

# **UML**

**Unified Modeling Language**

# Un approccio visuale alla progettazione

Per quale motivo è utile un approccio visuale alla progettazione?

- Chi progetta un qualsiasi tipo di costruzione o artefatto utilizza sempre figure, schemi, diagrammi per svolgere la propria attività:
  - ingegneri, architetti, ma anche stilisti utilizzano diagrammi e figure per visualizzare i propri progetti
- Anche i progettisti e gli analisti di sistemi informativi utilizzano figure e diagrammi per visualizzare il risultato del loro lavoro:
  - il sistema software

Ciò avviene anche se il tipo di prodotto finale che risulta dalla progettazione non è necessariamente visuale

## **Vantaggi dell'utilizzo di diagrammi nella fase di progettazione**

- Sia che si progetti un edificio sia che si progetti un sistema software il progettista ha la necessità di rappresentare i diversi aspetti del progetto
  - si utilizzano diagrammi differenti, ognuno focalizzato su uno o più aspetti
- Nel caso dell'edificio
  - alcuni disegni rappresentano una visione completa da diversi punti di vista
  - altri alcuni particolari come ad esempio gli impianti tecnologici
- Allo stesso modo per un sistema informativo
  - un diagramma può rappresentare i collegamenti tra le componenti
  - altri alcuni particolari come ad esempio la sequenza delle comunicazioni tra le componenti
- Si tratta di applicare il concetto di “astrazione” attraverso il quale una realtà anche molto complessa viene rappresentata semplificandola in un *modello*

## *Generazione dei modelli*

- Durante le fasi di analisi e progettazione vengono generati dei modelli che consentono di identificare e separare le caratteristiche (utili al progetto) di un sistema reale (ad esempio una classe che modella l'oggetto cliente).
- Il progettista dovrà quindi decidere quali caratteristiche sono rilevanti per il sistema che sta costruendo, inserirle nel modello e definire le relazioni tra gli elementi del modello

## *Tipi di relazione*

- Vi sono diversi tipi di relazione da considerare:
  - Strutturali, tra elementi interdipendenti  
(ad esempio associazioni tra classi)
  - Temporal, per rappresentare sequenze di eventi nel tempo  
(ad esempio messaggi sequenziali in un diagramma di interazione)
  - Causa-effetto, per definire precondizioni ad una determinata funzione  
(ad esempio stati di un diagramma di stato)
  - Organizzative, per raggruppare opportunamente elementi del sistema  
(ad esempio package di elementi)
  - Evolutive, per rappresentare le derivazioni tra elementi del modello  
nel tempo  
(ad esempio dipendenze tra diagrammi del modello)

# Cos'è UML

- **Unified Modeling Language (UML) è un linguaggio di modellazione visuale**

- *E' uno strumento per analisti e progettisti di sistemi orientati agli oggetti che consente di modellare, rappresentare e documentare sistemi software*
- Non è un linguaggio di programmazione.
- Vi sono strumenti di CASE che possono generare codice in diversi linguaggi a partire da modelli UML
  - si tratta del codice di struttura di oggetti che poi richiedono da parte dello sviluppatore la scrittura manuale del codice che implementa i metodi.
- UML non è una metodologia di sviluppo del software
- Molte metodologie di analisi e progettazione, pur mantenendo diversi procedimenti, hanno standardizzato la loro notazione per rappresentare visivamente i modelli di un sistema con UML

# Cos'è UML

- UML è un linguaggio, un insieme di elementi e di regole, di specifica formale.
- Gli elementi sono forme grafiche (linee, rettangoli, ecc.) che rappresentano ciò che si sta modellando.
- Le regole che spiegano come combinare gli elementi sono di tre tipi:
  - sintassi astratta, espressa tramite diagrammi e linguaggio naturale
  - regole sintattiche, espresse tramite Object Constraint Language (OCL) e linguaggio naturale
  - semantica, espressa in linguaggio naturale con il supporto di diagrammi

## Processo Unificato di sviluppo del software

- UML è un linguaggio che ha lo scopo di definire in modo formale un sistema
- Gli sviluppatori di UML hanno messo a punto per gli sviluppatori di sistemi anche un modo di procedere nel processo di sviluppo utilizzando UML:

– il **Processo Unificato di sviluppo del software**  
(Unified Software Development Process – USDP)

- *Il Processo Unificato coinvolge persone, progetti, strumenti, processi, prodotti: i partecipanti e gli sviluppatori coinvolti nel progetto di sviluppo di un sistema, seguono un determinato processo, utilizzando strumenti di ausilio nello sviluppo, generando prodotti software*



## Caratteristiche del Processo Unificato

- Il processo parte dai requisiti dell'utente, che vengono raccolti nei cosiddetti **“casi d'uso”**, delle sequenze di esecuzione del sistema in grado di fornire un valore all'utente
- Dai casi d'uso gli sviluppatori producono i modelli del sistema e le implementazioni che li realizzano
- Le caratteristiche del processo:
  - architetto-centrico: l'architettura del sistema viene sviluppata in modo da soddisfare i requisiti dei casi d'uso più importanti (piattaforma e struttura, sottosistemi)
  - iterativo: il progetto viene scomposto in sottoprogetti che costruiscono parti del sistema finale
  - incrementale: il sistema viene costruito incrementalmente unendo le singole parti sviluppate nei sottoprogetti

## Caratteristiche del Processo Unificato

- Fasi del ciclo di vita (un progetto può essere costituito da più cicli):
  - *Inizio, elaborazione, costruzione, transizione* (ogni fase può essere costituita da più iterazioni)
- I prodotti intermedi del processo sono i modelli (astrazioni delle caratteristiche del sistema).
- Ogni modello definisce il sistema da un certo punto di vista, i principali:
  - dei Casi d'Uso
  - di Analisi
  - di Progetto
  - di Deployment
  - di Implementazione
  - di Test

## Struttura di UML

- In un approccio top-down all'UML si possono distinguere gli elementi fondamentali della sua struttura:
  - Viste, Diagrammi, Elementi del modello
- Le viste mostrano i differenti aspetti di un sistema attraverso la realizzazione di un certo numero di diagrammi
  - si tratta di astrazioni, ognuna delle quali analizza il sistema da modellare con un'ottica diversa (funzionale, non funzionale, organizzativa, ecc.), la somma di queste viste fornisce il quadro d'insieme
- I diagrammi permettono di esprimere le viste logiche per mezzo di grafici
  - vi sono diversi tipi di diagrammi destinati ad essere utilizzati ognuno per una particolare vista
- Gli elementi del modello sono i concetti che permettono di realizzare i vari diagrammi
  - indicano gli attori, le classi, i packages, gli oggetti, ecc.

# Diagrammi UML

• *I diagrammi di UML sono dei grafici che visualizzano una particolare proiezione del sistema analizzato da una specifica prospettiva*

- Use Case Diagram (Diagramma dei casi d'uso)
- Class Diagram (Diagramma delle classi)
- Object Diagram (Diagramma degli oggetti)
- Sequence Diagram (Diagramma di sequenza)
- Collaboration Diagram (Diagramma di collaborazione)
- Statechart diagram (diagramma degli stati)
- Activity Diagram (Diagramma delle attività)
- Component diagram (diagramma dei componenti)
- Deployment diagram (diagrammi di dispiegamento)

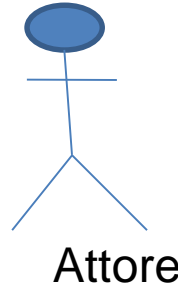
# Casi d'uso

- I casi d'uso costituiscono un ottimo strumento per ottenere una visione d'insieme del sistema che si stà analizzando;
  - Sono importanti nelle prime fasi del progetto.
- Rappresentano una vista esterna (utilizzatore) del sistema e sono finalizzati a modellare il dialogo tra utilizzatore e sistema:
  - descrivono l'interazione tra attori e sistema, non la "logica interna" della funzione
  - sono espressi in forma testuale, comprensibile anche per i non "addetti ai lavori"
  - possono essere definiti a livelli diversi (sistema o parti del sistema)
  - rappresentano le modalità di utilizzo del sistema da parte di uno o più utilizzatori (attori)
- Ragionare sui casi d'uso aiuta a scoprire i *requisiti funzionali*:
  - i casi d'uso possono essere un valido ausilio nel dialogo con l'utente ed in generale con esponenti non tecnici del progetto

## **Diagrammi dei casi d'uso**

- I diagrammi dei casi d'uso rappresentano i casi d'uso stessi, gli attori e le associazioni che li legano.
- Rappresentano le modalità di utilizzo del sistema da parte di uno o più utilizzatori che vengono definiti attori
- I singoli use case non descrivono la “logica interna” delle funzioni ma si occupano di descrivere le iterazioni tra sistema e attori.
- L'attore è un'entità esterna al sistema che fornisce lo stimolo a cui il sistema risponde

# Attore



- L'attore è un utilizzatore del sistema (essere umano oppure altro sistema) che interagisce con i casi d'uso
- Rappresenta un ruolo di un oggetto oppure un oggetto esterno al sistema che interagisce con lo stesso come parte di un unità di lavoro (work unit) a realizzare un use case
- Un oggetto fisico (o classe) può giocare più ruoli differenti ed essere modellato da diversi attori
- Esempi di attore per un sistema di prenotazioni: agente di viaggi, addetto check-in, ecc.
- E' bene ricordare nel caso di attori umani che il nome di un attore identifica il ruolo che l'attore svolge in relazione al sistema, non il suo incarico.
  - Esempio: un impiegato (incarico) gestisce ed archivia un documento svolgendo il ruolo di protocollatore

## Use case

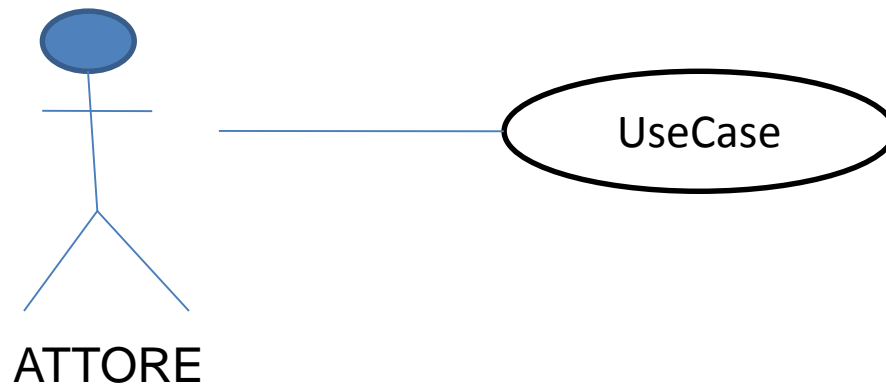


UseCase

- Il caso d'uso è una particolare modalità di utilizzo del sistema
- Rappresenta alcune funzioni visibili all'attore
- Raggiunge alcuni obiettivi per l'attore, in concreto può essere una scrittura, lettura, modifica di informazioni
  - Esempio per un sistema di prenotazioni: checking per un volo, assegnazione di un posto, ecc.

