Linguaggi di Programmazione I – Lezione 6

Prof. Marcello Sette
mailto://marcello.sette@gmail.com
http://sette.dnsalias.org

8 aprile 2008

Panoramica della lezione

Analisi di oggetti e classi

Relazioni tra classi

Analisi della dinamica del modello

Bibliografia

Analisi di oggetti e classi 3 / 56

Introduzione

- La fase di analisi identifica gli oggetti richiesti in esecuzione per assicurare la funzionalità del sistema.
- Segue la fase di *Individuazione delle Specifiche* e degli *Use Case*, precede la fase del *Progetto del Sistema*.
- Definisce *cosa* deve fare il sistema.
- Evita di descrivere dettagli di progettazione e realizzazione.
- Pone l'accento sui componenti del sistema.

Durante questa fase bisognerebbe rispondere a domande:

- Quali sono gli oggetti del sistema?
- Quali sono le possibili classi?
- Come sono correlati gli oggetti?
- Quali sono le responsabilità di ciascun oggetto o classe?
- Come sono correlati oggetti e classi?

Astrazioni chiave

- Riferiscono ad una sottolista di parole all'interno della lista dei possibili oggetti.
- Rappresentano gli oggetti primari o principali nel sistema.
- Sono identificate dopo aver studiato ciascun possibile oggetto e aver deciso se è abbastanza importante per essere una Astrazione Chiave.

Esempi

Esempio 1

Siano stati individuati i seguenti oggetti possibili, in riferimento ad un sistema-tavolo:

- Tavolo
- Gamba
- Piano
- Colore
- Lunghezza
- Larghezza
- Altezza
- Peso
- Data di acquisto
- Materiale

Quali di questi sono componenti "veri" di un tavolo, e pertanto gli oggetti all'interno di un sistema-tavolo?
Quali di questi sono attributi di altri degli oggetti?
La risposta è soggettiva e dipende da ciò che inizialmente era richiesto al modello.
Ma una possibile lista di astrazioni chieve potrebbe essere:

- Tavolo
- Gamba
- Piano

Gli altri elementi, come il Peso o il Colore, sono attributi di uno (o più) oggetti.

Esempio 2

Siano stati individuati i seguenti oggetti possibili, in riferimento al sistema Jolly-Hotel.

- Camera
- Hotel
- Ospite
- Prezzo Base
- Correzione Stagionale al Prezzo
- Prenotazione
- Receptionist

Ma una possibile lista di astrazioni chiave potrebbe essere:

- Camera
- Hotel
- Ospite
- Prenotazione

Oggetti e classi

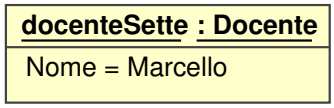
Il passo successivo è quello di rappresentare gli aspetti logici e fisici del modello statico.
I diagrammi utilizzati sono i Class Diagram e gli Object Diagram.

- I diagrammi Class mostrano le classi che dovrebbero costituire il sistema, insieme con tutte le possibili relazioni tra esse.
- I diagrammi Object rappresentano gli oggetti presenti nel sistema ad un certo istante di tempo.

Entrambi i diagrammi utilizzano la stessa sintassi di base e possono essere prodotti contemporaneamente.
Di solito i diagrammi Class sono prodotti prima di quelli Object (pensare prima in modo astratto, poi specificare e verificare), anche se alcuni analisti preferiscono lavorare in modo inverso (prima elencare i particolari, poi produrre le astrazioni).

Classi e oggetti UML

In aggiunta a quanto detto nelle precedenti lezioni:



- Gli oggetti non hanno la sezione dei metodi, poiché essi possiedono tutti i metodi della classe di cui sono istanze.
- Analogamente gli oggetti non hanno la sezione degli attributi, ma possono avere una sezione in cui sono elencati solo i *valori* degli attributi della classe che rendono unico quell'oggetto.

Attributi e metodi

- Non sono noti prima della fase di Progettazione.
- Alcuni, però, sono già ovvi e possono essere aggiunti anche nella fase di Analisi.
- In questo momento il fatto che un attributo o un metodo esista è già sufficiente.
- Non è essenziale aggiungere informazioni sul tipo di un attributo o sui parametri di un metodo.
- Gli attributi che si possono qui identificare sono i nomi del dizionario dati che non erano stati classificati come astrazioni chiave. Il fatto che essi siano stati utilizzati per descrivere il sistema, significa che sono rilevanti in qualche modo.

Esercizio/Progetto (Quarta parte)

1. Nel sistema scelto, identificare la lista delle astrazioni chiave dalla lista degli oggetti candidati.
2. Studiare le astrazioni chiave aggiungendo gli attributi e metodi che si possono già identificare.
3. Tracciare un diagramma Class usando le astrazioni chiave dei punti precedenti.

Relazioni tra classi

Introduzione

- C'è una classe che ha bisogno di un'altra per una certa funzionalità?
- Vi sono relazioni così forti che un oggetto non può esistere senza un altro?
- Vi sono classi che hanno strutture o attributi simili?
- Vi sono classi che esibiscono comportamenti simili?
- Vi sono classi che hanno una genealogia simile nell'ambito del dominio del problema?

Le relazioni tra classi che costruiremo rispondendo alle domande precedenti sono:

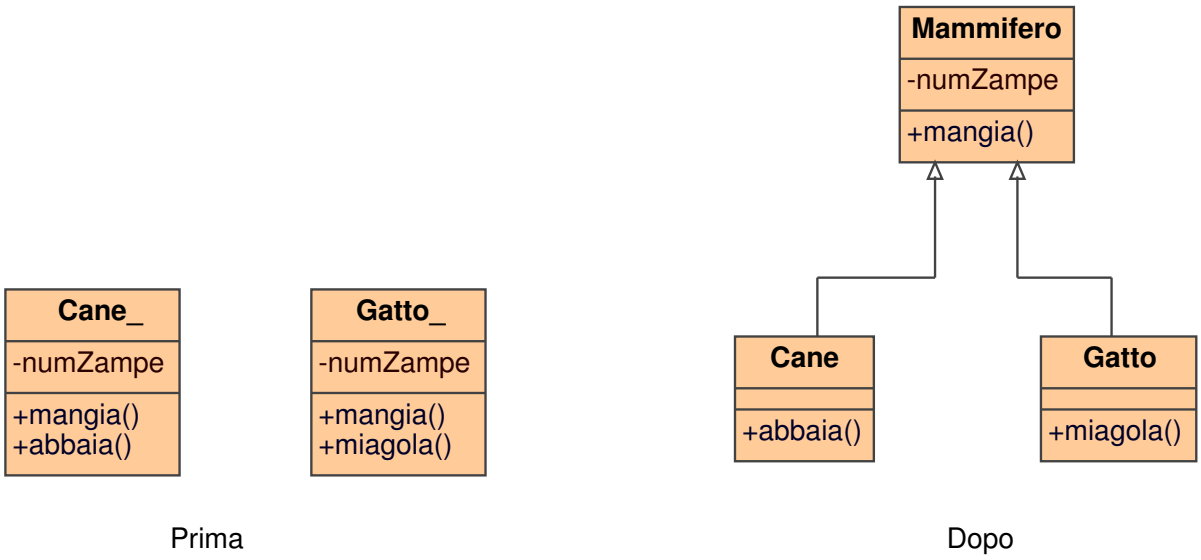
- Ereditarietà
  - Generalizzazioni
  - Specializzazioni
- Polimorfismo
- Classi astratte
- Associazioni
  - Aggregazione
  - Composizione
  - Associazioni complesse

Ereditarietà

- Il concetto di ereditarietà descrive come si possano condividere attributi e funzionalità tra classi di natura o scopo simile.
- Si rivedano le lezioni precedenti per la notazione UML usata per le ereditarietà e per i concetti di:
  - *classificazione*;
  - *ereditarietà singola* o *multipla*;
  - *sottoclassi* o *superclassi*.
- Vi sono due modi con cui si può aggiungere il concetto di ereditarietà al modello del sistema:
  - Generalizzazione.
  - Specializzazione.

