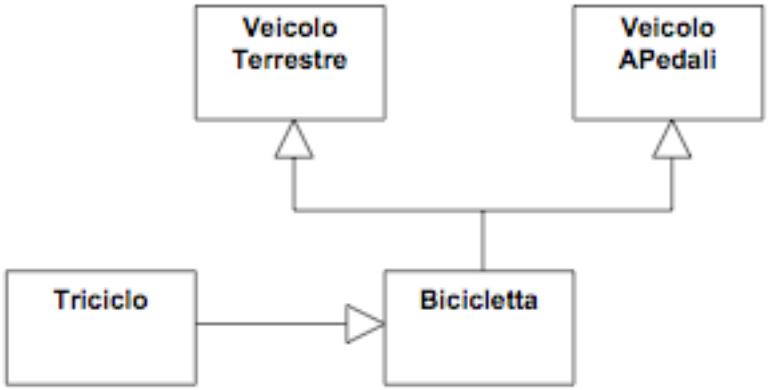
# Esempio3

```
class Persona {
   private String nome;
   private String cognome;
   private String codFiscale;
   private int stipendio;
}

class Banca {
   private Persona[] clienti;
}
```

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

# Ereditarietà (Generalizzazione)

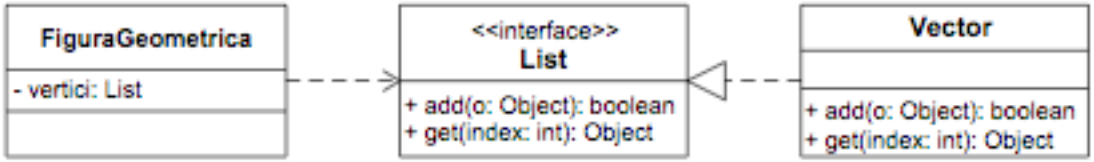
- Esplicita eventuali comportamenti comuni




```
class Triciclo extends Bicicletta {

...
```

- 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

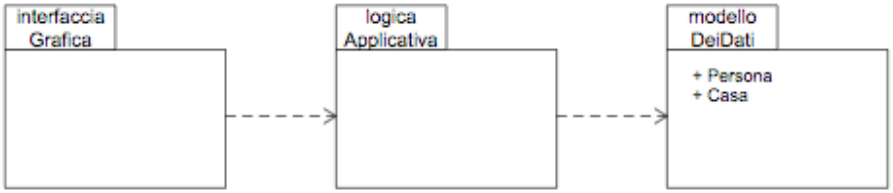
# Interfacce

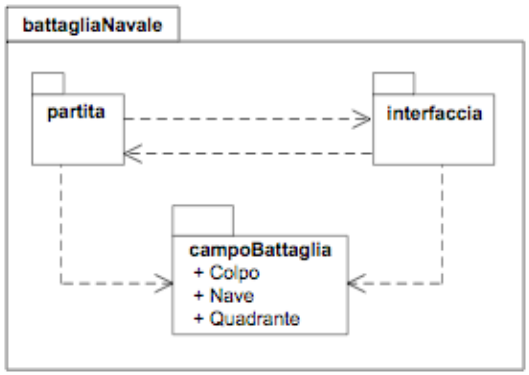

FiguraGeometrica
- vertici: List
<<interface>>
List
+ add(o: Object): boolean
+ get(index: int): Object
Vector
+ add(o: Object): boolean
+ get(index: int): Object

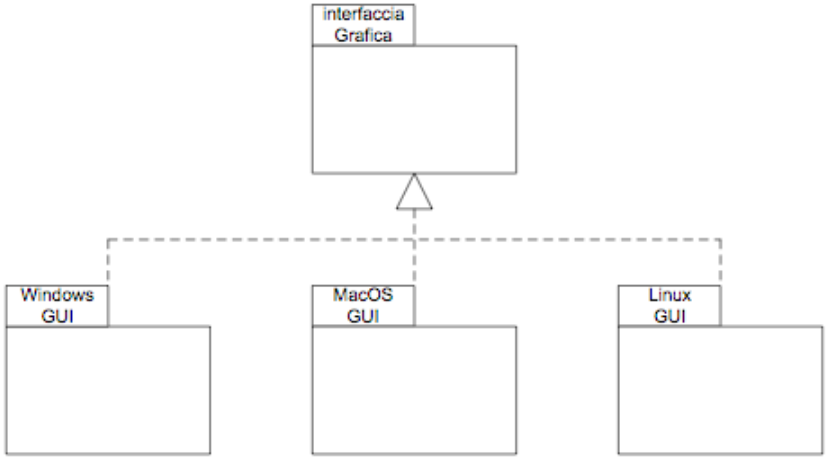

FiguraGeometrica
- vertici: List
List
Vector
+ add(o: Object): boolean
+ get(index: int): Object

