Laboratorio di Sistemi Interattivi

Lezione 4: Modellazione orientata agli oggetti

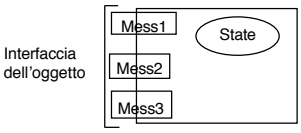
Sommario

- Modellazione orientata agli oggetti
- Viste sul modello
- Lo strumento UML

Modellazione orientata agli oggetti

- Oggetti
- Classi
- Operazioni
- Metodi
- Relazioni

Oggetti

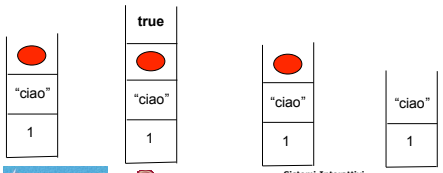


Possiedono identità
Incapsulamento dello stato, inaccessibile ad altri oggetti
Comunicazione fra oggetti attraverso messaggi
Messaggi esprimono query, richieste di modifiche di stato, richieste di svolgimento di computazioni

Esempio: lo stack

push(new Boolean(true))pop()

pop()



Classi

Definizione interfaccia	Definizione struttura stato	Codice dei metodi

```
class Stack {
  private List elements;
  public Stack() {
     elements = new ArrayList();
  }
  …
}
```

Oggetti istanze di classi
Classe definisce caratteristiche comuni a ogni sua istanza.
Ogni istanza di una classe è in grado di rispondere agli stessi messaggi, utilizzando gli stessi metodi

Operazioni

- Nome dell'operazione (nome del messaggio)
- Tipi degli argomenti
- Tipo del risultato

Es.

```
void push(Object o)
Object pop()
boolean isEmpty()
```


Metodi

- Codifica concreta, in un linguaggio di programmazione, del comportamento da mettere in atto per realizzare un'operazione (in risposta a un messaggio).
- Codice del metodo presente una sola volta nella classe, non replicato in ogni istanza.

```
Object pop() {
    Object ob = null;
    if (!isEmpty()) {
      ob = elements.at(0);
      elements.remove(0);
    }
    return ob;
}
```


Relazioni

- Ereditarietà
  - Struttura
  - Comportamento
- Associazione
  - Possibilità di collaborare
- Composizione / aggregazione
  - Relazione parte-tutto
- Dipendenza
  - Necessità dell'esistenza

Lo strumento UML

- Permette di costruire e utilizzare un modello come composizione di viste diverse.
- Linguaggi diagrammatici per ogni aspetto del modello
- Coerenza complessiva del modello
- Modello esprime vincoli sulle possibili realizzazioni

Viste sul modello

- Vista di utente
- Vista architetturale
  - Vista strutturale
  - Vista dinamica
    - Attività collaborative
    - Attività singole

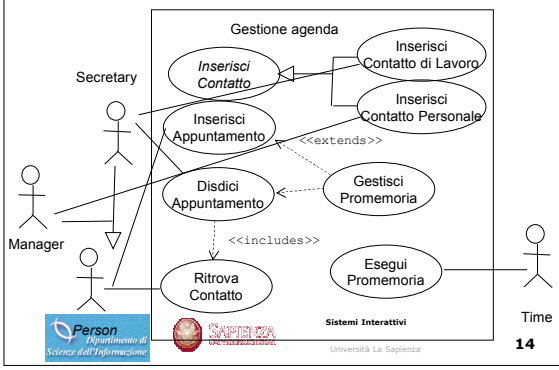
Vista di utente

- Funzionalità del sistema dal punto di vista degli attori
- Questioni
  - Chi sono gli attori?
    - Chi ricava valore dall'utilizzo del sistema?
    - Chi fornisce servizi al sistema?
  - Quali sono le funzionalità?
  - Cosa deve avvenire per ottenere la funzionalità?

Casi d'uso (vista di utente)

- Descrizione funzionalità del sistema che producono valore per gli utenti del sistema.
- Sequenze di azioni che il sistema può eseguire interagendo con gli utenti.
- Descrizione dei ruoli di sistema e utenti.
- Richiede pre- e asserisce post-condizioni
- Casi d'uso atomici. Non interagiscono.

