

# **Analisi dei Requisiti**

# Analisi: Obiettivi

- Fornire un modello del sistema che sia
  - Corretto
  - Completo
  - Consistente
  - Non ambiguo
- Gli sviluppatori
  - formalizzano la specifica delle richieste prodotta durante la fase di “raccolta dei requisiti”
  - validano, correggono e chiariscono eventuali errori presenti nella specifica delle richieste
  - esaminano più in dettaglio le condizioni limite e i casi eccezionali
- Il cliente e l'utente sono coinvolti
  - se devono essere cambiate delle richieste o
  - se c'è bisogno di ulteriori informazioni

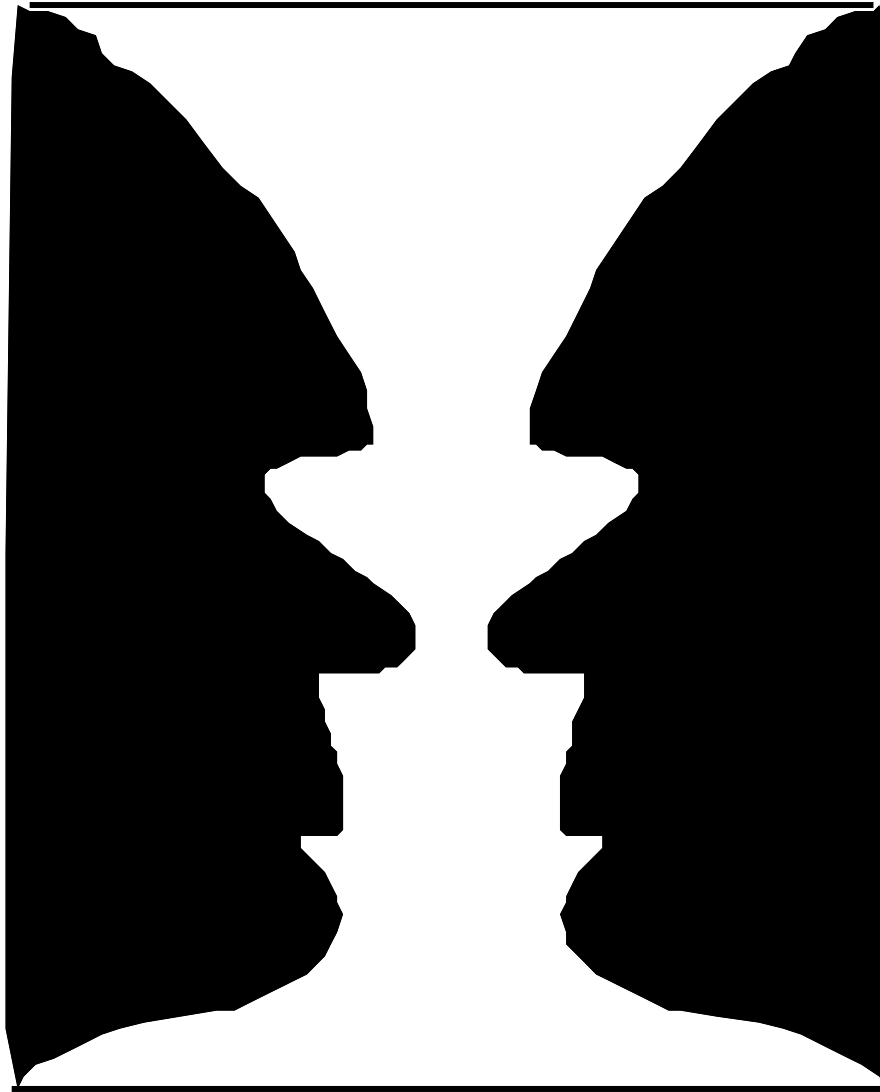
# Analisi OO: Obiettivi

- Viene costruito un modello che descrive il dominio di applicazione
  - Es. Il modello di analisi relativo all' orologio descrive come l' orologio rappresenta il tempo:
    - I giorni della settimana
    - Le fasi lunari
    - Gli anni bisestili
- Il modello di analisi è esteso per descrivere come gli attori e il sistema interagiscono per manipolare il modello del dominio di applicazione:
  - Come il proprietario resetta l' ora?
  - Come il proprietario resetta il giorno della settimana?
- Il modello di analisi e le richieste non funzionali saranno utilizzati per la definizione dell' architettura del sistema che avverrà nella fase di design ad alto livello