# Package

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

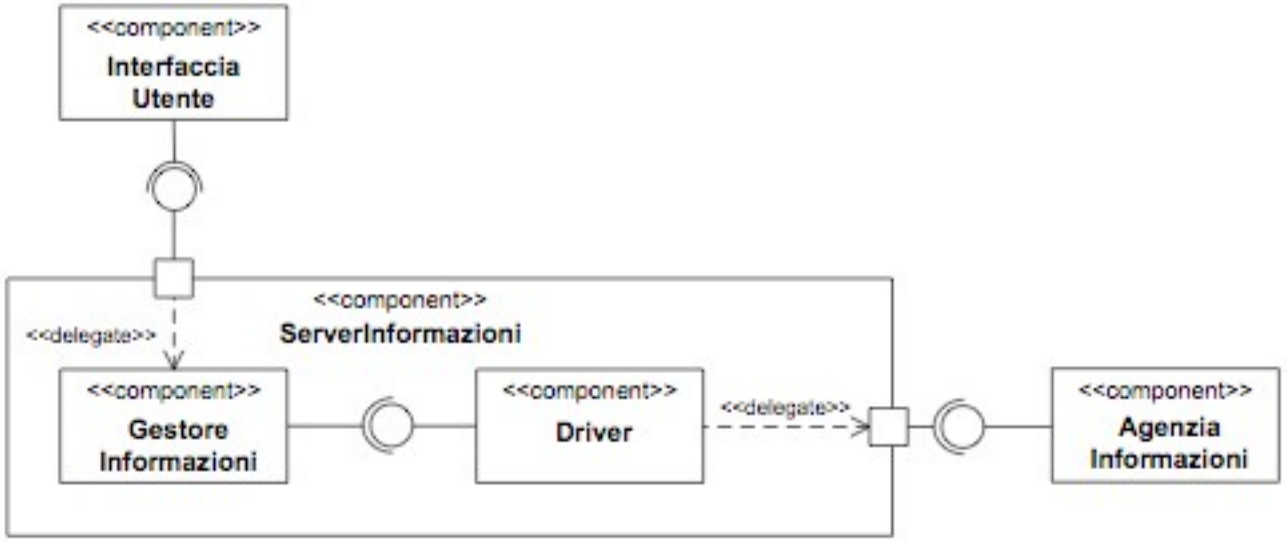


# Componenti

- Utili per “decomporre” il sistema in esame

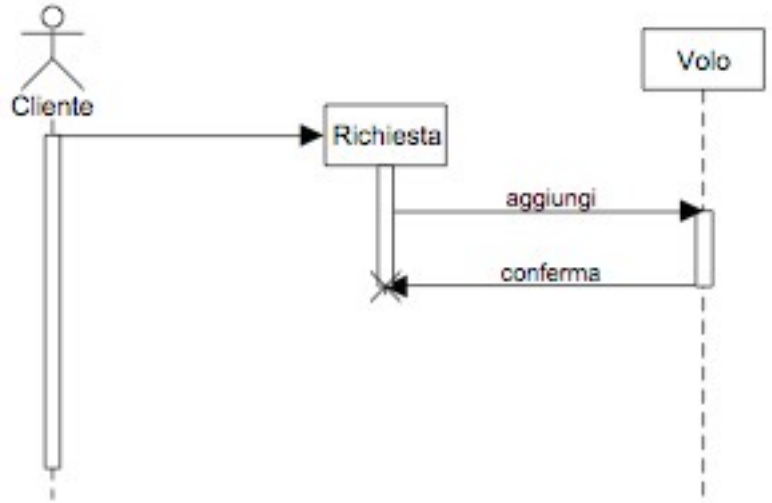


# Diagrammi di sequenza

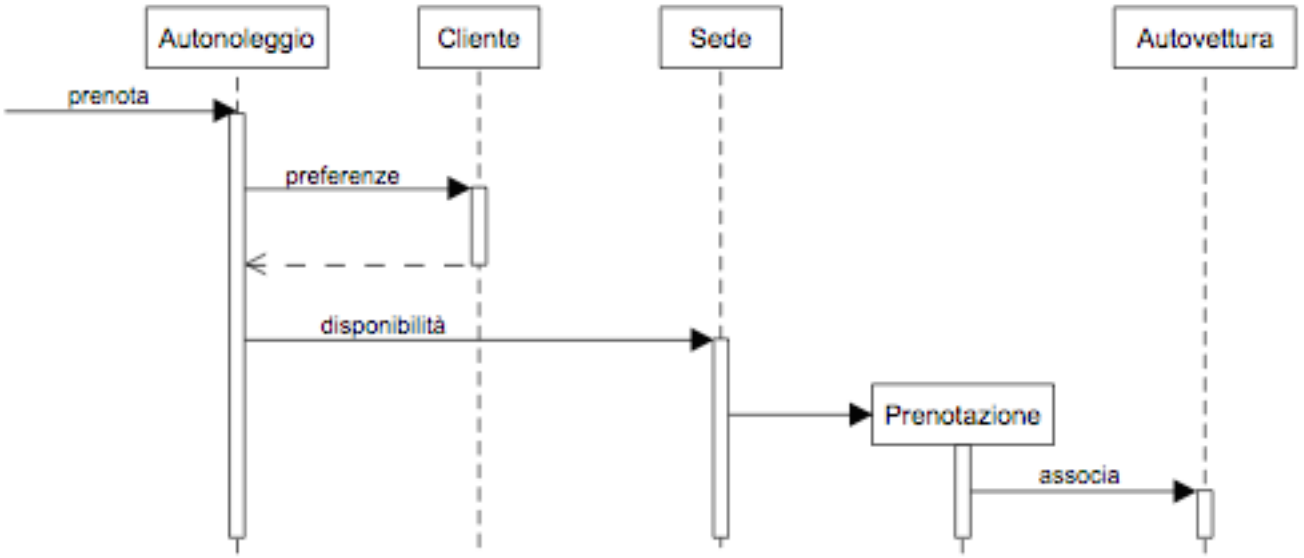
- I diagrammi di sequenza rappresentano interazioni tra oggetti
  - Materializzazione di scenari specifici

- Sono utili per
  - Evidenziare le interazioni tra oggetti e quindi i metodi da associare alle diverse classi
  - Provare l'efficacia dei metodi identificati

# Il caso più semplice



# Cosa succede se …



# Frame di interazione

- ref
- alt
- opt
- loop
- par
- neg
- …