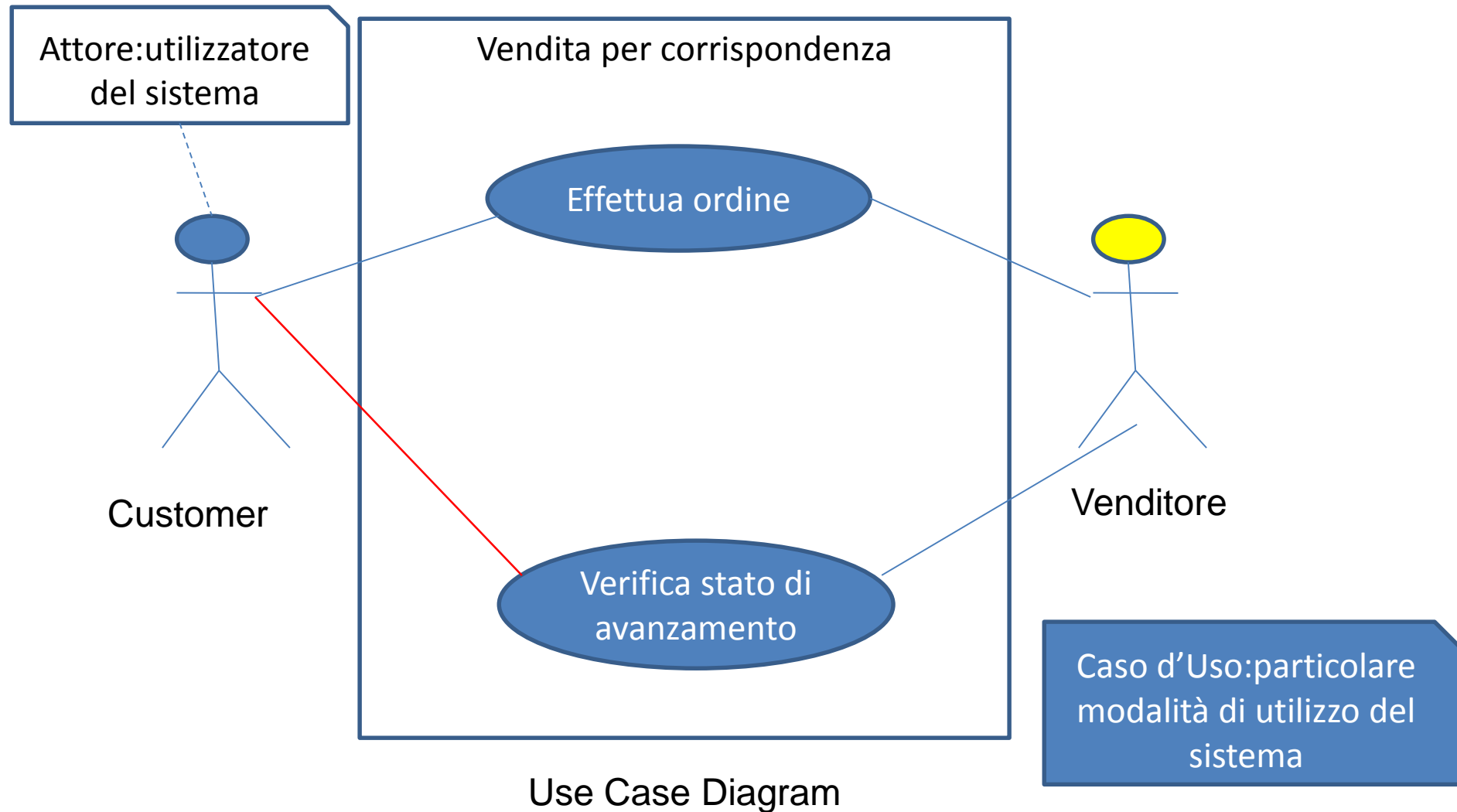


# Relazione tra attore e use case

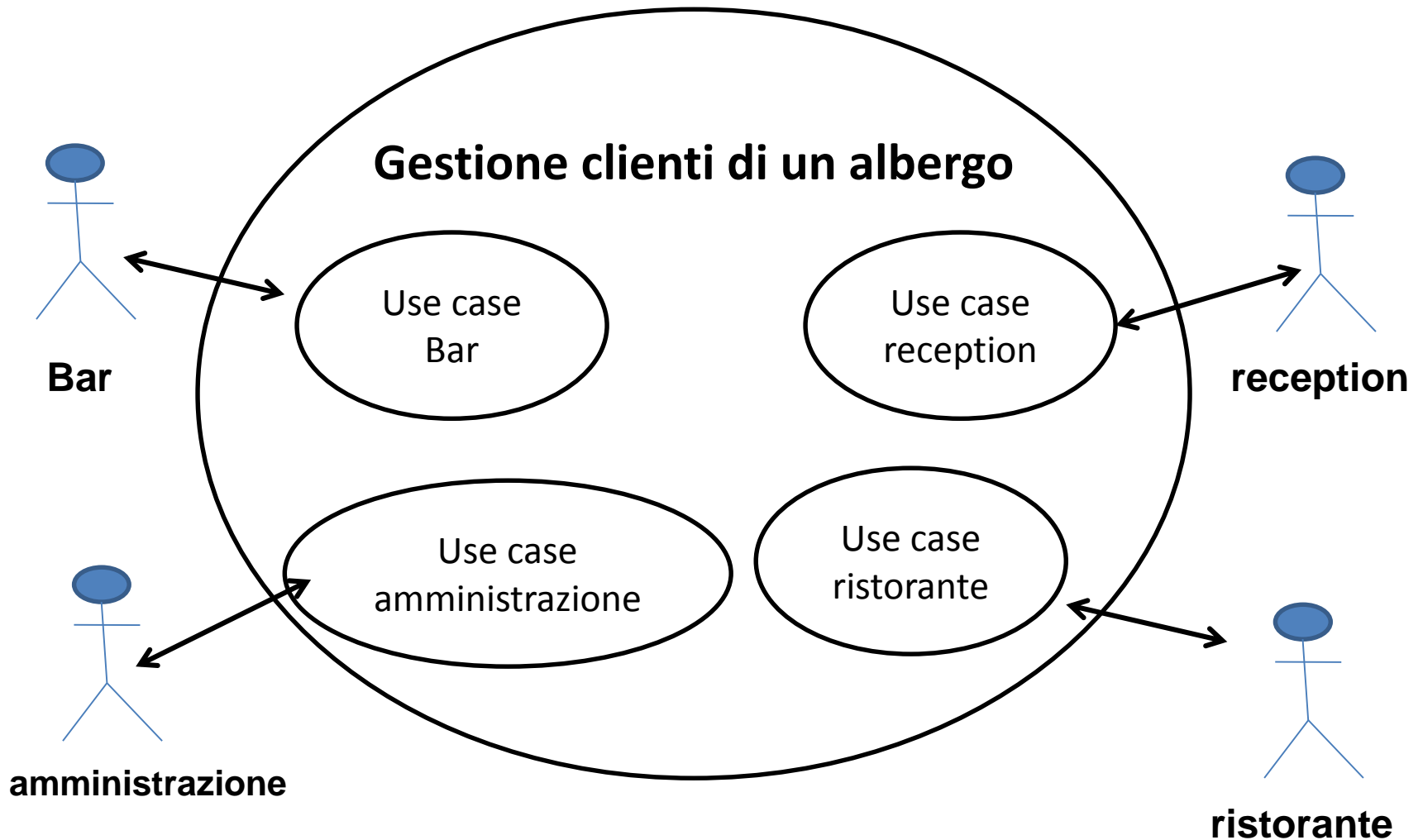


- La relazione indica l'esistenza di un'associazione tra un attore ed un caso d'uso
- *Quindi una particolare persona (o sistema) che si trova in un certo ruolo comunica con specifiche istanze del caso d'uso, partecipando agli eventi rappresentati dal caso*
- In concreto poi il caso d'uso è realizzato come una funzione software utilizzata dagli attori trattando (inserendo o ricevendo) informazioni
  - Nel diagramma gli attori sono collegati ai casi d'uso con i quali interagiscono attraverso una linea che rappresenta la relazione tra loro esistente

# Esempio di caso d'uso



# Gli Attori e gli Use Cases



## Specifica comportamentale

- Ogni caso d'uso costituisce una sequenza di attività che generano un risultato per l'attore che con esso interagisce
- La sequenza di attività viene descritta in una specifica comportamentale
  - può consistere in un diagramma di sequenza, o di collaborazione, o di stato, oppure in istruzioni di un linguaggio di programmazione
- Normalmente viene prodotta una specifica informale nella forma di descrizione del caso d'uso
- La descrizione dei casi d'uso non ha una sintassi formale, a differenza di altri diagrammi UML con precise regole sintattiche, che rendono possibile la simulazione del sistema e la conversione del modello in un programma (C++ oppure Java)

## Descrizione del caso d'uso

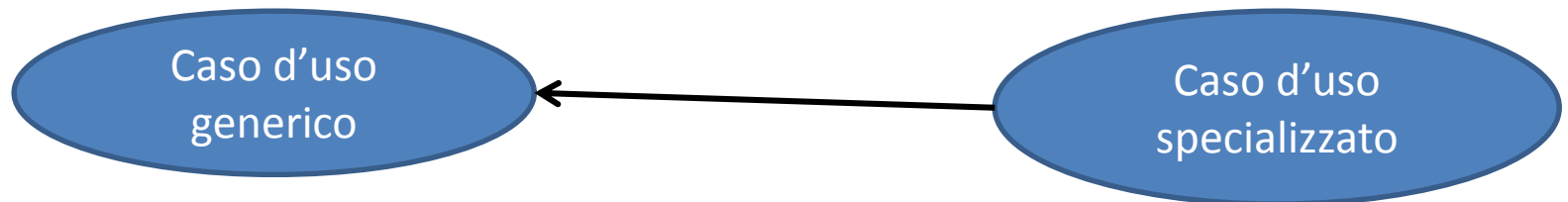
- La scelta delle modalità di scrittura della descrizione è al progettista
  - normalmente in un certo contesto (ad es. software house) vi sono delle regole alle quali attenersi
- I due approcci più diffusi per la descrizione:
  - uno o più paragrafi che descrivono la sequenza di attività che si verifica nel caso d'uso
  - elencare, su due colonne, le attività compiute dall'attore e le risposte date dal sistema a tali attività

## Altri tipi di associazioni e relazioni

- Altri tipi di associazioni e relazioni rappresentabili nel diagramma dei casi d'uso:
  - generalizzazione tra casi d'uso
  - generalizzazione tra attori
  - relazione di inclusione (include) tra casi d'uso
  - relazione di estensione (extends) tra casi d'uso

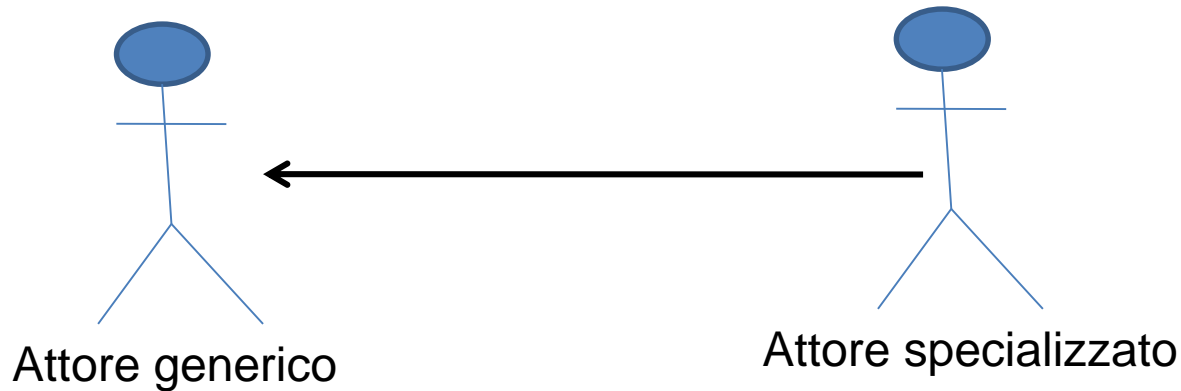
## Generalizzazione tra casi d'uso

- Può esistere più di una versione di un caso d'uso, aventi ognuna alcune azioni in comune ed altre uniche per ciascun caso
- Nel diagramma l'associazione di generalizzazione viene indicata da una freccia puntata verso il caso d'uso più generale
- Il caso d'uso specializzato eredita (vedi diagrammi delle classi) parte delle funzionalità dal caso d'uso generico