# Outline della lezione

- Vengono descritte in dettaglio le attività di analisi:
  - Identificazione degli oggetti
  - Definizione del comportamento degli oggetti
  - Definizione delle relazioni tra gli oggetti
  - Classificazione degli oggetti
  - Organizzazione degli oggetti
- Vengono descritti alcune questioni di gestione dell'attività di analisi nel contesto di un progetto di sviluppo multi-team

# Ambiguità: Cosa vedete?



# Ambiguità

- Se l'immagine “multi-stable” fosse stata una specifica delle richieste, cosa avreste costruito, un vaso o due facce che si guardano?
- Le specifiche contengono ambiguità determinate dalla
  - inaccuratezza inerente nel linguaggio naturale
  - assunzioni fatte dagli autori della specifica e non esplicitate
    - Una quantità specificata senza l'unità di misura
    - Un'ora senza la zona a cui si riferisce
- Il **processo di formalizzazione** aiuta ad individuare
  - aree di ambiguità
  - inconsistenze
  - omissioni
- Identificato un problema nella specifica, occorre risolverlo acquisendo maggiore informazioni dal cliente e dall'utente

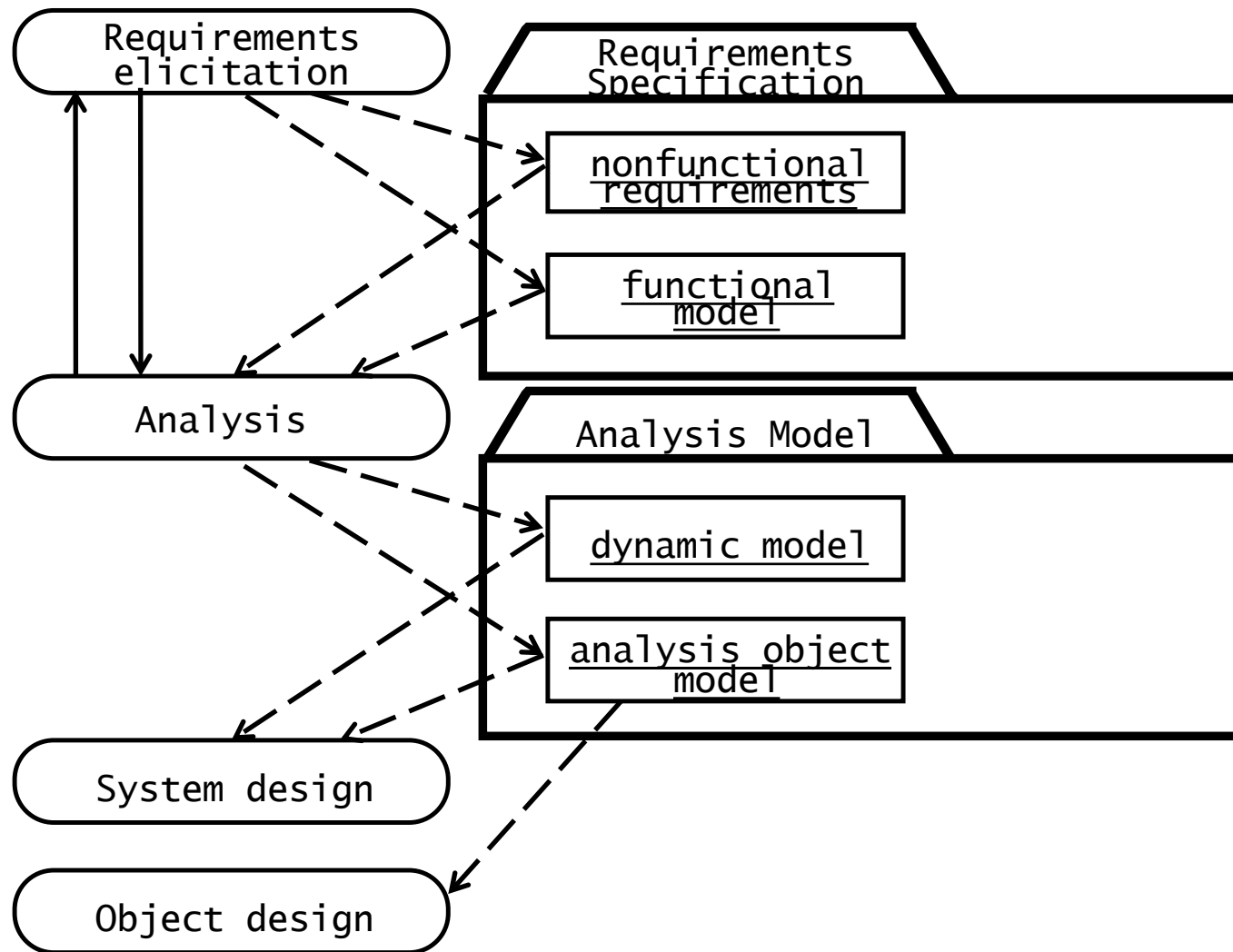