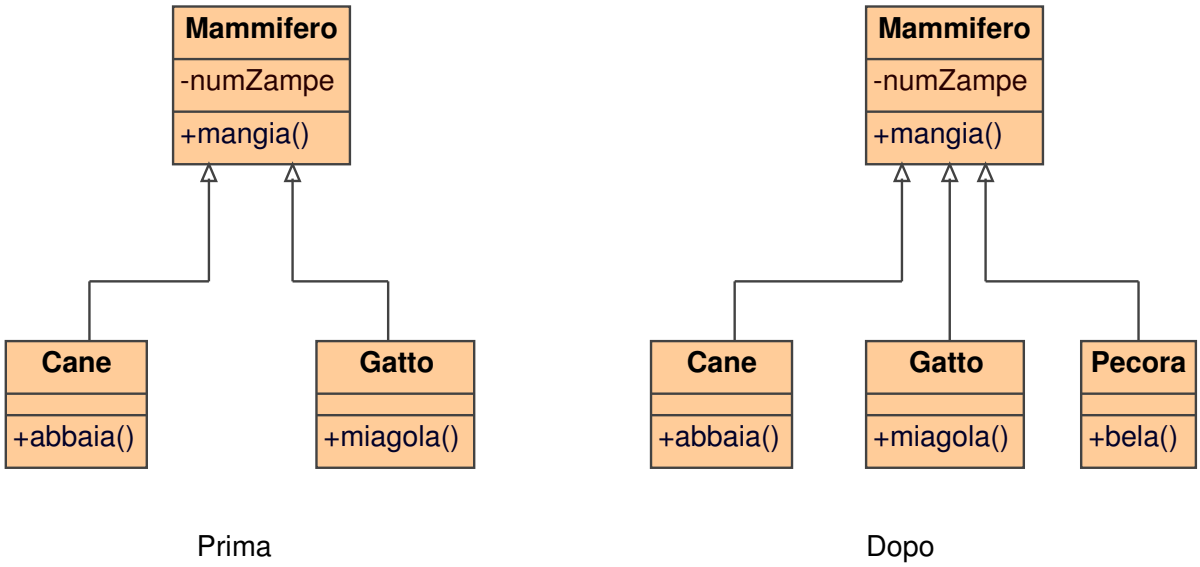
Generalizzazione

Avviene quando si riconosce che più classi dello stesso diagramma Class esibiscono funzionalità, struttura, scopo comuni.
L'analista decide allora di creare una nuova classe contenente gli attributi e le funzionalità comuni e semplifica le classi precedenti come estensioni della nuova (generalizzata).



Specializzazione

Avviene quando si riconosce che la nuova classe che si sta aggiungendo ha le stesse funzionalità, struttura e scopo di una classe esistente, ma ha bisogno di nuovo codice o attributi.
La nuova classe è una forma *specializzata* di quella già esistente.



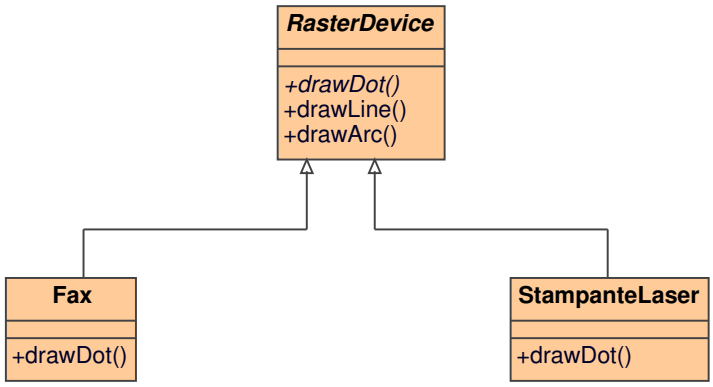
Prima Dopo

Polimorfismo

- Strettamente correlato con l'ereditarietà.
- Riguardando gli esempi precedenti si può dire che vi sono *molte forme* di Mammiferi nel sistema.
- L'effetto pratico è che un riferimento ad un Mammifero può essere usato sia per riferirsi ad un oggetto Cane, sia ad un oggetto Gatto.
- Senza polimorfismo sarebbe difficile scrivere metodi che possono operare su diversi tipi di oggetti.
- In una classe si può dichiarare una variabile per riferirsi ad un Mammifero prima che si sappia esattamente quante classi estendano Mammifero e quali saranno istanziate dal sistema in una particolare circostanza.

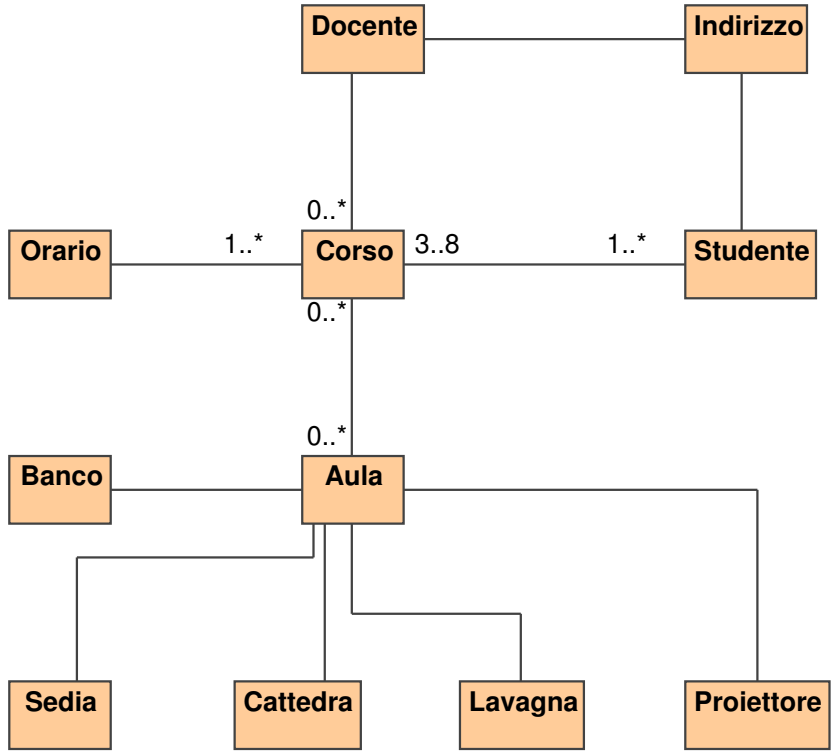
Classi astratte

- Contengono funzionalità incomplete (metodi astratti, privi di realizzazione) e non può avere istanze.
- Una classe che estende una classe astratta eredita tutti i suoi metodi (inclusi quelli astratti).
- Ogni classe che eredita metodi astratti è considerata astratta essa stessa, a meno che essa non realizzi tutti i metodi astratti ereditati.
- In UML i nomi di classi o metodi astratti sono in carattere *italico*.