# Generalizzazione tra attori

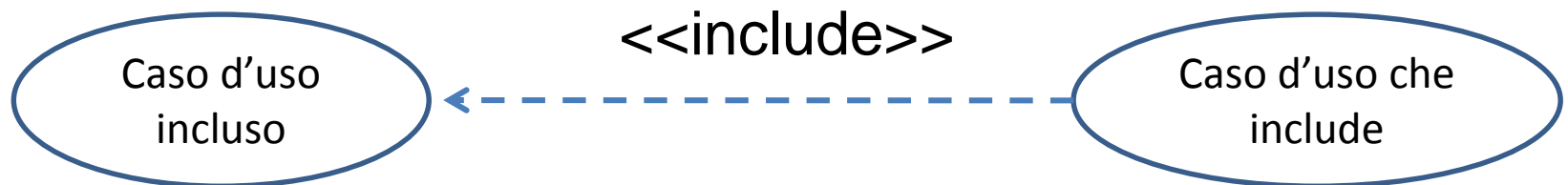
- L'attore specializzato eredita parte delle caratteristiche dell'attore generico





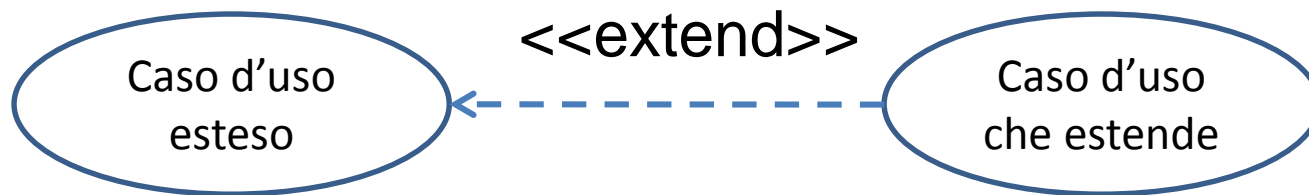
## Relazione di inclusione (include) tra casi d'uso

- Un caso d'uso comprende le funzionalità di un altro caso



## Relazione di estensione (extend) tra casi d'uso

- Un caso d'uso può essere esteso nella sua funzionalità ad un altro caso



## Come produrre casi d'uso

- Nelle prime fase di progetto il progettista/analista lavora tipicamente su appunti e trascrizioni da interviste
- Di norma il processo di analisi e definizione dei casi d'uso avviene per fasi
  - Definire attori e casi d'uso  
tipiche domande alle quali dare risposta:
    - *Chi sono le persone che utilizzeranno questo sistema per inserire informazioni?*
    - *Chi sono i destinatari delle informazioni fornite dal sistema?*
    - *Quali altri sistemi interagiscono con questo?*
  - Organizzare secondo ordine di priorità i casi d'uso
    - I casi d'uso più importanti vanno sviluppati prima
  - Sviluppare ciascun caso d'uso (secondo priorità)
    - Generare la specifica dettagliata di ogni caso d'uso (analista del sistema)
  - Strutturare il modello dei casi
    - Aggiunta della struttura al diagramma, attraverso generalizzazione, inclusione, estensione ed attraverso il raggruppamento dei casi in “package”

# **L'analisi dei requisiti**

# I requisiti

- **Esprimono la volontà del committente**
- **Definiscono i confini del sistema applicativo**
- **Costituiscono il punto di partenza dell'analisi**
- **Sono espressi con:**
  - **Linguaggio naturale**
  - **Linguaggio formale**
  - **Grafici**
  - **prototipi**

# **Modello dei requisiti**

- **USE CASE**
- **PROTOTIPO DELL'INTERFACCIA**
- **INDIVIDUAZIONE DELLE CLASSI CANDIDATE**

# Specificare i requisiti

Dall'osservazione del business, dalle interviste con l'utente specificare:

- I requisiti del sistema

Attraverso la tecnica degli use case

per ogni rilevante evento di business descrivere uno o più Use Case

Le informazioni relative al sistema applicativo si raccolgono in genere attraverso interviste, osservando le attività svolte, oppure leggendo i documenti disponibili.

È necessario stabilire cosa il sistema dovrà automatizzare.

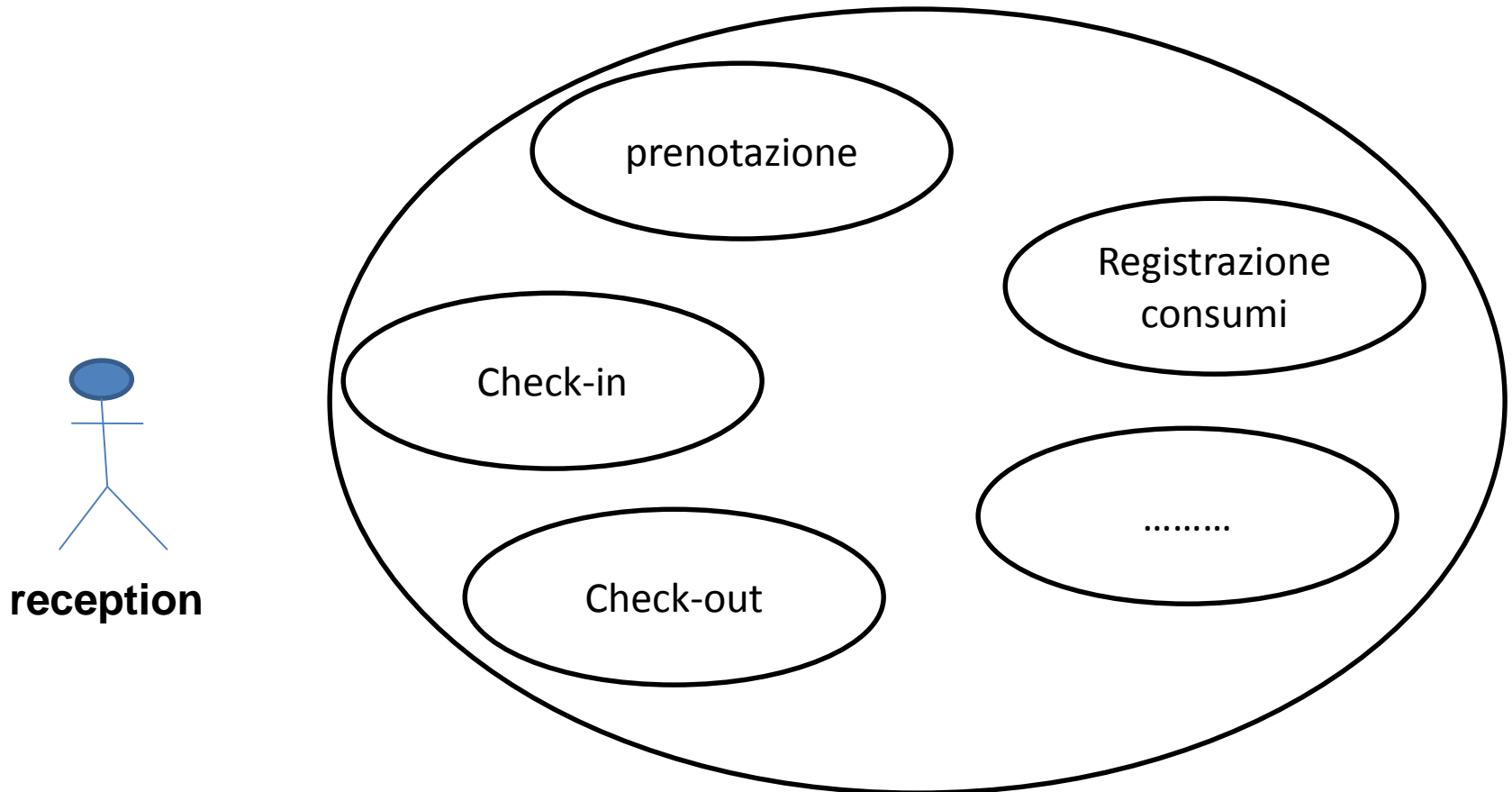
Verrà esaminato il concetto di EVENTO. Gli USE CASE sono le funzionalità che lo soddisfano.

Bisogna quindi individuare correttamente lo SCENARIO tradotto normalmente con il termine

**PROBLEM DOMAIN**

## Gli attori e gli Use Cases

Rappresentazione di una serie di Use Cases destinati all'attore RECEPTION. Il numero di use case è determinato dal numero di funzionalità necessarie a svolgere i compiti.





# Evento

- Accadimento che determina potenzialmente una risposta del sistema
- Di due tipi
  - Esterno
    - Accade all'esterno del sistema
  - Temporale
    - Legato ad una precisa scadenza

## **Esempio albergo**

### **Lista degli eventi**

1. L'addetto al ristorante registra i costi dei pasti
2. L'addetto al bar registra le consumazioni
3. La reception registra i consumi
4. La reception effettua il check-in
5. La reception effettua il check-out
6. La reception immette una prenotazione
7. L'amministrazione richiede statistiche