Analisi dei requisiti e identificazione dei requisiti sono attività iterative e incrementali che avvengono in concorrenza

# Una Overview dell' Analisi

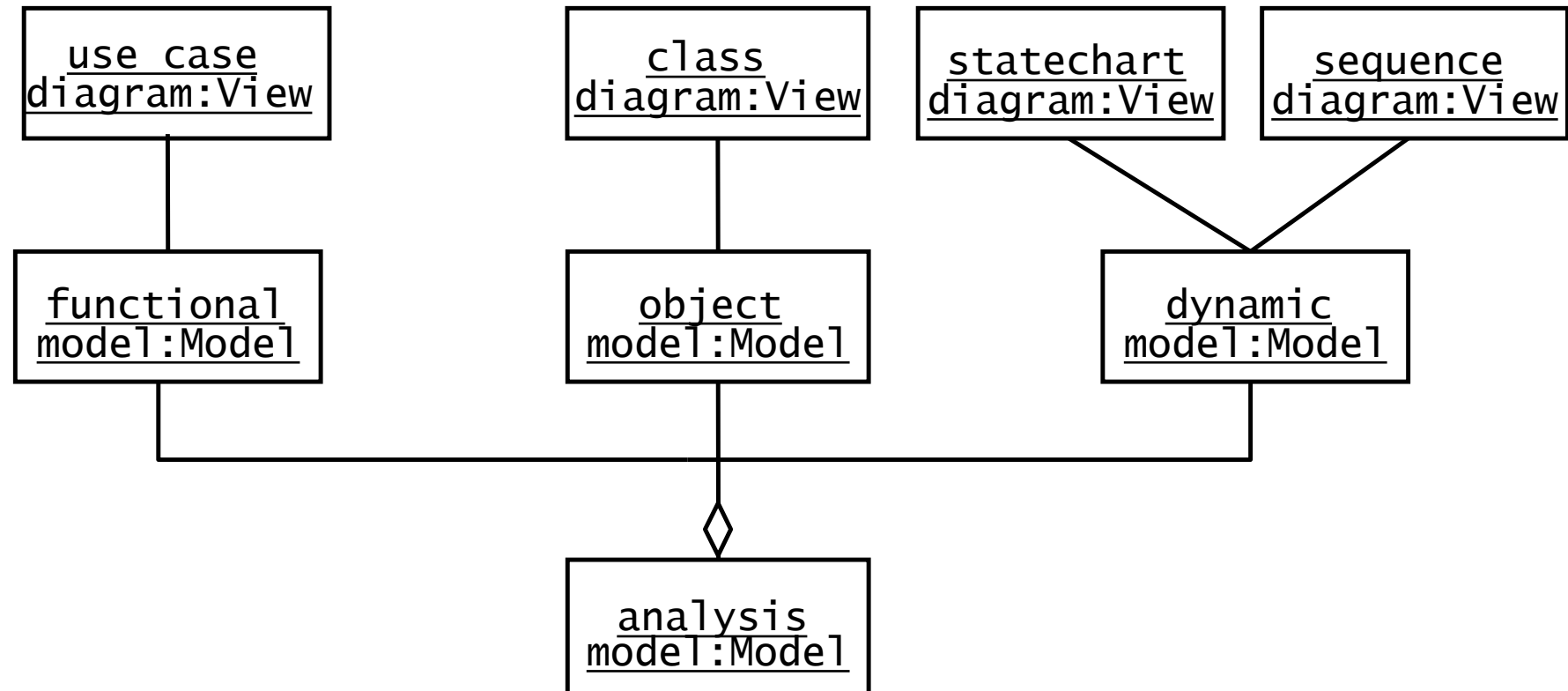
- L' analisi dei requisiti ha come obiettivo quello di tradurre le specifiche dei requisiti in un modello del sistema formale o semiformale:
  - Formalizzando e strutturando i requisiti si acquisisce maggiore conoscenza ed è possibile scoprire errori nelle richieste
  - La formalizzazione costringe a risolvere subito questioni difficili che altrimenti sarebbero rimandate
    - Poiché il modello di analisi può non essere comprensibile al cliente e all' utente è necessario modificare di conseguenza anche il documento di specifica delle richieste
- Il modello di analisi è composto di tre modelli:
  - **Modello funzionale**, rappresentato da use case e scenari
  - **Modello degli oggetti di analisi (analysis object model)**, rappresentato dal diagramma delle classi e dal diagramma degli oggetti
  - **Modello dinamico**, rappresentato da statechart e sequence diagram
- **Gli use case e gli scenari prodotti nella fase di raccolta dei requisiti sono raffinati per derivare il modello ad oggetti e il modello dinamico**