Associazioni e molteplicità

Rivedere le definizioni date nelle lezioni precedenti ed interpretare il Class Diagram:

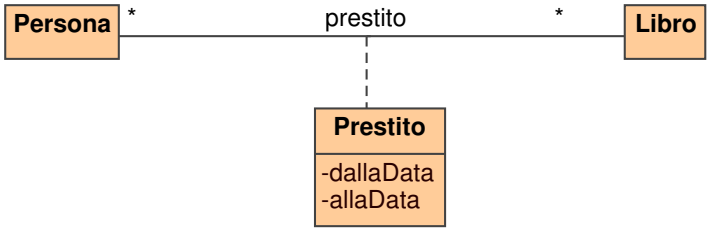


Associazioni complesse

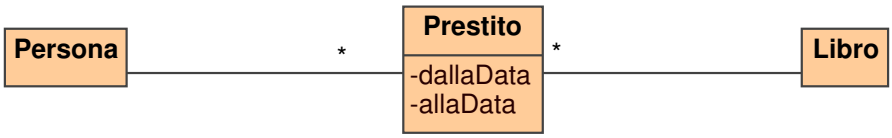
- Sono quelle in cui vi sono molteplicità maggiori di 1 in entrambi i ruoli.

- È difficile pensare ad una possibile realizzazione.
- Due tecniche per la risoluzione:
  - Classe di associazioni.
  - Associazione qualificata.

Classe di associazioni

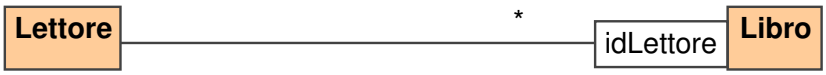


- Il nome della classe deve essere lo stesso della associazione.
- Una Cl. di Ass. è usata per documentare la risoluzione di una associazione complessa, non fa parte delle astrazioni chiave.
- Restano nei diagrammi Class finché, nella fase di progettazione, non si decide di realizzarle.
- Per realizzarle, bisognarà inserirle in altro modo nei diagrammi:



Associazioni qualificate

- Usate quando si vuole evidenziare che la realizzazione futura sarà mediante un array, una hash table, un dizionario ...
- In questo caso, se, per esempio, nella biblioteca ogni lettore (istanza di Lettore) ha il proprio unico codice identificativo (indice nell'array), ogni accesso dal punto di vista del Libro avviene mediante tale codice identificativo:

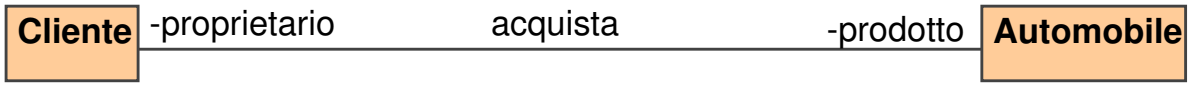


Viceversa, se nella biblioteca ogni libro è registrato usando un unico codice, il riferimento ad esso, dal punto di vista del Lettore, avviene mediante tale codice:

- La differenza realizzativa è che, mentre con una vera e propria classe di associazioni, le informazioni sul singolo prestito sono localizzate in una istanza di quella classe, nel caso delle associazioni qualificate, le informazioni sui prestiti sono delocalizzate e modificare un prestito significa modificare una istanza di Libro ed una istanza di Lettore.

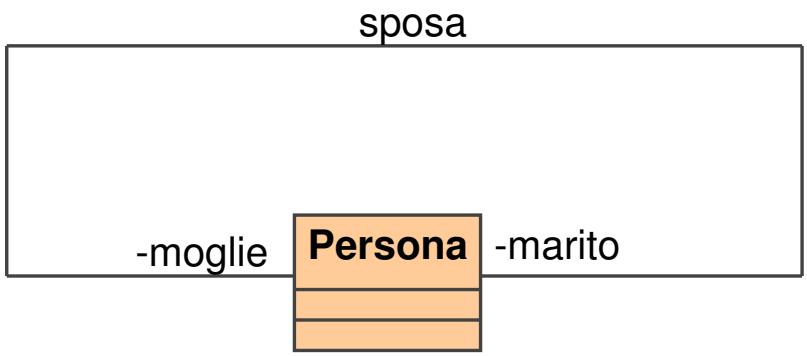
Ruoli nelle associazioni

- Riferiscono ad una associazione tra classi.
- Etichette poste alle estremità di una associazione.
- Rappresentano un attributo all'interno della classe che si trova all'estremità opposta del nome.



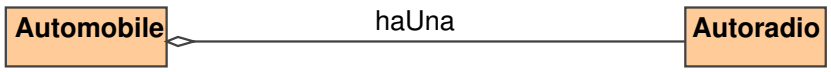
Assoc. riflessive

- In un diagramma Object possono esistere link tra istanze della stessa classe.
- Ma c'è una sola classe nel diagramma Class per rappresentare entrambi gli oggetti.
- In tal caso è usata una associazione riflessiva.



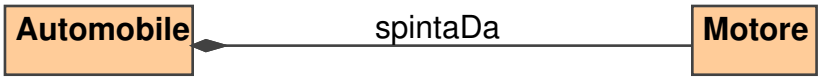
Aggregazioni

- Sono forme di associazioni.
- Maggiore enfasi su come i due oggetti sono correlati nel sistema.
- Caratterizzata dalla relazione "ha un".
- Possono avere molteplicità.
- In UML:



Composizioni

- Sono forme di associazioni.
- Maggiore enfasi su come i due oggetti sono correlati nel sistema.
- Caratterizzata dalla relazione "contiene sempre".
- Possono avere molteplicità.
- In UML:



Esercizio/Progetto (Quinta parte)