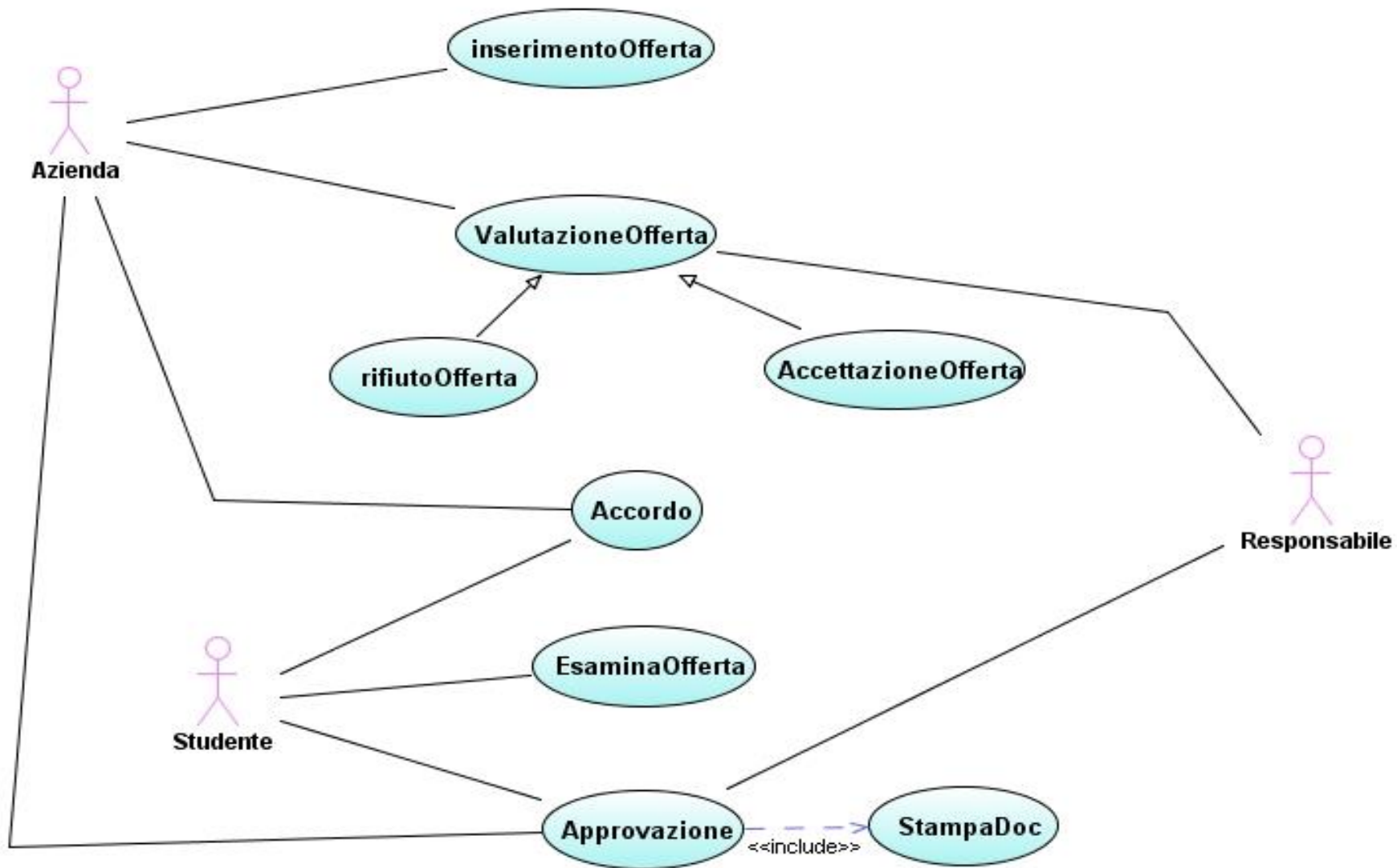
## Esercizio 1°

Modelliamo il sistema di gestione dei tirocini di una università

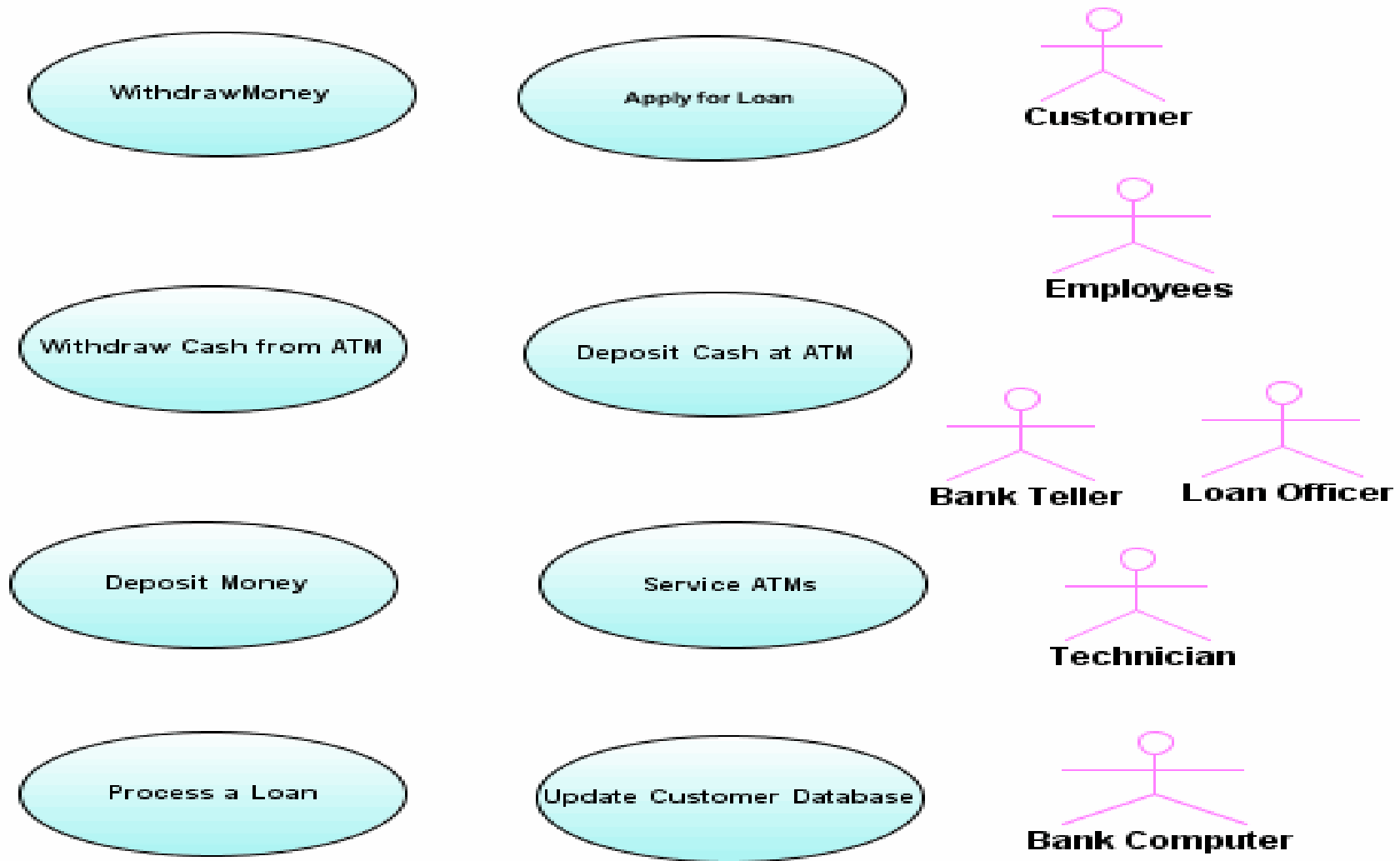
Siamo interessati a rappresentare i requisiti utente, quindi usiamo gli use case diagram.

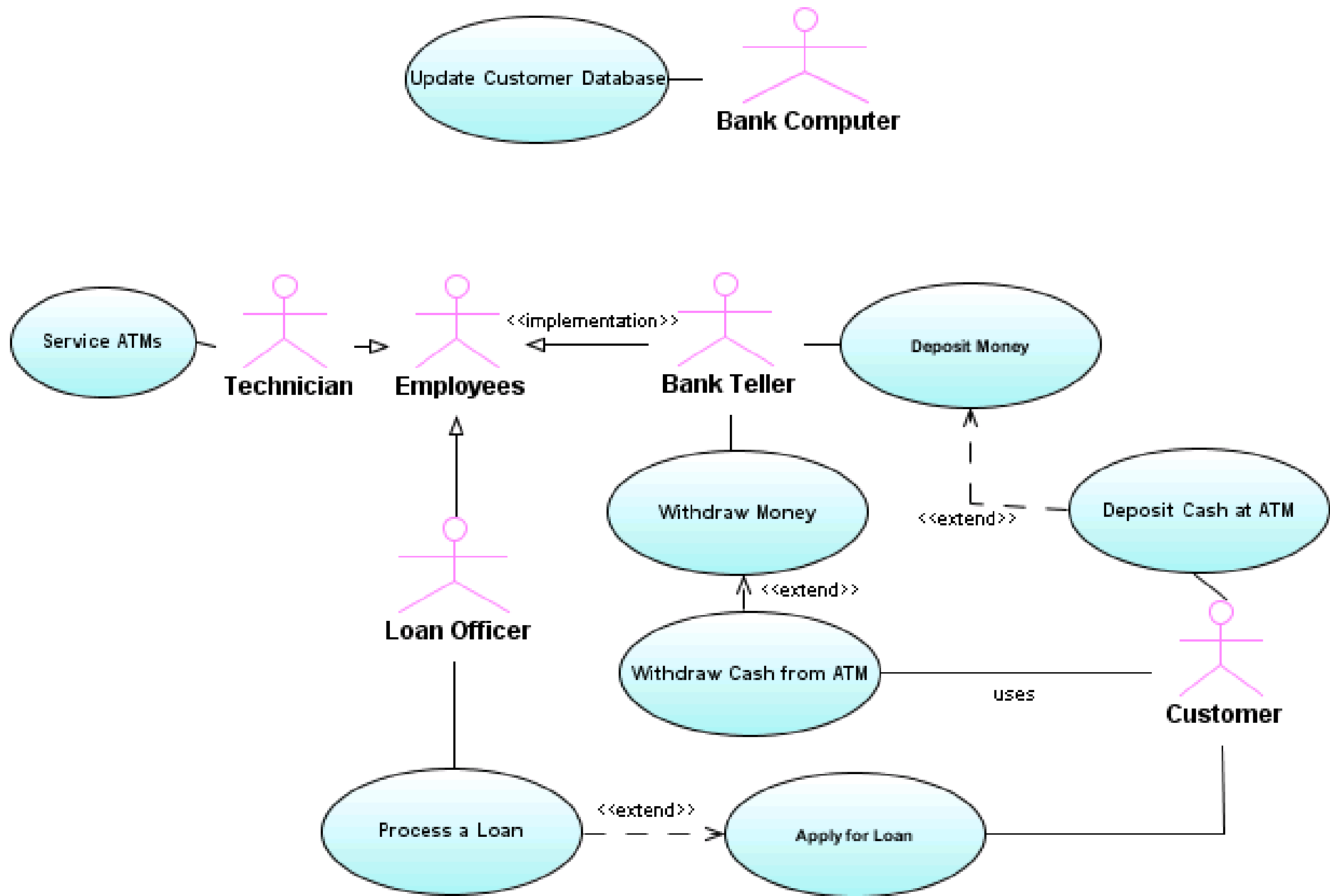
# Gestione tirocini

- Le aziende interessate producono delle offerte di tirocinio.
- Il responsabile dei tirocini approva o rifiuta le offerte.
  - I rifiuti sono notificati all'azienda proponente.
  - Le offerte accettate diventano visibili agli studenti.
- Gli studenti visualizzano le offerte. In conseguenza di ciò possono accordarsi con l'azienda proponente.
- Le aziende assegnano gli studenti graditi ai tirocini offerti.
- Il responsabile dei tirocini approva o rifiuta gli accoppiamenti studente-tirocinio.
  - In caso di accettazione si stampa l'accordo che dovrà essere firmato da tutte le parti in causa.