# Prodotti della raccolta dei requisiti e dell'analisi dei requisiti.





**The analysis model is composed of the functional model, the object model, and the dynamic model.**

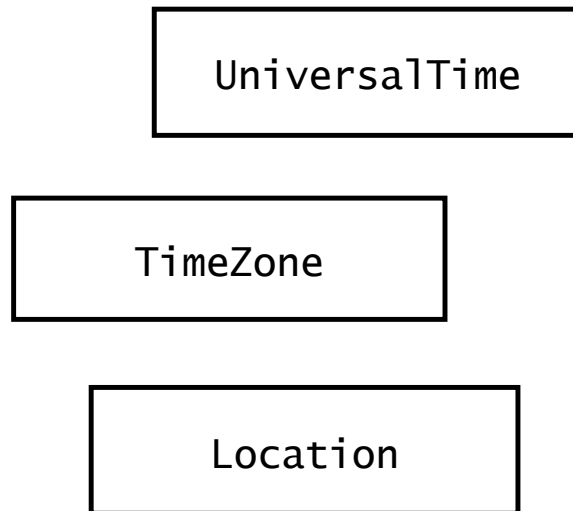


# Concetti di Analisi

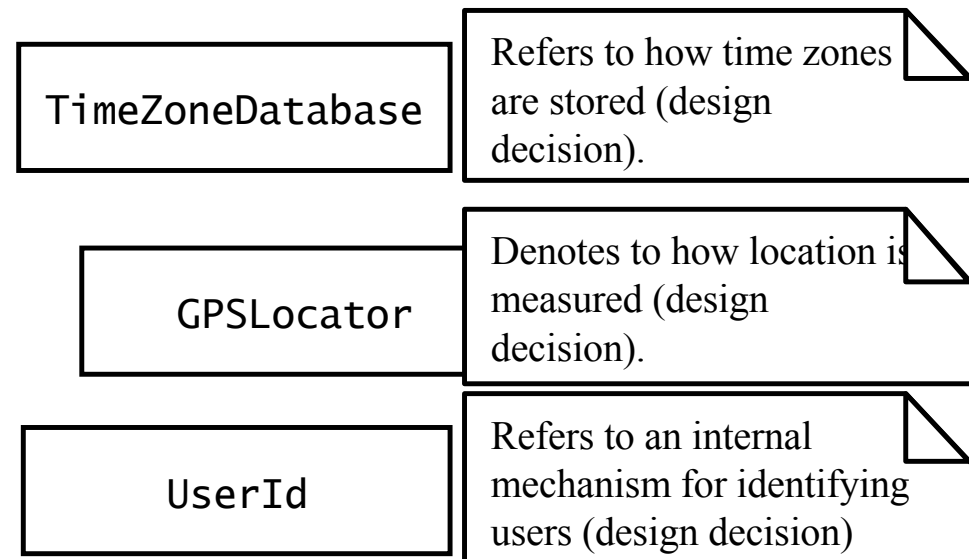
- **Il modello degli oggetti di analisi**
  - rappresenta il sistema dal punto di vista dell'utente
  - si focalizza sui concetti che sono manipolati dal sistema, le loro proprietà e relazioni
  - è un dizionario visuale dei concetti principali visibili all'utente
  - l'UML class diagram include classi, attributi e operazioni
- **Il modello dinamico**
  - si focalizza sul comportamento del sistema
    - I sequence diagram rappresentano le interazioni tra un insieme di oggetti durante un singolo use case
    - Gli statechart rappresentano il comportamento di un singolo oggetto o di alcuni oggetti strettamente accoppiati
  - consente di assegnare le responsabilità alle classi e quindi individuare nuove classi che sono aggiunte al modello degli oggetti dell'analisi
- **Ricorda:** sia il modello dinamico che il modello degli oggetti rappresentano concetti a livello utente non a livello di componenti e classi software
  - Le classi di analisi rappresentano astrazioni che saranno realizzati con molti più dettagli successivamente