1. Usando le astrazioni chiave preparate nell'esercizio precedente, aggiungere linee di associazione tra classi.
2. Nominare le associazioni aggiunte al diagramma di Classe, aggiungendo indicatori di direzione se necessario.
3. Aggiungere valori di molteplicità alle associazioni.
4. Simulando l'esecuzione del sistema, tracciare un diagramma Object relativo ad una parte del diagramma Class ad un certo istante di tempo, in modo da verificare i valori di molteplicità.
5. Determinare e marcare le associazioni complesse nel diagramma Classe sviluppato negli esercizi precedenti.
6. Decidere quali attributi sono richiesti per risolvere ogni associazione complessa.
7. Risolvere le associazioni complesse nel modo più appropriato.
8. Riconsiderare il diagramma Class del proprio problema e ricercare le classi simili che possono essere generalizzate.
9. Aggiungere la classe generalizzata al diagramma e tracciare archi di ereditarietà tra le opportune classi.
10. Riconsiderare le specifiche del proprio problema. Può esistere una nuova classe che era sfuggita al diagramma Class? Se è così, aggiungere la classe e tracciare gli opportuni archi di ereditarietà.
11. Riprendere il diagramma Class e cercare, se possibile, di evidenziare le classi ed i metodi astratti.
12. Specificare quali debbano essere, nella gerarchia di ereditarietà, le classi che realizzano i metodi astratti.
13. Aggiungere i nomi di ruolo ad ogni associazione nel sistema in esame.
14. Evidenziare, se possibile, associazioni riflessive.
15. Individuare se possibile, nel problema in esame, le relazioni di aggregazione e composizione. Rappresentarle in modo appropriato, usando la sintassi UML.

Analisi della dinamica del modello 27 / 56

Riepilogo

- Formalizzazione del problema.
- Dizionario.
- Diagrammi Use Case.
- Scenari di uno Use Case.
- Astrazioni chiave.
- Diagrammi Class e Object.

Modellazione dinamica

Consiste nello specificare come gli oggetti operano e cooperano nel tempo.
Essa avviene due volte:

- durante la fase di analisi iniziale (quando si vuole essere sicuri che le operazioni siano possibili nel sistema);
- durante la fase di progetto fisico (per assegnare le descrizioni dei metodi alle classi appropriate).

In UML quattro diagrammi base:

- diagrammi Sequence;
- diagrammi Collaboration;
- diagrammi State;
- diagrammi Activity.

Responsabilità

Nel paradigma OO, "Responsabilità" significa:

- qualcosa che la classe conosce (stato);
- qualcosa che la classe sa (comportamento);
- qualcosa che l'oggetto conosce (stato);
- qualcosa che l'oggetto sa (comportamento).

Un comportamento definito in una interfaccia può essere realizzato in più di una classe. In questo caso si ha una realizzazione *polimorfica* di una responsabilità.

Il comportamento di una classe può essere distribuito in più sottoclassi oppure realizzato tutto nella stessa classe.
La qualità complessiva di progettazione OO è determinata da come e dove vengono distribuite le responsabilità.
Questo è un compito cruciale nella programmazione OO, tanto che sono stati sviluppati alcuni principi generali da usare come linee guida per l'assegnazione delle responsabilità (*Patterns*).

Evoluzione del sistema

Descrivere come cambia nel tempo un sistema significa considerare vari aspetti:

1. determinare se ciascuna operazione è una richiesta singola da parte di un utente o di altri sistemi;
2. determinare gli oggetti coinvolti in ogni singola operazione, come essi interagiscono e come ciascuna operazione può alterare lo stato interno degli oggetti.

I diagrammi usati in questo caso sono i *Sequence*, *Collaboration*, *State*.

Diagrammi Sequence e Collaboration

- Sono usati per descrivere gli scenari di uno stesso Use Case (si veda la lezione 5).
- Durante la fase di Analisi devono riflettere solo le interazioni.
- Durante la fase di Progettazione, ciascuna interazione sarà convertita nell'invocazione di un metodo nel linguaggio OO scelto.
- Per un certo Use Case dovrebbero essere prodotti tanti diagrammi Sequence e/o Collaboration, uno per ogni scenario di quello Use Case.
- Fare riferimento alla bibliografia e alla discussione della lezione 5 per la sintassi di diagrammi Sequence e Collaboration.

Esercizio/Progetto (Sesta parte)

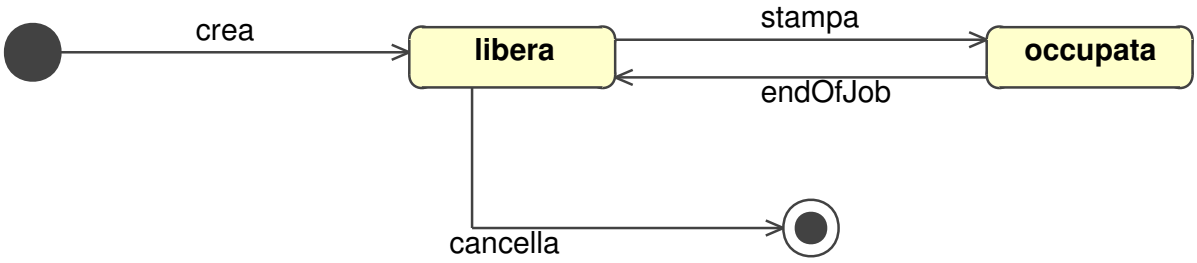
1. Rivedere ed eventualmente correggere gli scenari per il sistema in esame.
2. Alla luce dei diagrammi Class e Object già sviluppati, tracciare i diagrammi Sequence per gli scenari utilizzati.
3. Tracciare in alternativa o a supplemento i diagrammi Collaboration.

Diagrammi State

- Mostrano come cambia un oggetto nel tempo a causa dell'invocazione di metodi.
- Consistono di:
  - *Stati*: uno stato è l'insieme di valori degli attributi;
  - *Eventi*: un evento è lo stimolo che in un oggetto causa la transizione da uno stato all'altro.
- Sono importanti per descrivere gli stati legalmente assumibili dagli oggetti e la natura delle modifiche che possono legittimamente accadere nel tempo.
- Fare riferimento alla bibliografia e alla discussione della lezione 5 per la sintassi di diagrammi State.

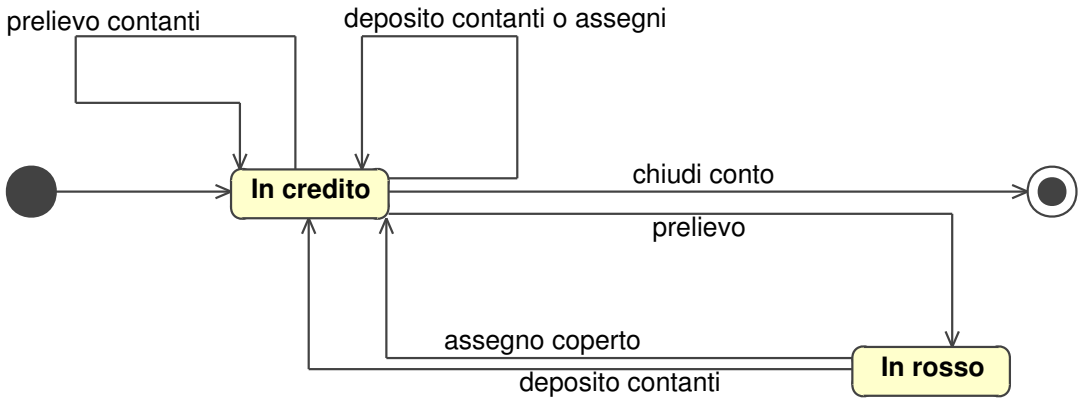
Esempio 1

Diagramma di una istanza di Stampante:

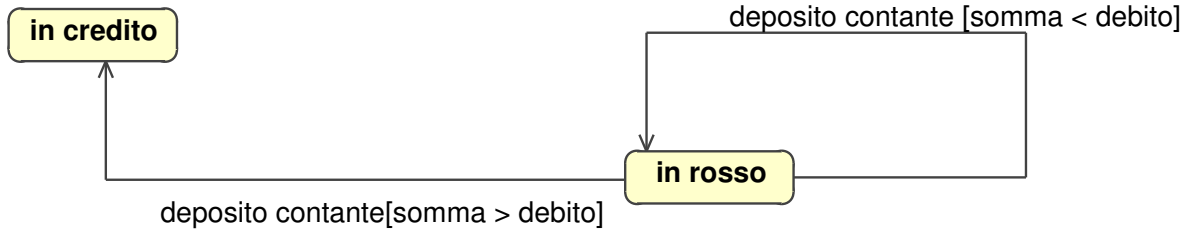


Esempio 2

Diagramma di una istanza di ContoBanca:



Sono possibili anche le transizioni con "guardie":



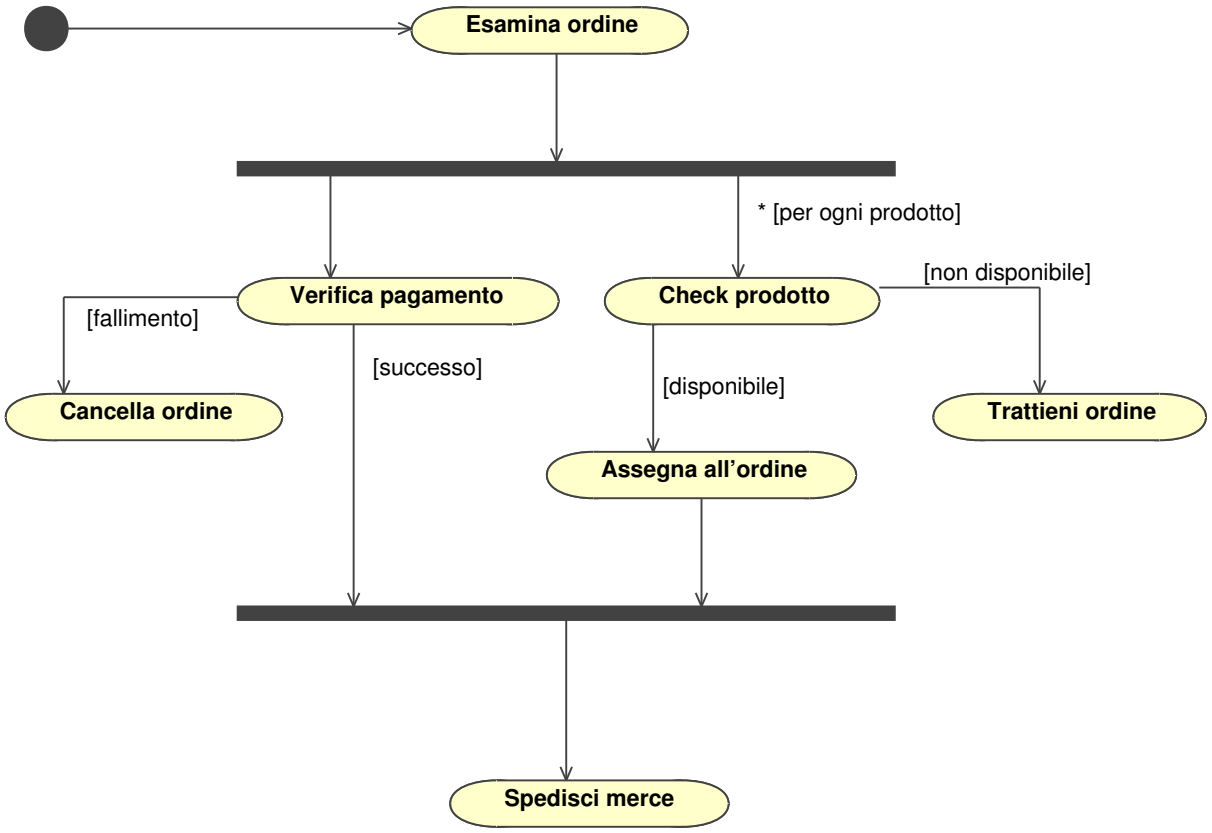
Esercizio/Progetto (Settima parte)

1. Rappresentare il diagramma State per alcune classi rappresentative del progetto in esame.

Diagrammi Activity

- Mostrano il diagramma di flusso delle attività.
- Un solo diagramma per ciascuno Use Case che riassuma tutti i diagrammi *Sequence*.
- Fare riferimento alla bibliografia e alla discussione della lezione 5 per la sintassi di diagrammi Activity.

Esempio 3



Esercizio/Progetto (Ottava parte)

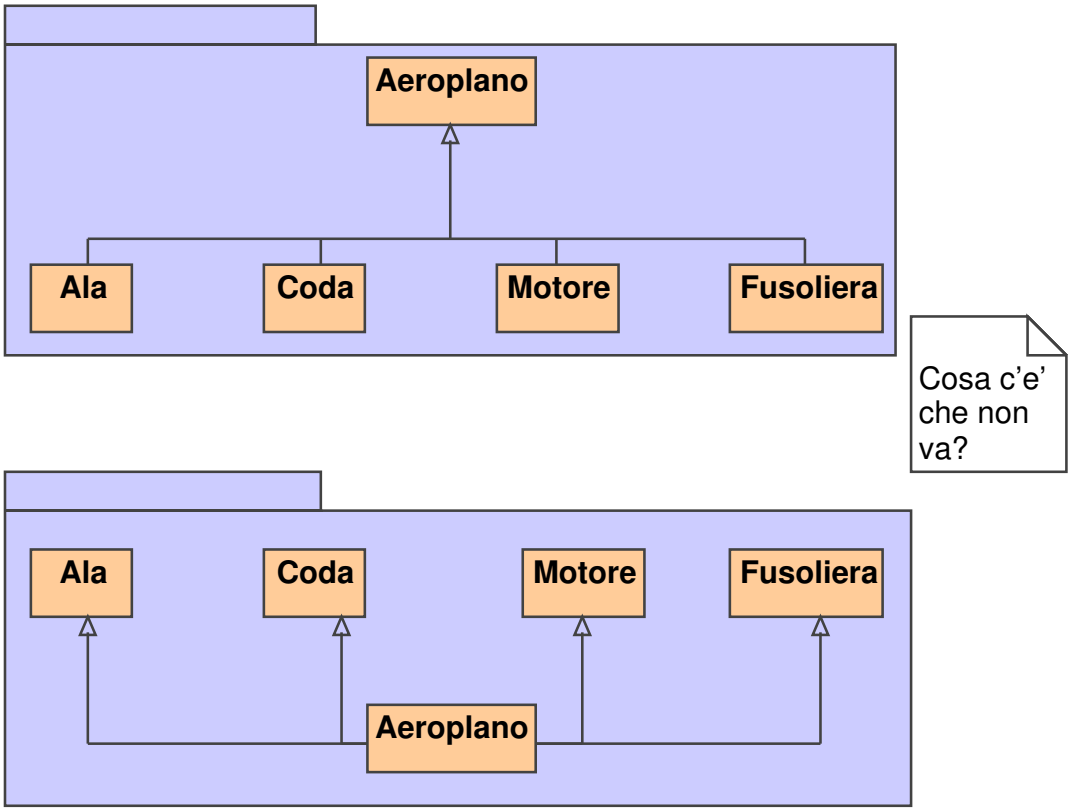
1. Rappresentare il diagramma Activity relativo ad alcuni Use Case rappresentativi del progetto in esame.