Esempio



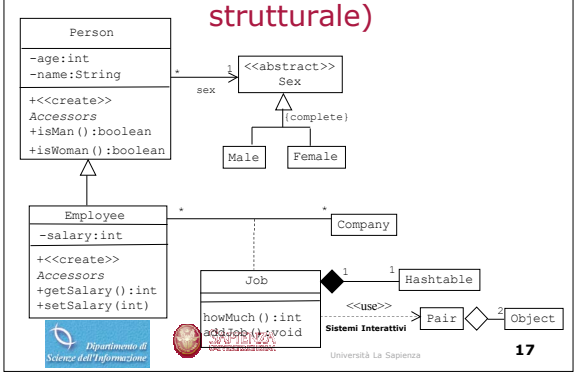
Descrizione di un caso d'uso

Use case:	InserisciAppuntamento
Actors:	Person
Preconditions: 1. Il sistema è attivo. 2. Memoria appuntamenti non è satura	
Flow of Events: 1. Caso d'uso inizia quando utente seleziona inserimento appuntamento 2. Sistema inizia dialogo per chiedere ora, luogo, e contatti appuntamento 3. Utente inserisce dati richiesti 4. Utente produce segnale di conferma 5. Sistema acquisisce i dati	
Extension points: 1. Attore è Manager <<GestisciPromemoria>>	
Postconditions: 1. Sistema ha memorizzato un nuovo appuntamento 2. Memoria appuntamenti non è vuota	

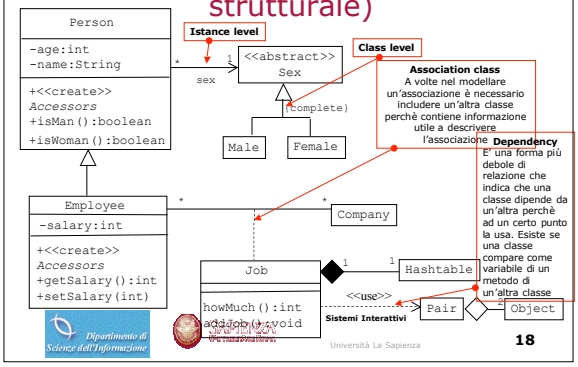
Vista strutturale

- Livello delle classi
  - Quali sono i tipi di componente necessari a realizzare le funzionalità?
  - Quali sono relativi al problema?
    - Quali sono comuni a tutti i problemi in uno stesso dominio?
    - Quali sono specifici dell'applicazione?
  - Quali sono relativi all'interazione?
  - Quali sono di utilità indipendenti dal dominio?
  - Quali sono le relazioni fra le classi?
- Livello delle istanze
  - Come è fatta la configurazione iniziale del sistema?
  - Come è fatta una configurazione in un particolare stato dell'interazione?

Diagrammi delle classi (vista strutturale)



Diagrammi delle classi (vista strutturale)



Ereditarietà e associazioni

```
class Person {
 String name; int age; Sex sex;
 Person(String nm, int years, Sex
  sx) {
   name=nm; age=years; sex=sx;
 }
 boolean isMan() {
   return sex instanceof Male;
 }
boolean isWoman() {
   return sex instanceof Female;
 }
}
```

```
class Employee extends Person {
 int salary;
 Employee(String name, Sex sex,
  int age, int sal) {
   super(name, age, sex);
   salary = sal;
 }
 int getSalary() {return salary;}
 void setSalary(int s) {salary=s;}
}
```

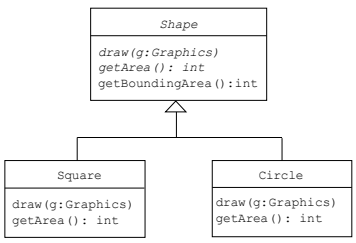
```
abstract class Sex {}
class Male extends Sex{}
class Female extends Sex{}
```


Associazione, composizione e dipendenza

```
class Pair {
  Object one, two;
  Pair(Object a, Object b) { one = a; two = b; }
}
```

```
class Job {
   Map salaries = new Hashtable();
   addJob(Company cmp, Employee pers, int amt) {
      Pair pair = new Pair(cmp, pers);
      salaries.put(pair,new Integer(sal));
      …
   }
   int howMuch(Company cmp, Employee pers) {
      …
   }
}
```