# Esempi di oggetti durante l'analisi di SatWatch.

Concetti del dominio che dovrebbero essere rappresentati nel modello di analisi.



Classi ottenute a seguito di scelte software che non dovrebbero essere rappresentate nel modello di analisi.



# *Object Modeling*

- Steps during object modeling
  1. Class identification
  2. Find the attributes
  3. Find the methods
  4. Find the associations between classes
- Order of steps
  - Order of steps secondary, only a heuristic
  - Iteration is important
  - Static model will be refined when devising dynamic models

# Concetti di Analisi:

## Oggetti Entity, Boundary, e Control

- In ogni sistema troviamo oggetti di tipi ricorrente. Conviene classificarli
- Il modello degli oggetti di analisi consiste di **oggetti Entity, Boundary e Control**
- Gli oggetti *Entity*
  - rappresentano l'informazione persistente (oggetti del dominio di applicazione, "oggetti business")
- Gli oggetti *Boundary*
  - rappresentano le interazioni tra gli attori e il sistema (oggetti relativi all'interfaccia utente, oggetti dell'interfaccia dei dispositivi, oggetti di interfaccia del sistema)
- Gli oggetti *Control*
  - si occupano di realizzare gli use case, rappresentano il controllo dei task eseguiti dal sistema, contengono la logica e determinano l'ordine dell'interazione degli oggetti)
- Es. 2Bwatch:
  - Year, Month e Day sono oggetti **Entity**
  - Button e LCDDisplay sono oggetti **Boundary**
  - ChangeDateControl è un oggetto **Control**, che rappresenta l'attività di cambiare la data premendo combinazioni di bottoni
- Si ha una semplice euristica per distinguere concetti diversi ma correlati:
  - Es. L'ora e il display che mostra l'ora hanno proprietà diverse. La differenziazione tra oggetti Boundary e Entity forza tale distinzione
- L'approccio **three-object-type** porta a modelli che sono più flessibili e facili da modificare:
  - L'interfaccia al sistema (rappresentata da oggetti boundary) è più soggetta a cambiamenti rispetto alle funzionalità (rappresentate da oggetti entity e control)

# ***Use of Object Types***

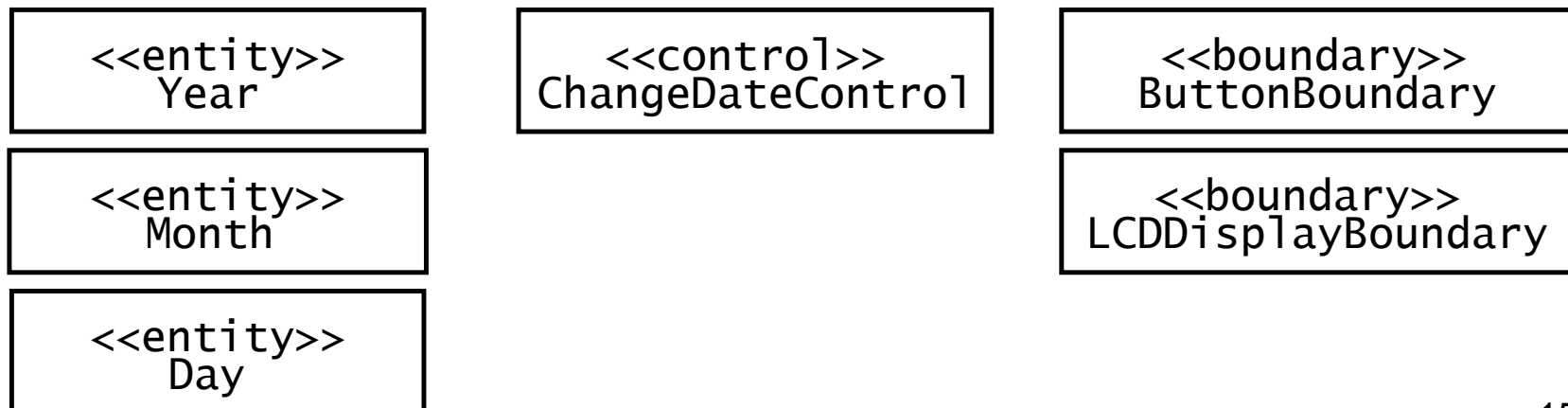
- Helps identify objects and clearly identify responsibilities
- Helps verify objects:
  - Most objects should fit in one (and only one) of these categories
- Helps read class diagram (using stereotypes)