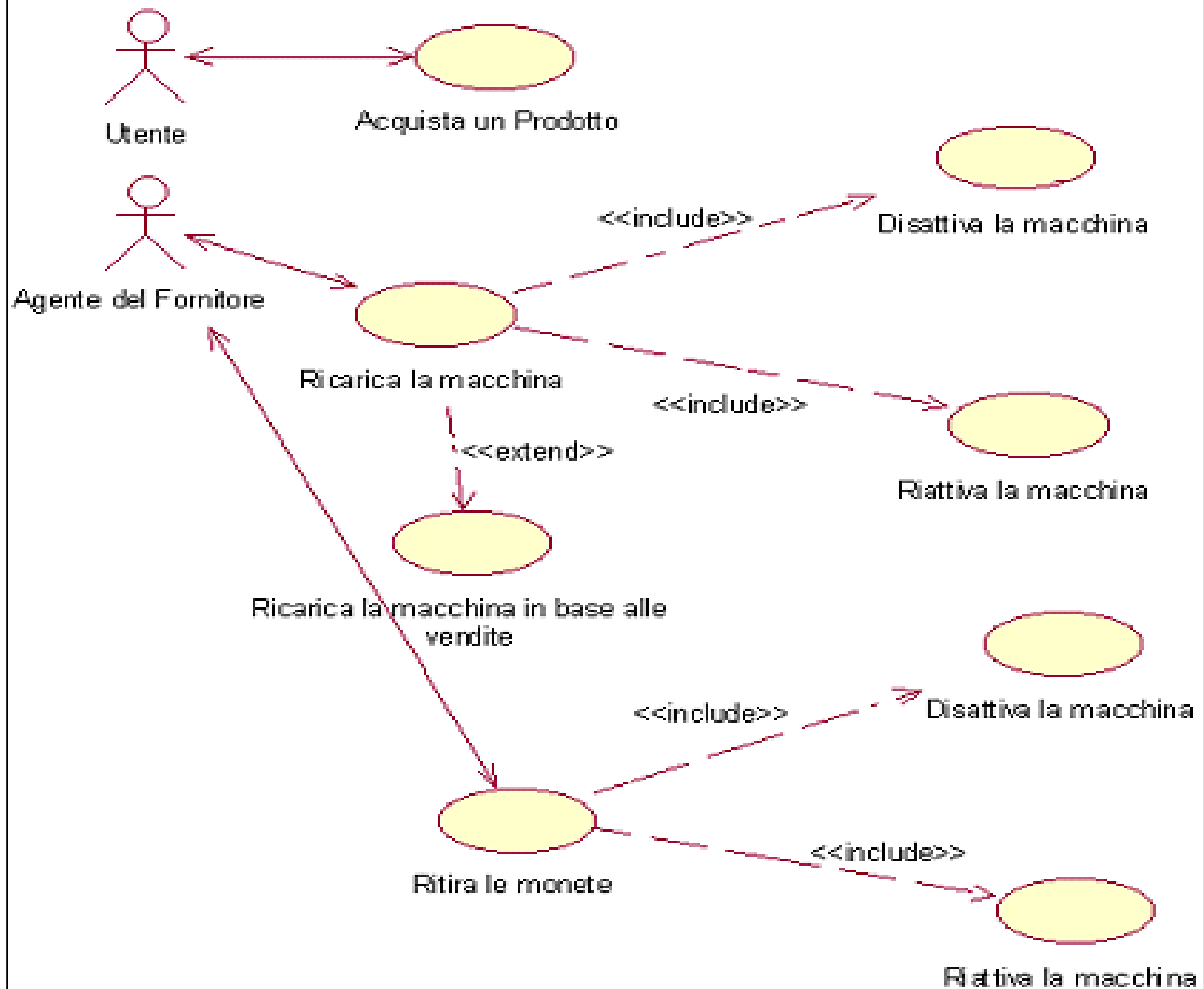
Use Case:  
Banca-Bancomat







Use case macchina  
self-service



## Diagrammi delle classi

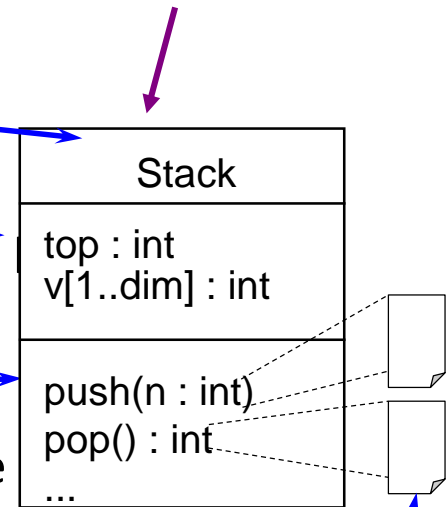
- I diagrammi delle classi rappresentano le classi e gli oggetti, con i relativi attributi ed operazioni, che compongono il sistema
- *L'obiettivo dei diagrammi delle classi è visualizzare la parte statica del sistema*
- Il diagramma delle classi specifica, mediante le associazioni, i vincoli che legano tra loro le classi
- Può essere definito a diversi livelli (analisi, disegno di dettaglio).
- Può rappresentare diverse tipologie di oggetti (boundary, control, entità...)

# Struttura di una Classe

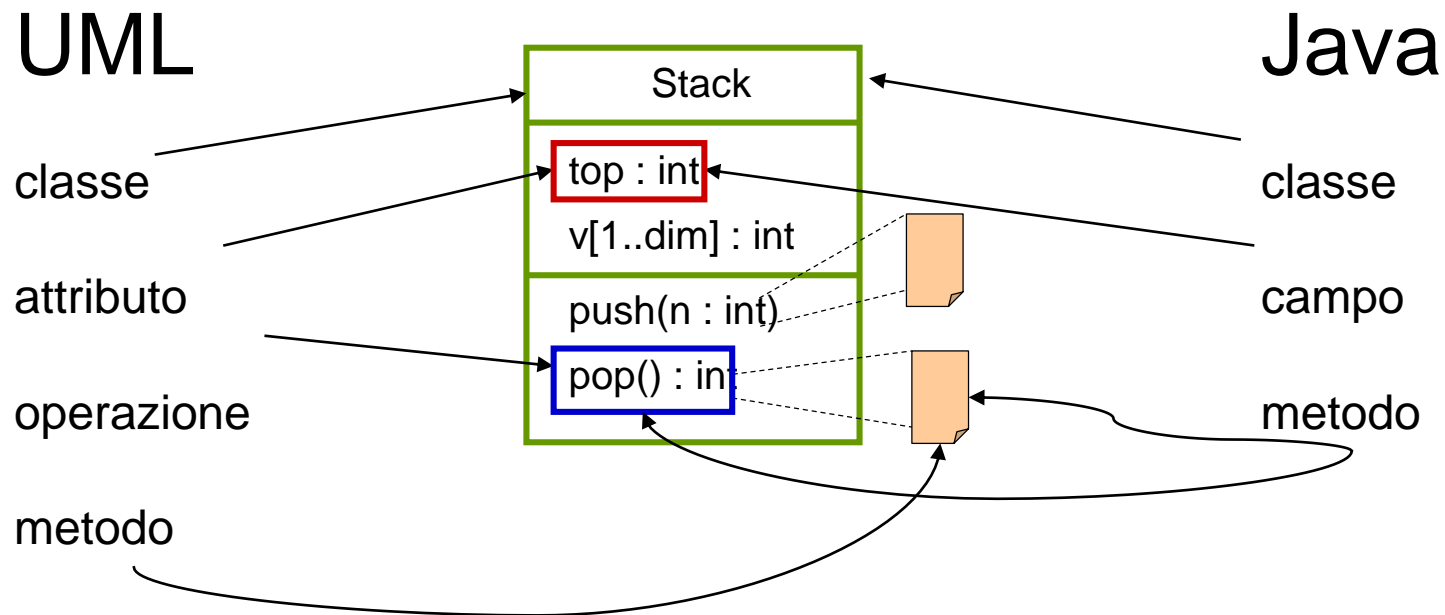
- Una classe è caratterizzata da:

- un IDENTIFICATORE
  - identifica univocamente la classe
- un insieme di ATTRIBUTI
  - rappresentano le strutture dati che determinano istanze della classe
- un insieme di OPERAZIONI
  - identificano il comportamento esterno della classe
- un insieme di METODI
  - rappresentano l'implementazione delle operazioni.

rappresentazione  
di una classe in UML



# Terminologia UML e Terminologia Java



# Esempio di una Classe Conto Corrente e di Due Istanze

classe

ContoCorrente
correntista numeroConto saldo tassoAttivo tassoPassivo
apri() chiudi() deposita() preleva() stampaSaldo() variaTassoAttivo() variaTassoPassivo()

istanze

<u>RossiContoCorrente : ContoCorrente</u>	
correntista	<i>"Mario Rossi"</i>
numeroConto	<i>553430</i>
saldo	<i>9870000</i>
tassoAttivo	<i>7</i>
tassoPassivo	<i>18</i>

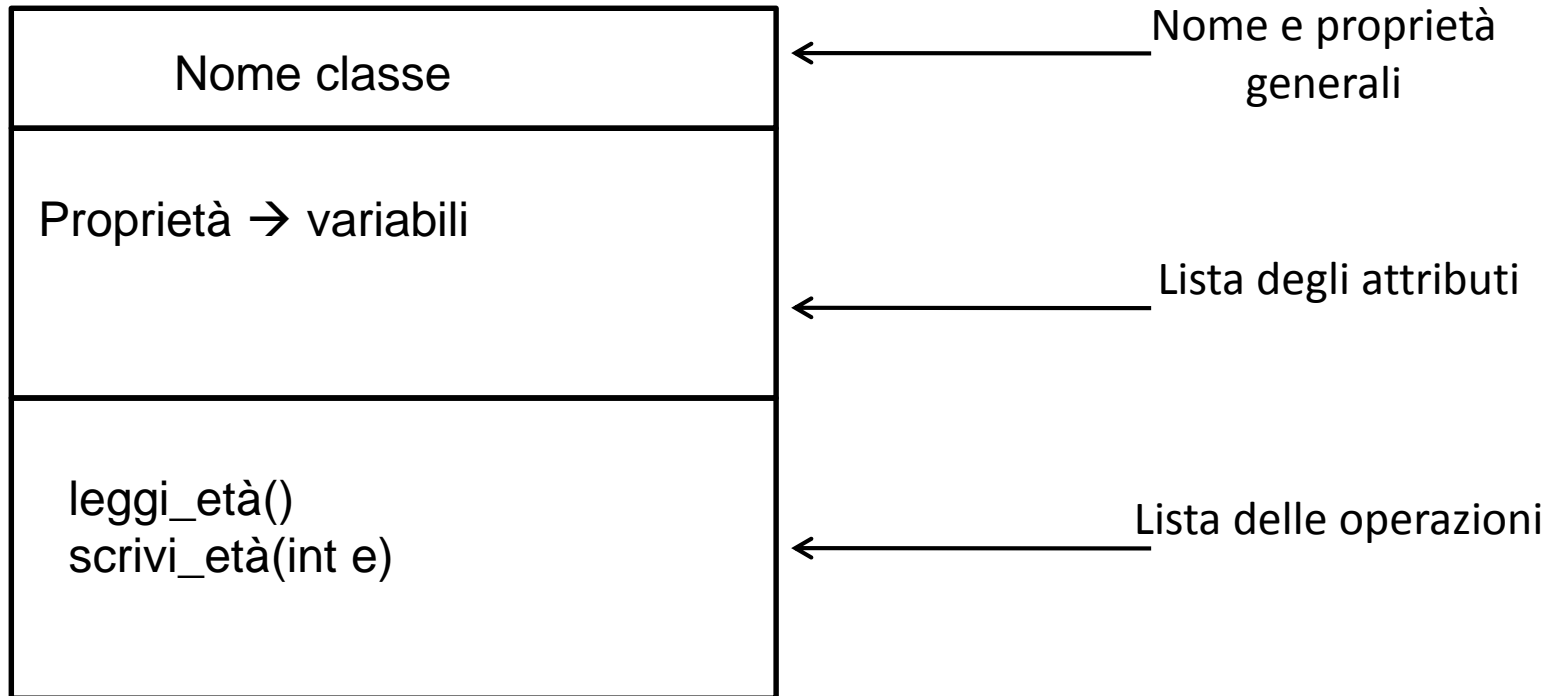
<u>BianchiContoCorrente : ContoCorrente</u>	
correntista	<i>"Carlo Bianchi"</i>
numeroConto	<i>144118</i>
saldo	<i>30143500</i>
tassoAttivo	<i>10</i>
tassoPassivo	<i>18</i>

# Il diagramma delle classi

Gli elementi contenuti in un diagramma delle classi possono essere:

- Classi
- Interfacce
- Relazioni
- package

# Il diagramma delle classi





# Il diagramma delle classi

Il nome di una classe può essere semplice, cioè solo il nome, oppure può essere corredata dalla lista dei package a cui appartiene.

Classe con nome  
semplice

Vector

Classe con nome  
composto

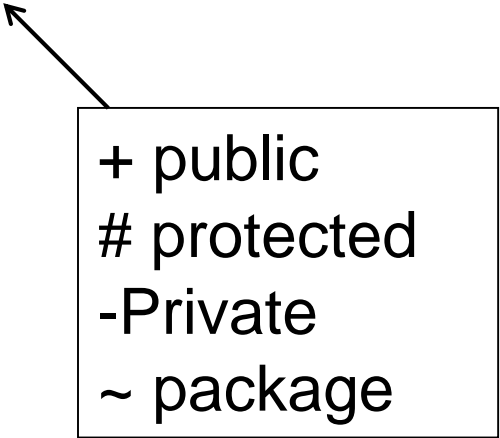
java::util::Vector

Notare il tipo di notazione per separare i  
package

# Definizione degli attributi

La sintassi di base per descrivere gli attributi all'interno del simbolo di classe è la seguente:

[ visibilità ] idattributo [molteplicità] [ :tipo ] [ =valore iniziale]