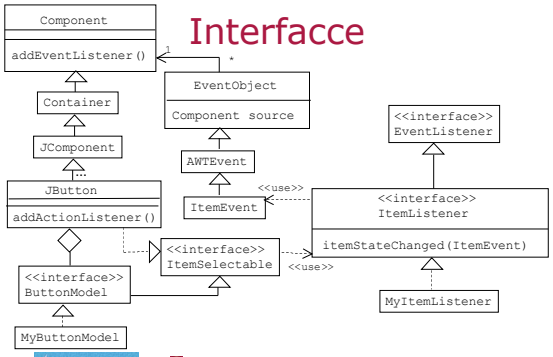

Metodi astratti



Interfacce

- Una interfaccia è una variante di una classe.
- Una classe fornisce una **implementazione** "incapsulata" di una certa funzionalità
- Una **interfaccia** fornisce solo la **definizione** della funzionalità mentre una classe **separata** implementa la funzionalità definita
- Classe **astratta** con metodi **astratti**, e classi **figlie** che implementano le funzionalità?
- Gli elementi vengono **legati** in una gerarchia, ma magari rappresentano entità **diverse**
- Classi **differenti** che appartengono a **gerarchie differenti** possono mantenere le relazioni di **ereditarietà originarie** e tuttavia **realizzare** le funzionalità definite dai metodi dell'interfaccia

Interfacce



Vista dinamica (attività collaborative)

- Quali elementi devono collaborare per realizzare una certa funzionalità?
- Che ruolo svolgono nella collaborazione?
- Quali sequenze di interazioni fra gli elementi sono necessarie per realizzare una certa funzionalità?

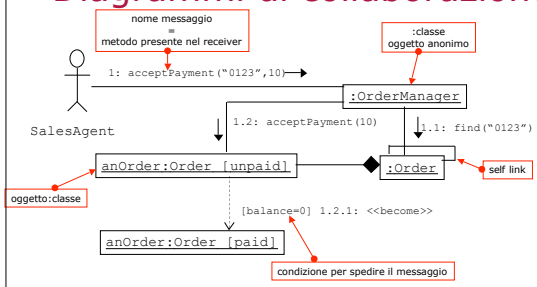
Diagrammi d'interazione (vista dinamica di collaborazione)

- Modellano collaborazioni e interazioni fra oggetti che realizzano casi d'uso
- Unico modello UML con due viste:
  - Collaboration (Communication in UML 2.0) diagrams: enfatizzano le relazioni strutturali fra gli oggetti
  - Interaction diagrams: enfatizzano la sequenza temporale ordinata degli scambi dei messaggi

Diagrammi di collaborazione

- Enfatizzano le relazioni strutturali fra gli oggetti
  - Per scambiarsi messaggi, gli oggetti (le classi di appartenenza) devono essere in una qualche relazione

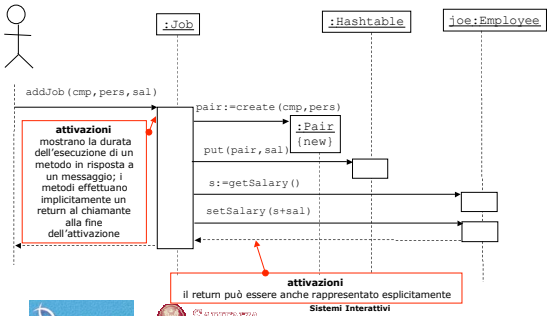
Diagrammi di collaborazione



Diagrammi di sequenza

- Notazione alternativa per stesso contenuto semantico
- Descrivono l'ordine in cui i messaggi vengono mandati in uno scenario di un caso d'uso
- Rappresentano uno specifico insieme di messaggi e interazioni tra oggetti