# Concetti di Analisi: Oggetti Entity, Boundary, e Control.

- UML fornisce il meccanismo degli stereotipi per consentire di aggiungere tale meta-informazione agli elementi di modellazione
- E' opportuno usare convenzioni sui nomi:
  - Gli oggetti *control* possono avere il suffisso Control
  - Gli oggetti *boundary* dovrebbero avere nomi che ricordano aspetti dell'interfaccia (es. suffisso Form, Button, ecc)

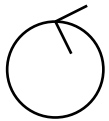
## Classi di analisi per l'esempio 2Bwatch

---

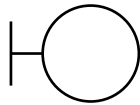


# Notazioni alternative per gli stereotipi

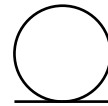
- Le tre icone di Jacobson per gli stereotipi



<<control>>



<<boundary>>

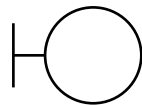
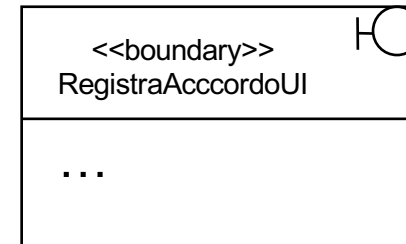
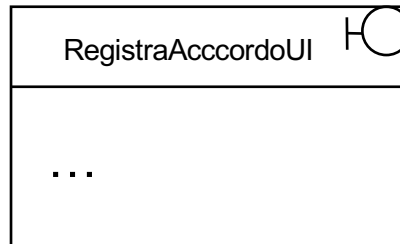
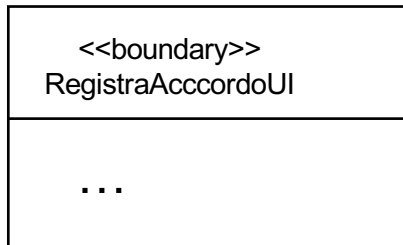


<<entity>>

- Sono possibili tre diverse notazioni
  - Utilizzo dei simboli di stereotipi nei simboli di classe
  - Collasso del riquadro della classe nel solo simbolo di stereotipo
  - Uso combinato del nome di stereotipo e del simbolo



# Sono possibili diverse notazioni



RegistraAccordoUI

# Attività di Analisi: dagli use case agli oggetti

- Le attività che consentono di trasformare gli use case e gli scenari della raccolta dei requisiti in un modello di analisi sono:
  - Identificare gli Oggetti Entity
  - Identificare gli Oggetti Boundary
  - Identificare gli Oggetti Control
  - Mappare gli Use Case in Sequence Diagram
  - Identificare le Associazioni
  - Identificare gli Aggregati
  - Identificare gli Attributi
  - Modellare il Comportamento dipendente dallo stato degli Oggetti individuali (statechart)
  - Modellare le Relazioni di Ereditarietà
  - Rivedere il Modello di Analisi
- Queste attività sono guidate da euristiche
- La qualità dei risultati dipende dall'esperienza degli sviluppatori nell'applicare le euristiche e i metodi