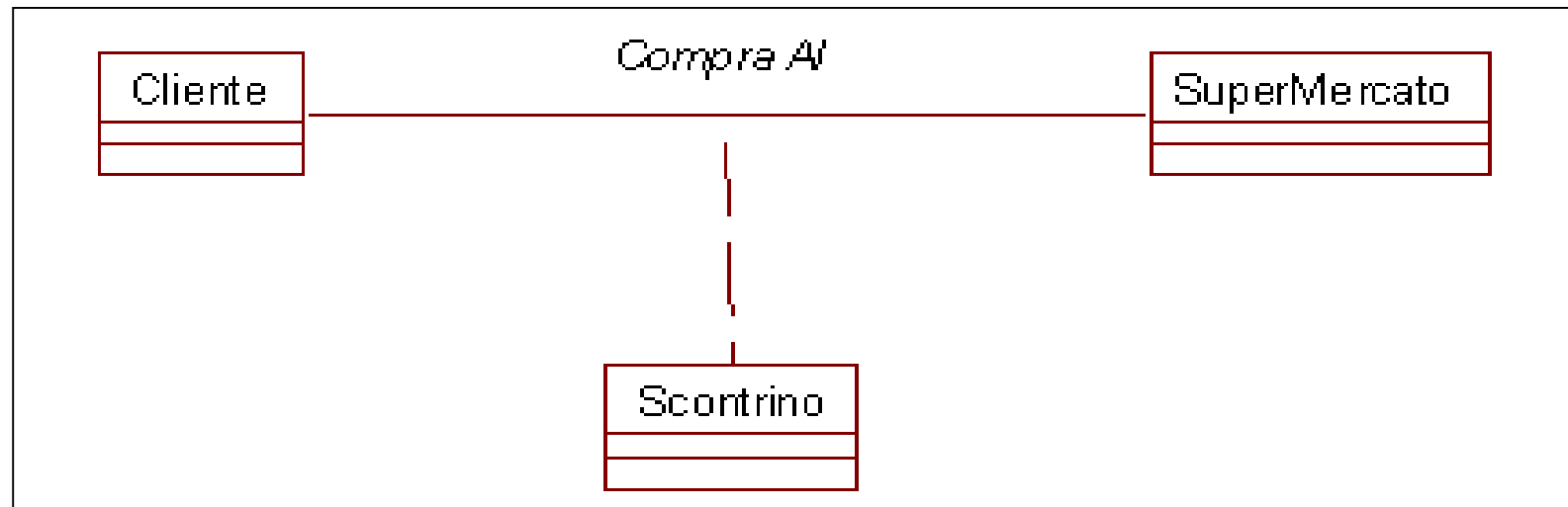
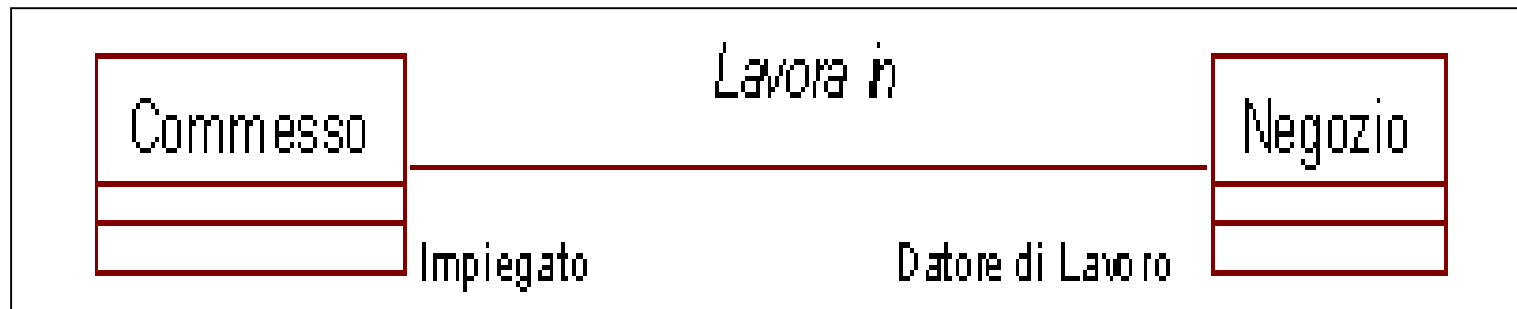


- + public
- # protected
- Private
- ~ package

## Definizione delle operazioni

ClassName
Attributo1 : tipo1 Attributo2 : tipo2 = "Valore di Default" ..... .....
operazione1() operazione2(Lista di parametri) operazione3() : Tipo restituito

# Associazioni

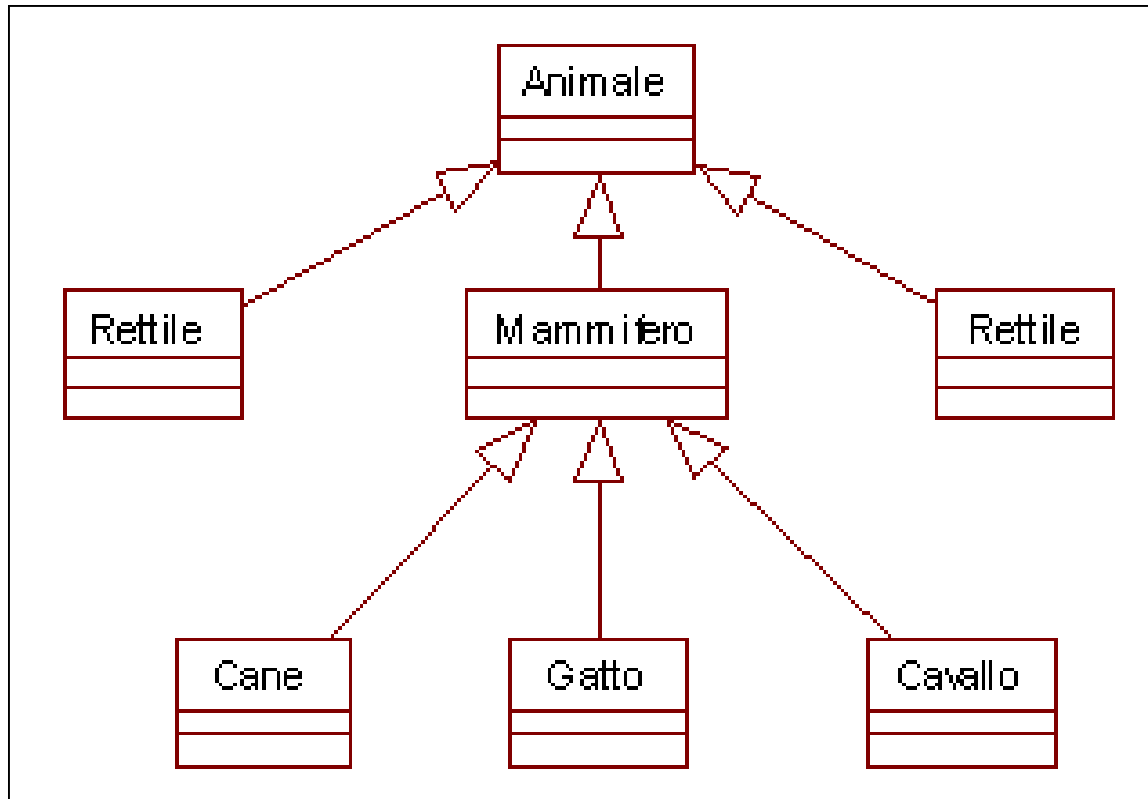


# Molteplicità

La molteplicità è un tipo speciale di associazione in cui si mostra il numero di oggetti appartenenti ad una classe che interagisce con il numero di oggetti della classe associata. Una classe, in generale, può essere correlata ad una altra nei seguenti modi:

- Uno ad uno
- Uno a molti
- Uno ad uno o più
- Uno a zero o uno
- Uno ad un intervallo limitato (es.: 1 a 2 - 20)
- Uno ad un numero esatto n
- Uno ad un insieme di scelte (es.: 1 a 5 o 8)

# Ereditarietà e Generalizzazione



# Aggregazione

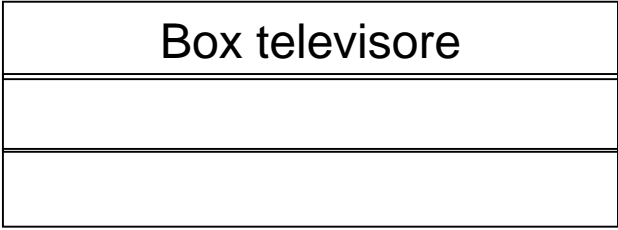
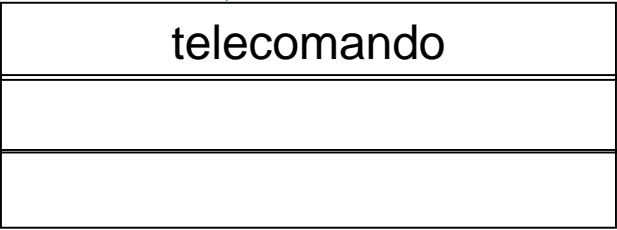
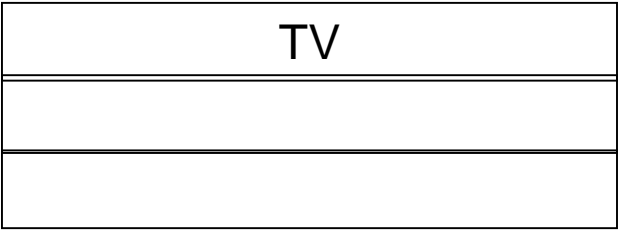
Qualche volta una classe può rappresentare il risultato di un insieme di altre classi che la compongono. Questo è un tipo speciale di relazione denominata **aggregazione**. Le classi che costituiscono i componenti e la classe finale sono in una relazione particolare del tipo: **parte - intero (part-whole)**

Un'aggregazione è rappresentata come una gerarchia in cui l' "intero" si trova in cima e i componenti ("parte") al di sotto. Una linea unisce "l'intero" ad un componente con un rombo raffigurato sulla linea stessa vicino all' "intero".

## Un esempio di aggregazione

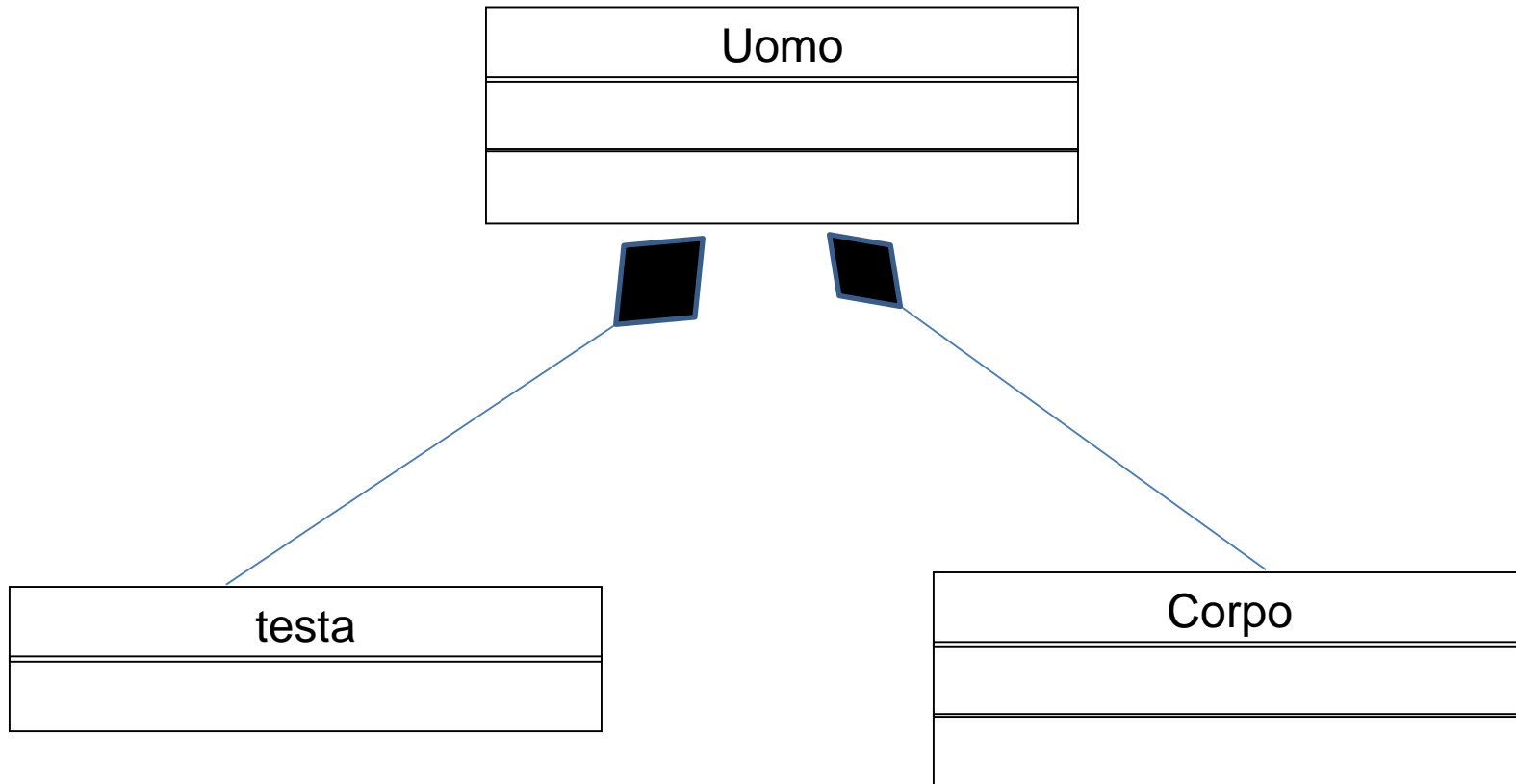
Esaminiamo le "parti" che costituiscono un Televisore. Ogni TV ha un involucro esterno (Box), uno schermo, degli altoparlanti, delle resistenze, dei transistor, un circuito integrato e un telecomando (oltre, naturalmente, ad altri tantissimi componenti!). Il telecomando può, a sua volta, contenere le seguenti parti: resistenze, transistor, batterie, tastiera e luci remote.