Diagrammi di sequenza



Realizzazione del caso d'uso

```
class Pair {
  Object one, two;
  Pair(Object a, Object b) { one = a; two = b; }
}
```

```
class Job {
   Map salaries = new Hashtable();
   addJob(Company cmp, Employee pers, int sal) {
     Pair pair = new Pair(cmp, pers);
     salaries.put(pair,new Integer(sal));
     pers.setSalary(sal+pers.getSalary());
   }
   int howMuch(Company cmp, Employee pers)
      return ((Integer)(salaries.get(new Pair(cmp,pers))).intValue();
   }
}
```

Vista dinamica (attività singole)

- Quali sono i possibili stati di un'entità (oggetto)?
  - A quali configurazioni di valori degli attributi corrispondono?
- Quali eventi determinano le transizioni di stato?
- Possiamo decomporre uno stato in sottostati? (e.g. stati veglia/sonno, sonno ha fasi REM e fasi non REM)
- Cosa può fare il sistema permanendo nello stesso stato?

Macchine a stati (vista dinamica singola)

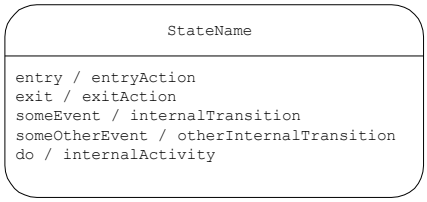
- Hanno unico stato iniziale (pseudostato) e unico stato finale.

- Stato iniziale è pseudostato, ha solo una transizione uscente verso il "vero" stato iniziale.
- Stato finale indica il completamento dell'attività di una macchina. Se è stato finale di una macchina associata a una classe, indica terminazione dell'istanza.

Azioni e attività

- Azioni: istantanee e ininterrompibili
  - entry: eseguita ad ogni ingresso nello stato, su qualsiasi transizione
  - exit: eseguita ad ogni uscita dallo stato, su qualsiasi transizione
  - internal: azione associata a qualche evento, ma che non fa lasciare lo stato (entry e exit non eseguite)
- Attività: hanno durata e possono essere interrotte
  - do: eseguita fino a che si permane nello stato, può terminare prima di lasciare lo stato, indipendentemente. Produce evento di completamento

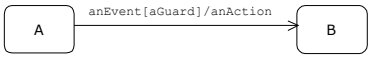
Sintassi degli stati



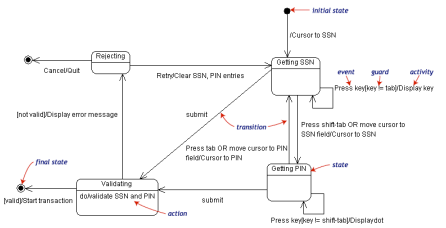
Transizioni

- Event: occorrenza interna o esterna che fa scattare la transizione
- Guard: condizione booleana che deve verificarsi prima che la transizione possa avvenire
- Action: occorre quando la transizione scatta

Sintassi grafica



Esempio

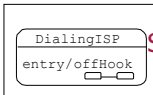


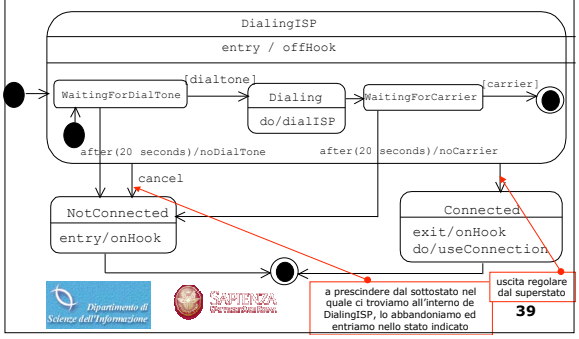
SSN = social security number
PIN = personal id number

Stati composti

- Contengono una o più macchine a stati annidate.
- Ogni sottostato eredita tutte le transizioni dello stato genitore
- Annidamento è transitivo (in genere non più di due-tre annidamenti)

Sintassi grafica (stati composti sequenziali)