Bibliografia 41 / 56

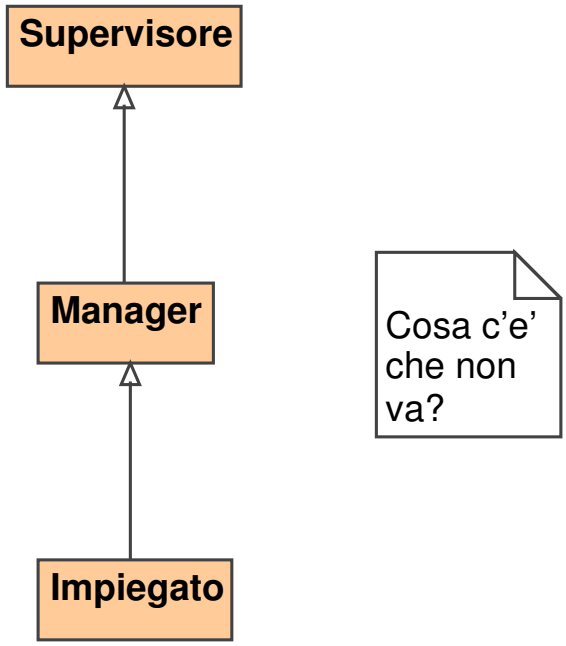
Bibliografia

- Booch G., Rumbaugh J., Jacobson I., *The Unified Modeling Language Reference Manual*. Addison-Wesley.
- Booch G., Rumbaugh J., Jacobson I., *The Unified Modeling Language User Guide*. Addison-Wesley.
- Fowler M. *UML distilled*. Addison-Wesley.

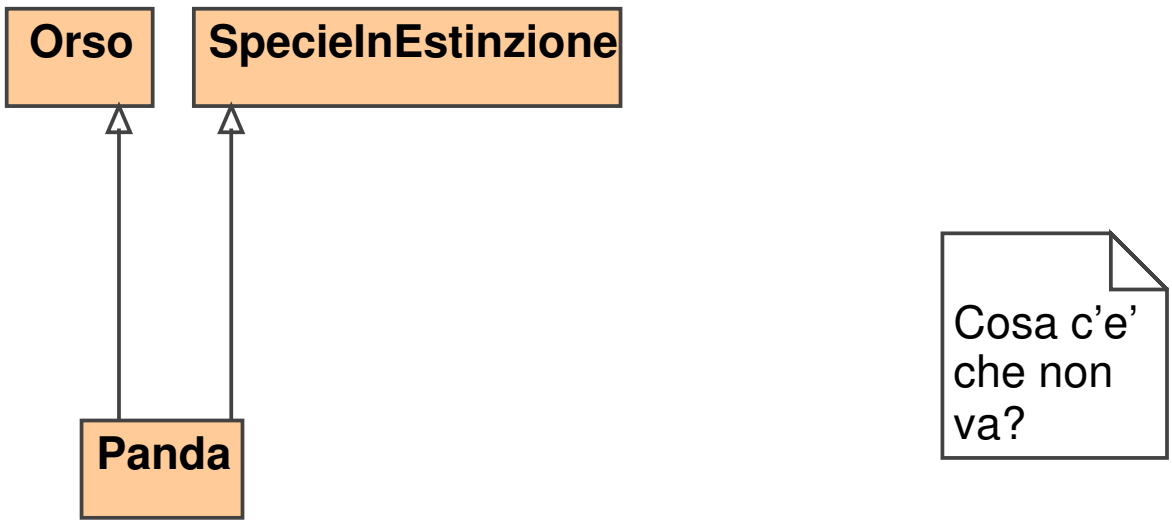
Aggregati


Aeroplano
Ala
Coda
Motore
Fusoliera
Cosa c'e' che non va?
Ala
Coda
Motore
Fusoliera
Aeroplano