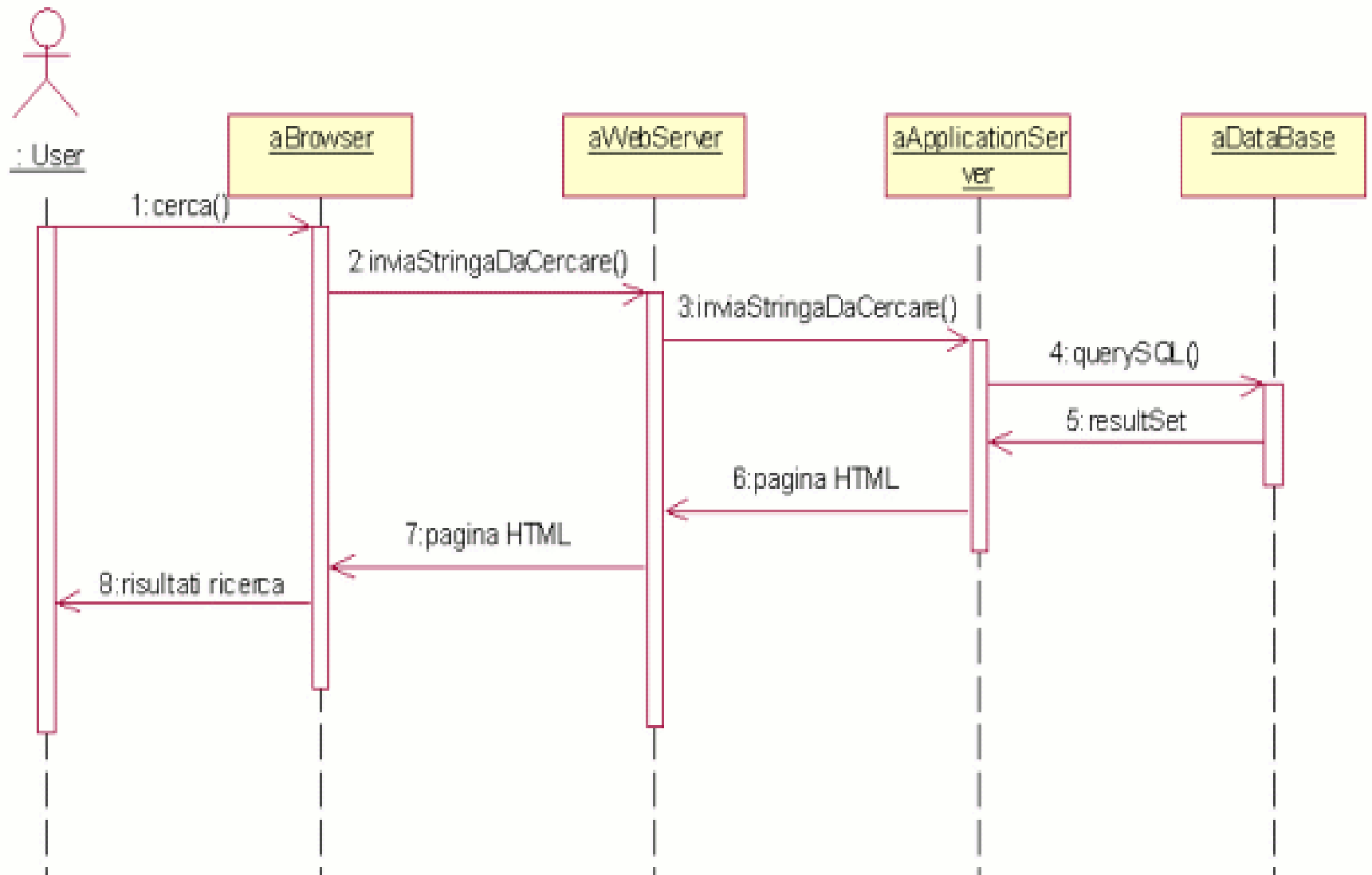
# Associazione composta

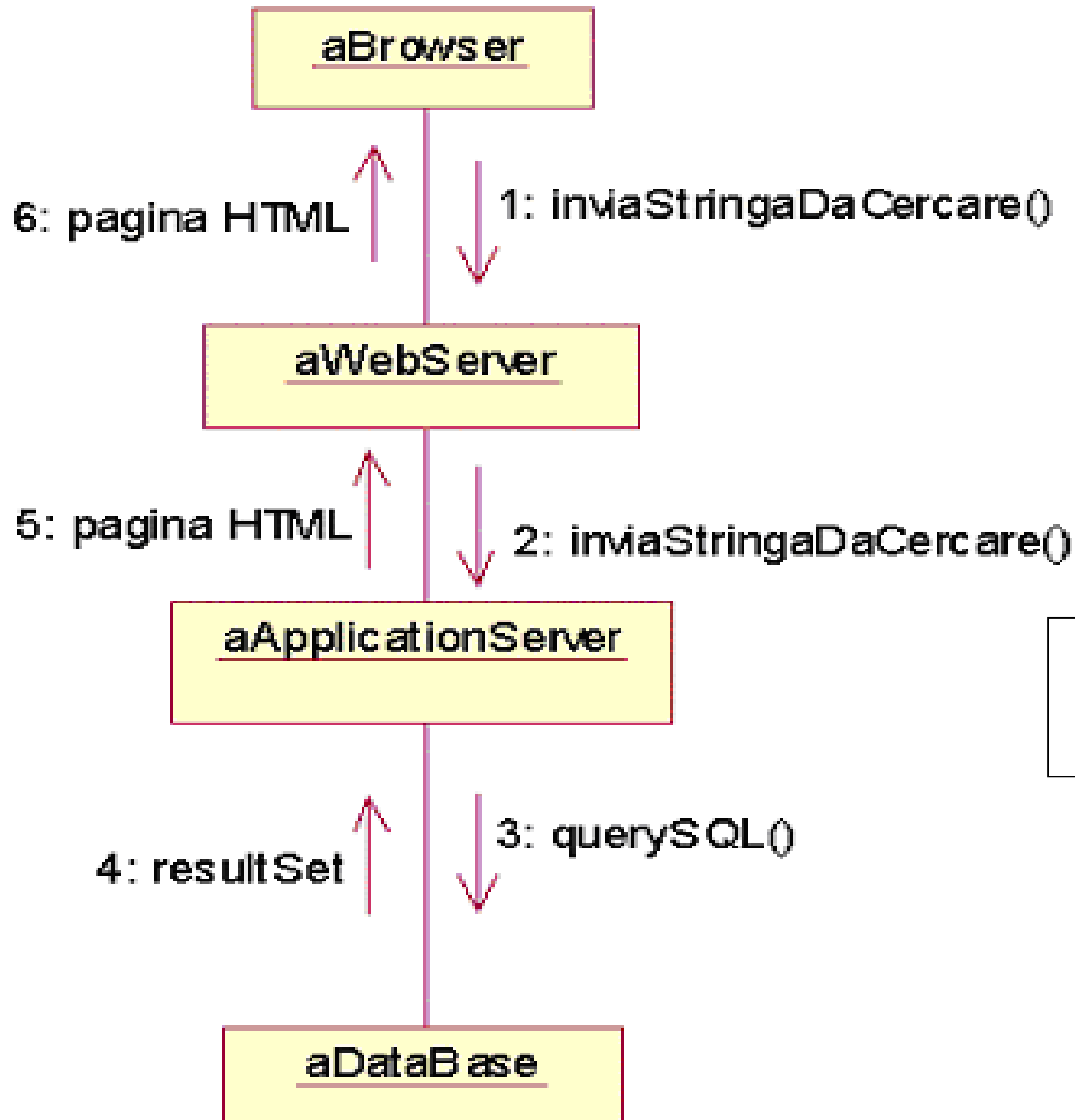


# Diagramma di sequenza e di collaborazione

Vediamo due diagrammi, molto simili tra loro, che evidenziano gli aspetti dinamici del sistema che si sta modellando. I due diagrammi servono ad **evidenziare le comunicazioni che avvengono tra gli oggetti coinvolti in uno scenario** (uno specifico percorso in un caso d'uso) specificando la sequenza dei messaggi che gli oggetti si scambiano; in particolare il diagramma di sequenza (**sequence**) evidenzia l'ordine temporale dei messaggi mentre il diagramma di collaborazione (**collaboration**) mette in risalto maggiormente il legame tra le entità (oggetti) che si scambiano i messaggi.

Gli elementi che compongono questi diagrammi sono chiaramente oggetti (istanze di classi) e spesso una sequenza di messaggi parte mediante l'attivazione da parte di un attore, visto, in questo contesto, anch'esso come un oggetto.





collaboration diagram  
di un ipotetico motore di  
ricerca

## **Diagramma di stato**

I diagrammi di stato (statechart) permettono di rappresentare il ciclo di vita di un oggetto. Sono particolarmente utili per descrivere il comportamento dinamico di quelle classi che hanno un ciclo di vita complesso rappresentabile, comunque, mediante un certo numero di stati finito e mediate le transizioni tra questi, al verificarsi di particolari eventi. Gli elementi del modello sono gli stati e le transizioni. In particolare nel diagramma si distinguono lo stato iniziale e quello finale per la loro diversa rappresentazione.