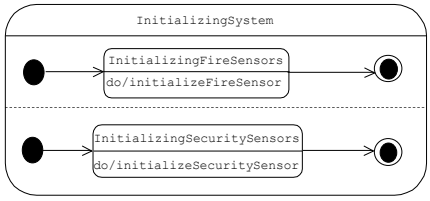




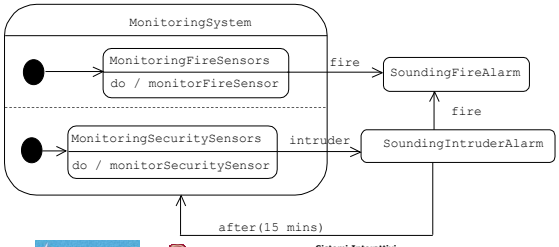
a prescindere dal sottostato nel quale ci troviamo all'interno de DialingISP, lo abbandoniamo ed entriamo nello stato indicato

uscita regolare dal superstato

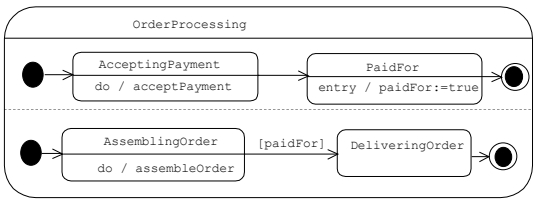
Stati composti concorrenti



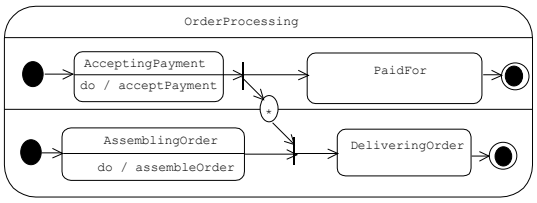
Terminazioni indipendenti



Comunicazione fra sottomacchine (con attributi)



Comunicazione (con stati di sincronizzazione)



Stati History

- Permettono di ricordare lo stato in cui si è lasciata una macchina e rientrare nella macchina in quello stato anziché nello stato iniziale
- Shallow history (H)
  - Ci si ricorda solo dello stato al livello indicato
- Deep history (H*)
  - Ci si ricorda degli stati delle sottomacchine

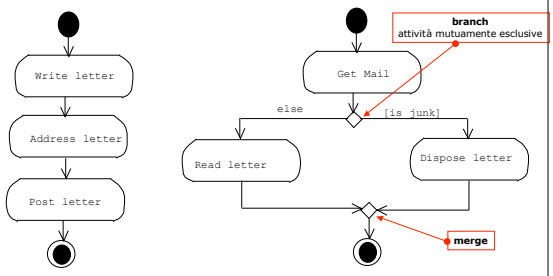
Diagrammi di attività

- In una macchina a stati c'è un oggetto che attraversa un processo modificando il proprio stato (o un processo visto come un oggetto)
- I diagrammi di attività invece modellano un processo come una collezione di attività e transizioni fra attività
- Sono espressi tramite composizioni di azioni

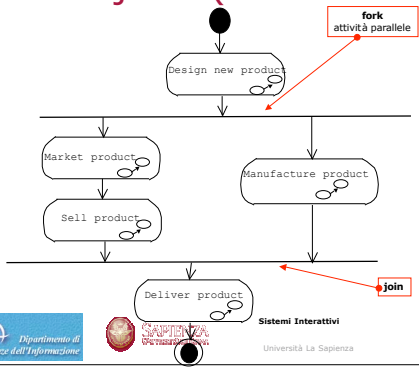
Azioni

- Atomiche
- Non interrompibili
- Istantanee

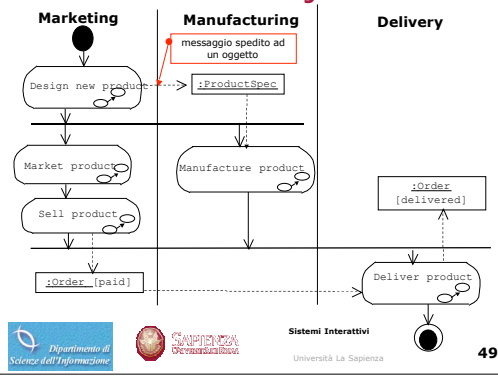
Transizioni, decisioni



Forks e joins (e sub-attività)



Swimlanes e object flows



Altro esempio