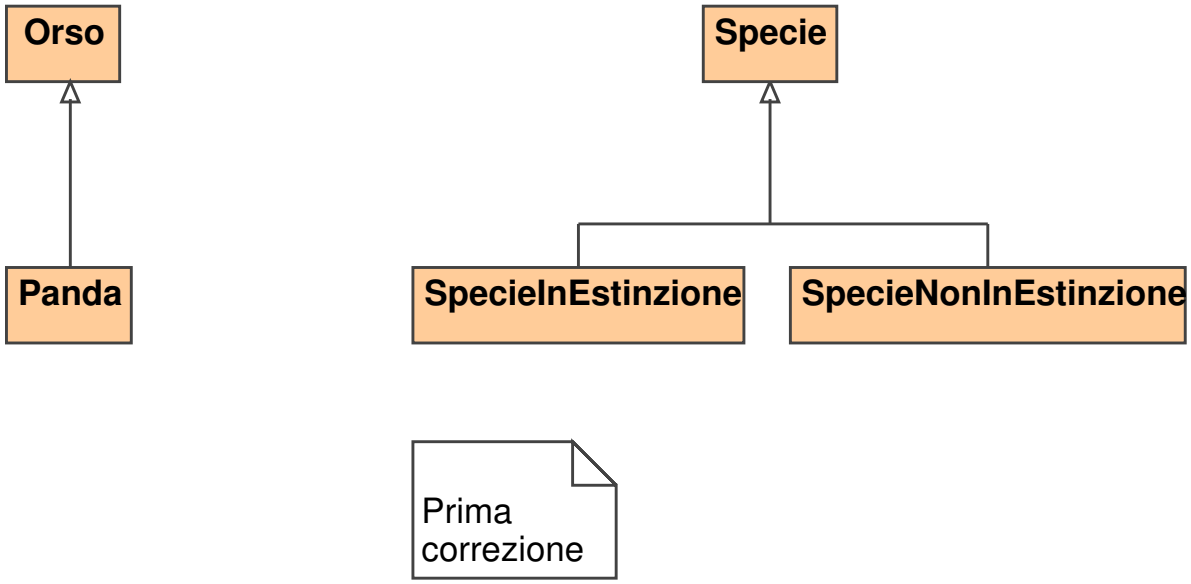
Gerarchie



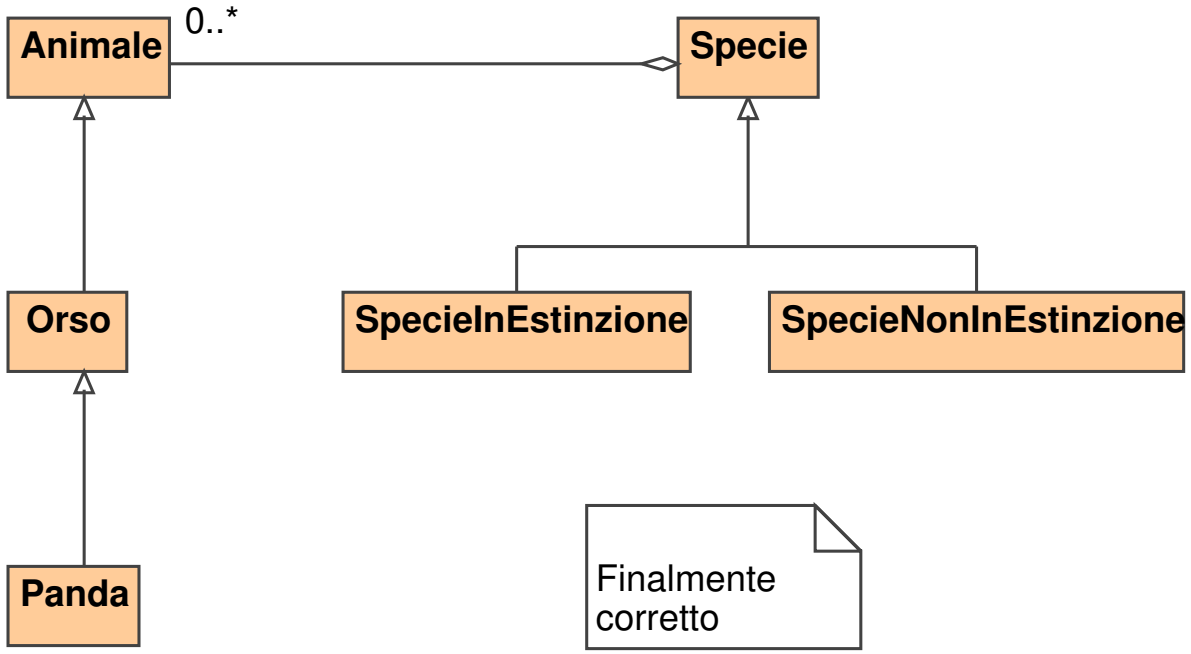
Classi/Istanze 1



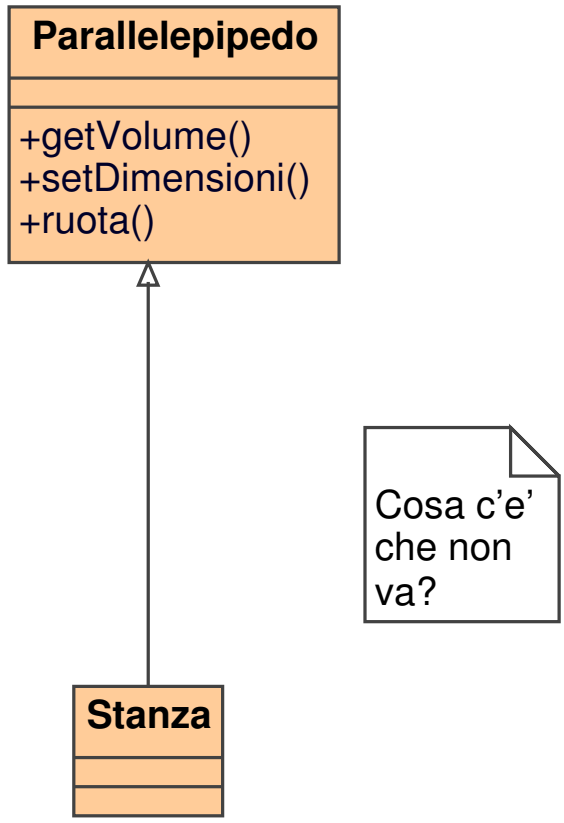
Classi/Istanze 2



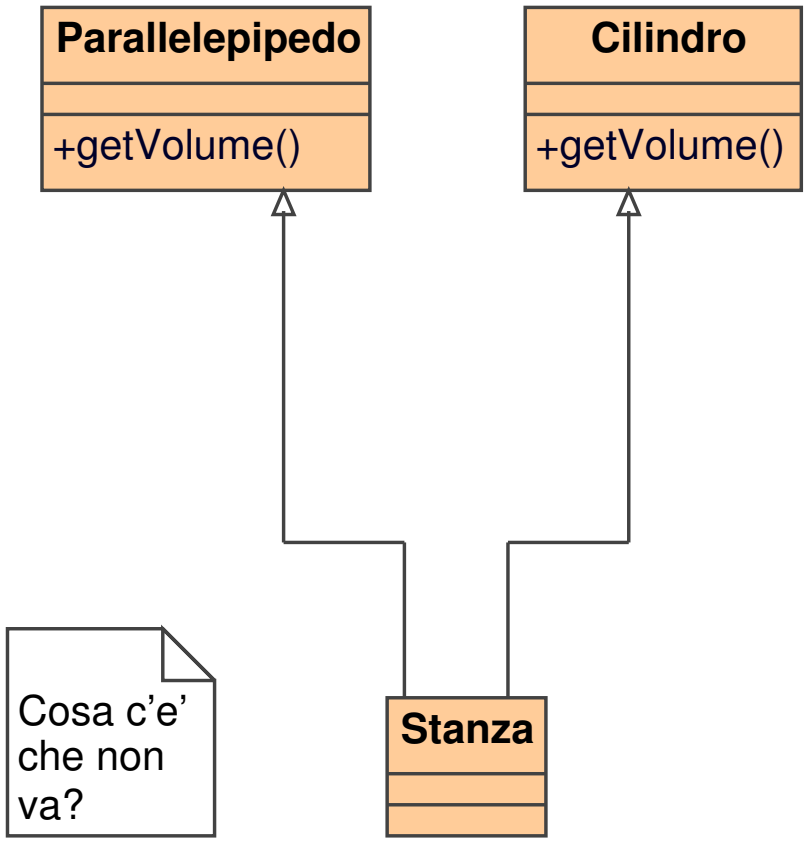
Classi/Istanze 3



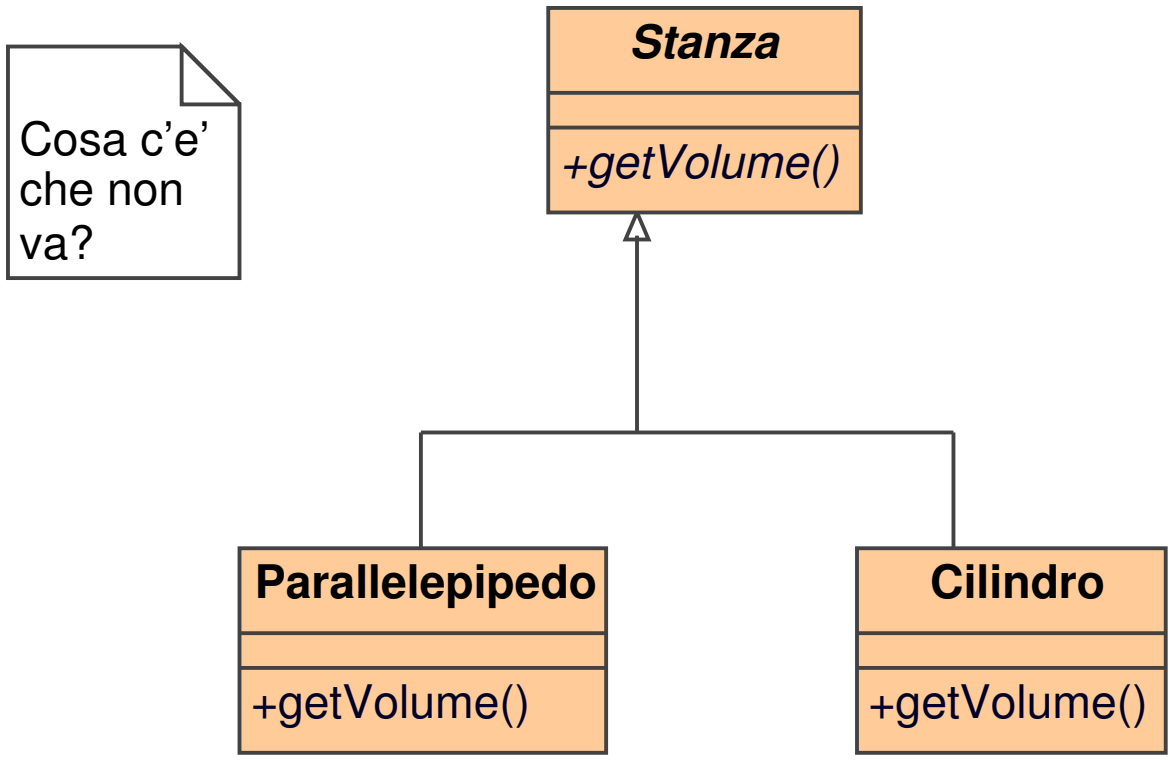
IsA 1


Parallelepipedo
+getVolume()
+setDimensioni()
+ruota()
Cosa c'e' che non va?
Stanza

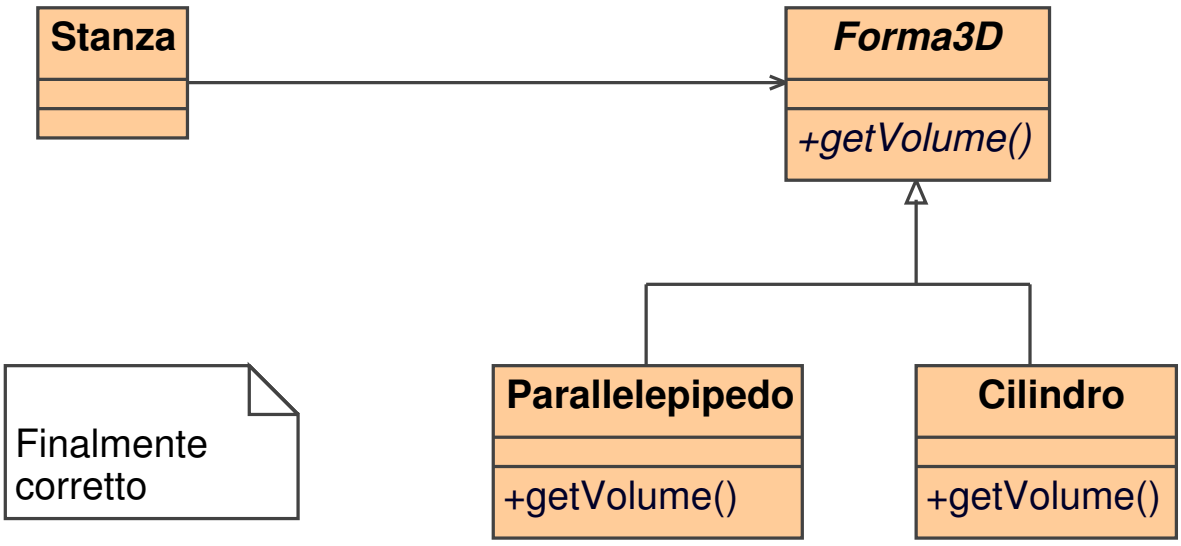
IsA 2


Parallelepipedo
+getVolume()
Cilindro
+getVolume()
Cosa c'e' che non va?
Stanza

IsA 3


Cosa c'e' che non va?
Stanza
+getVolume()
Parallelepipedo
+getVolume()
Cilindro
+getVolume()

IsA 4


Stanza
Forma3D
+getVolume()
Finalmente corretto
Parallelepipedo
+getVolume()
Cilindro
+getVolume()

Appendice B: Codifica Java di relazioni

Generalità

Una composizione è anche una aggregazione, una aggregazione è anche una associazione. Pertanto:

- usare associazioni è sempre corretto;
- bisogna fare attenzione ad usare le altre relazioni;
- in una aggregazione (composizione), una operazione eseguita sull'aggregato (composto) viene propagata su uno o più aggreganti (componenti);
- il tutto deve sapere delle parti, ma non viceversa;
- la composizione implica una relazione a vita:
  - se il tutto è creato, anche le parti sono create,