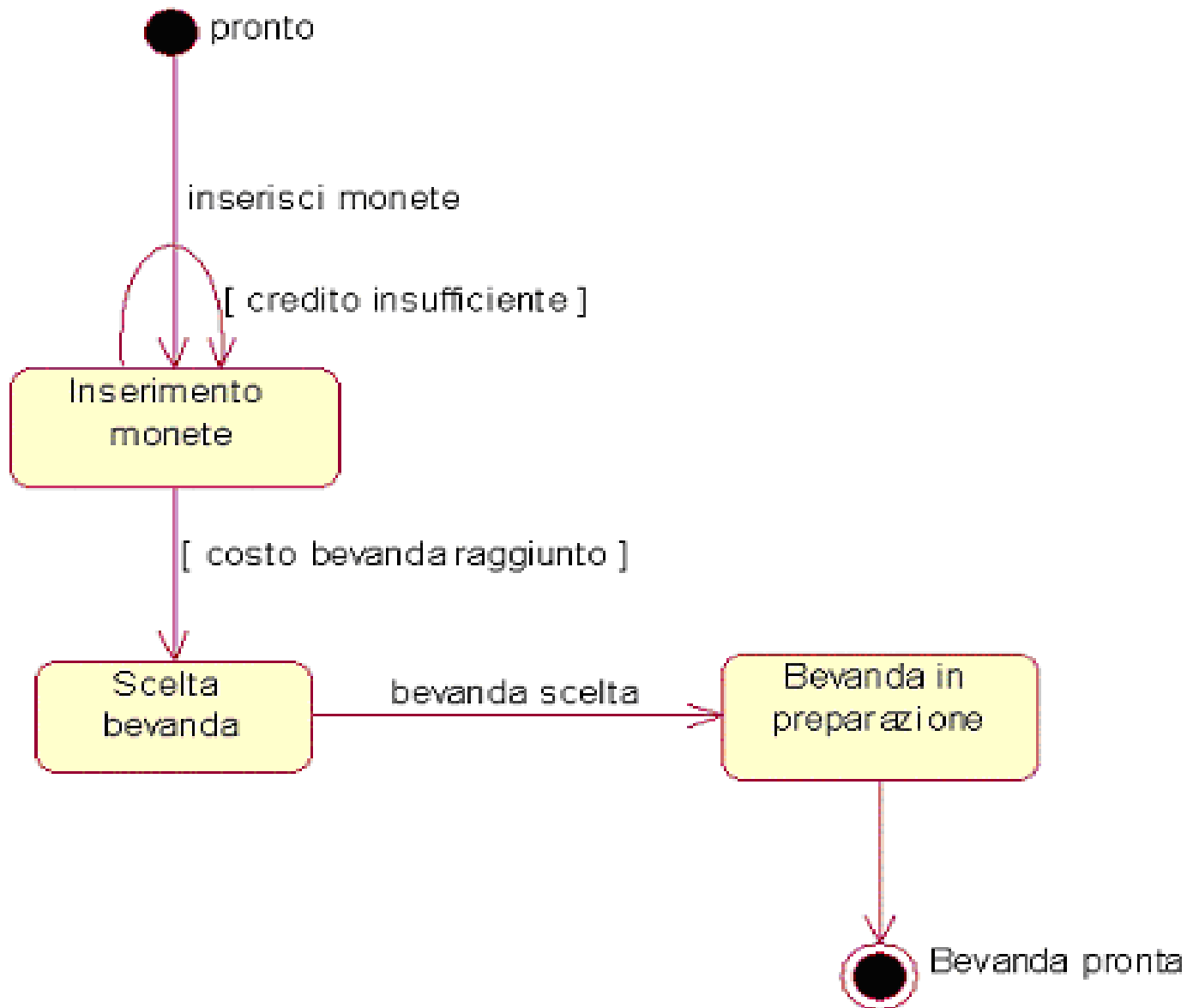


Stato iniziale

Stato



Stato finale



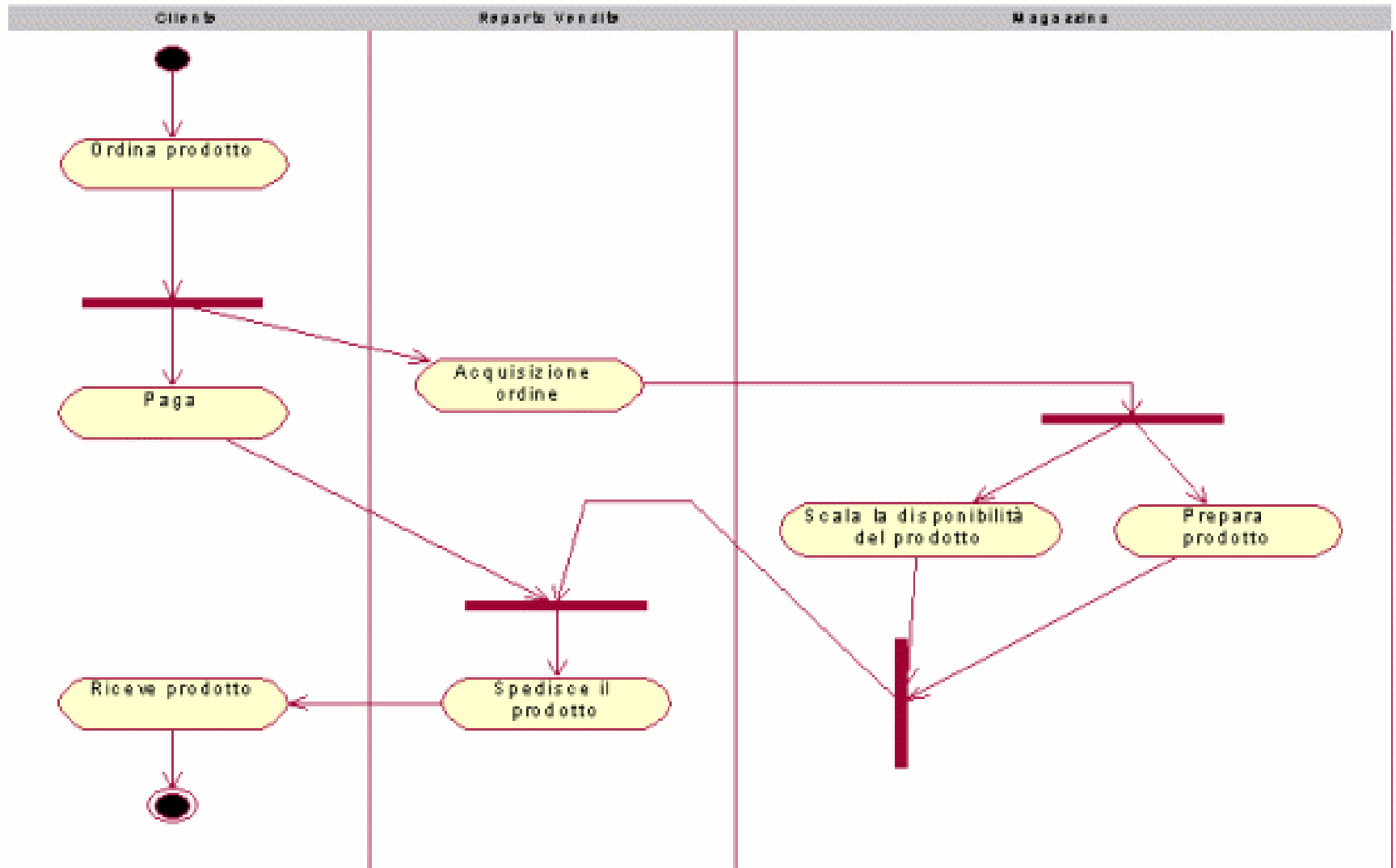
## **Diagramma di attività**

**I diagrammi di attività (activity) servono a rappresentare sistemi di workflow a qualsiasi livello, a partire dalla descrizione di un flusso di una logica di business in cui ad esempio sono coinvolti i vari settori di un'azienda, sino alla rappresentazione di un flusso di uno use case. I diagrammi di attività permettono anche di rappresentare processi paralleli e la loro sincronizzazione.**

Attività

Nodo  
decisionale

barre di  
sincronizzazione





## Diagramma di attività

**Esempio di activity diagram che descrive il flusso relativo ad un'operazione di ordine di un prodotto da parte di un cliente ad un ipotetico reparto vendite di un'azienda. Nota: Le frecce che "escono" da una barra di sincronizzazione rappresentano attività parallele**

## **Diagramma dei componenti**

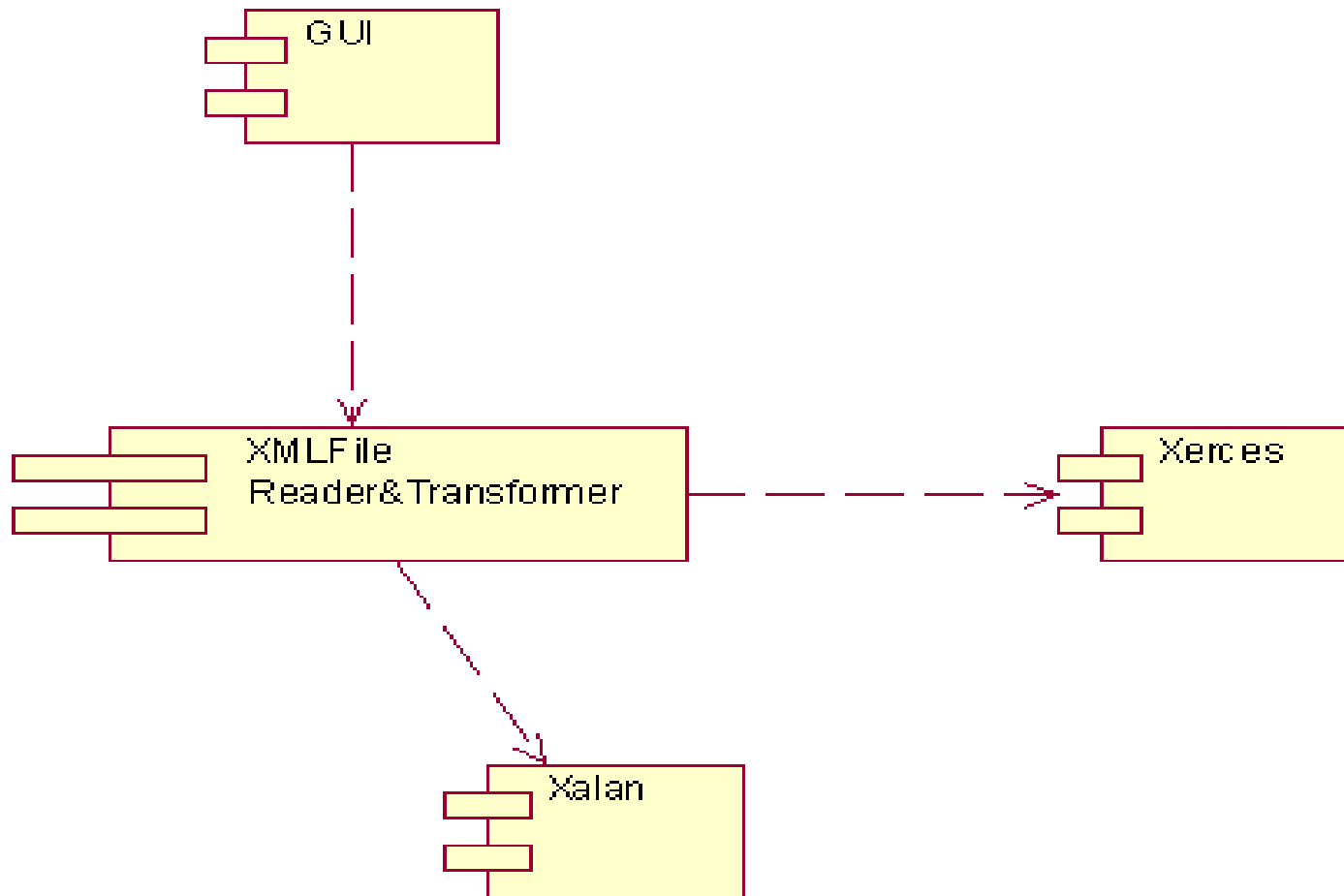
Il diagramma dei componenti (component) descrive la struttura di un sistema in termini dei moduli software (componenti) e di dipendenze tra loro.

Ad esempio, in java, un modulo software è un package di una particolare API.

**Questo diagramma contribuisce, insieme al diagramma delle classi, alla vista statica sul sistema.**

Gli elementi del modello sono rappresentati da componenti e dipendenze.

**Esempio di component diagram di un lettore di file XML che utilizza i noti parser Xerces e Xalan per leggere e visualizzare sulla GUI il contenuto di un file xml**



## Diagramma di distribuzione

Un diagramma di distribuzione (deployment) serve a rappresentare l'architettura fisica del sistema. Esso è formato da nodi che rappresentano elementi hardware (macchine server, PC, dispositivi, reti,...) all'interno dei nodi è possibile rappresentare i componenti che "girano" su questo