# Use Case Example: ReportEmergency

- Nome dello use case: ReportEmergency
- Attori Partecipanti :
  - Field Officer (Bob and Alice nello Scenario)
  - Dispatcher (John nello Scenario)
- Eccezioni:
  - Il FieldOfficer viene subito notificato se la connessione tra il suo terminale e quello della centrale cade.
  - Il Dispatcher viene subito notificato se la connessione tra ciascun FieldOfficer che ha avuto accesso e la centrale cade.
- Flow of Events: **on next slide.**
- Vincoli:
  - Il report del *FieldOfficer* deve essere comunicato entro 30 secondi
  - La risposta selezionata arriva non più tardi di 30 secondi dopo che è stato mandato dal *Dispatcher*

# Use case: ReportEmergency

Nome Use Case	ReportEmergency
Entry Condition	1. Il <i>FieldOfficer</i> attiva la funzione “ReportEmergency” dal suo terminale
Flusso degli eventi	<p>2. <i>FRIEND</i> risponde presentando un form al <i>FieldOfficer</i> Il form include un menu con i tipi di emergenza (emergenza generale, incendio), locazione, descrizione dell'incidente e richiesta di risorse</p> <p>3. Il <i>FieldOfficer</i> completa il form specificando il tipo di emergenza e i campi descrizione. Il <i>FieldOfficer</i> descrive anche possibili risposte alla situazione di emergenza e richiede risorse specifiche. Quando il form è completo, il <i>FieldOfficer</i> sottomette il form premendo il bottone “Send Report” a quel punto il Dispatcher è notificato</p> <p>4. Il <i>Dispatcher</i> rivede l'informazione sottomessa da <i>FieldOfficer</i> e crea un <i>Incident</i> nel database invocando lo use case <i>OpenIncident</i>. Tutta l'informazione contenuta nella form del <i>FieldOfficer</i> è inclusa automaticamente nell' <i>Incident</i>. Il <i>Dispatcher</i> seleziona una risposta allocando risorse all' <i>Incidente</i> (con lo use case <i>AllocateResource</i>) e informa della ricezione del report dell' incidente inviando un messaggio al <i>FieldOfficer</i></p>
Exit Condition	5. Il <i>FieldOfficer</i> riceve l' avviso di ricezione e le risposte selezionate

# Identificare gli Oggetti Entity

- Gli oggetti Partecipanti formano la base del modello di analisi
- Per individuare gli oggetti partecipanti si esaminano gli use case e si individuano i candidati
- **L'euristica di Abbott** si basa sull'analisi del linguaggio naturale per identificare oggetti, attributi, associazioni dalla specifica dei requisiti
  - mappano parti del parlato (nomi, verbo avere, verbo essere, aggettivi) per modellare componenti (oggetti, operazioni, relazioni di ereditarietà, classi)
- Vantaggio:
  - Ci si focalizza sui termini dell'utente
- Svantaggi:
  - Il linguaggio naturale è impreciso, anche il modello ad oggetti derivato rischia di essere impreciso
  - La qualità del modello dipende fortemente dallo stile di scrittura dell'analista
  - Ci possono essere molti più sostantivi delle classi rilevanti, corrispondenti a sinonimi o attributi
- Va bene per generare una lista iniziale di candidati a partire da una descrizione breve come il flusso di eventi di uno scenario o use case

# Euristica di Abbott

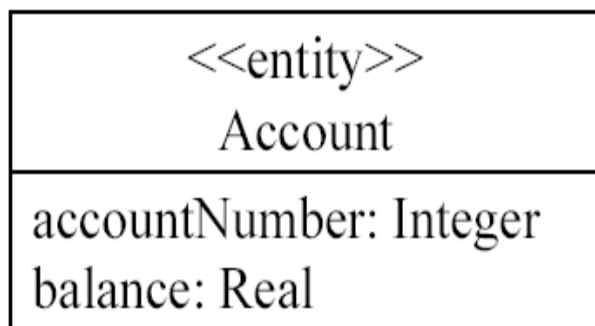
<b>Parti del parlato</b>	<b>Componente del modello</b>	<b>esempio</b>
Nome proprio	Istanza	Alice
Nome comune	Class	Funzionario(FieldOfficer)
Verbo fare/azione	Operazione	Crea, Sottoponi, Seleziona
Verbo essere	gerarchia	È un tipo di, è uno di
Verbo avere	aggregazione	Ha, consiste di, include
aggettivo	attributo	Descrizione dell' incidente