  - quando il tutto muore, anche le parti muoiono,
  - il costruttore per il tutto deve costruire anche le parti;
- nella aggregazione le parti continuano ad esistere anche dopo la morte del tutto;
- nella semplice associazione le vite degli oggetti, sia pure correlate, esistono in modo totalmente indipendente.

Composizione

```
class Automobile
{
   Motore motore;
   // altri attributi

   Automobile (int cilindrata, int numeroCilindri,
               String marca, String modello)
   {
      motore = new Motore (cilindrata, numeroCilindri);
      // altro codice
   }

   // altri metodi
}
```

Aggregazione

```
class Automobile
{
   Motore motore;
   // altri attributi

   Automobile (Motore motore, String marca, String modello)
   {
      this.motore = motore;
      // altro codice
   }

   // altri metodi
}

// altrove nel codice
Motore m = new Motore (1400, 3);
Automobile a = new Automobile (m, "Seat", "Ibiza");
```


Associazione

```
class Automobile
{
   Persona proprietario;
   // altri attributi

   setProprietario (Persona proprietario)
   {
      this.proprietario = proprietario;
   }

   // altri metodi
}

// altrove nel codice
Automobile a = new Automobile ("Seat", "Ibiza");
Persona io = new Persona ("Marcello", "Sette");
a.setProprietario(io);
```