# Euristica per Identificare gli **Oggetti Entity**

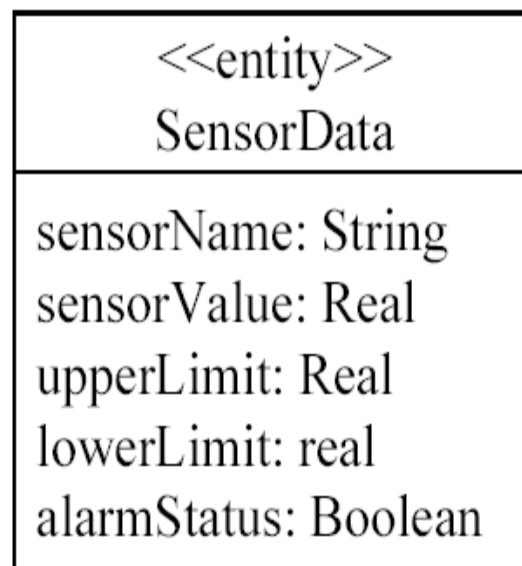
- In congiunzione con l'euristica di Abbott è possibile usare la seguente euristica:
  - **Termini** che gli sviluppatori e gli utenti hanno bisogno di chiarire per comprendere gli use case (information submitted by FieldOfficer)
  - **Sostantivi** ricorrenti negli use case (es. Incident)
  - **Entità** del mondo reale che il sistema deve considerare (es. FieldOfficer, Dispatcher, Resource)
  - **Attività** del mondo reale che il sistema deve considerare (es. EmergencyOperationPlan)
  - **Sorgenti o destinazioni** di dati (es. Printer)
- Per ogni oggetto identificato,
  - si assegna un nome (univoco) e una breve descrizione,
    - per gli oggetti Entity è opportuno utilizzare gli stessi nomi utilizzati dagli utenti e dagli specialisti del dominio applicativo
  - Si individuano attributi e responsabilità (non tutti, soprattutto non quelli ovvii)
  - Il processo è iterativo e varie revisioni saranno richieste: quando il modello di analisi sarà stabile sarà necessario fornire una descrizione dettagliata di ogni oggetto

# *Entity Object Examples*

Long-living objects that store information.



Account information in a  
bank system



Sensor data in a real-time system



# Identificazione degli Oggetti Entity

- Dall' esame dello use case ReportEmergency, dalla conoscenza del dominio e dalle interviste al cliente è possibile identificare i seguenti oggetti
  - Dispatcher
  - FieldOfficer
  - Incident
  - EmergencyReport
- Si noti che EmergencyReport non è nominato esplicitamente nello use case:
  - nel [passo 4](#) si nomina “informazione contenuta nella form del *FieldOfficer*”
  - e dal colloquio con il cliente si deduce che è proprio ciò che solitamente è detto “emergency report”

# Use case: ReportEmergency

Nome Use Case	ReportEmergency
Entry Condition	1. Il <i>FieldOfficer</i> attiva la funzione “ReportEmergency” dal suo terminale
Flusso degli eventi	<p>2. <i>FRIEND</i> risponde presentando un form al <i>FieldOfficer</i> <i>Il form include un menu con i tipi di emergenza (emergenza generale, incendio) e locazione, descrizione dell'incidente, richiesta di risorse)</i></p> <p>3. Il <i>FieldOfficer</i> completa il form <i>specificando il tipo di emergenza e i campi descrizione</i>. Il <i>FieldOfficer</i> descrive anche possibili risposte alla situazione di emergenza <i>e richiede risorse specifiche</i>. Quando il form è completo, il <i>FieldOfficer</i> sottomette il form premendo il bottone “Send Report” a quel punto il Dispatcher è notificato</p> <p>4. Il <i>Dispatcher</i> rivede l'informazione sottomessa da <i>FieldOfficer</i> e crea un <i>Incident</i> nel database invocando lo use case <i>OpenIncident</i>. Tutta <b>l'informazione contenuta nella form del <i>FieldOfficer</i></b> è inclusa automaticamente nell'incidente. Il <i>Dispatcher</i> seleziona una risposta allocando risorse all'incidente (con lo use case <i>AllocateResource</i>) e informa (acknowledge) della ricezione del report dell'incidente inviando un <i>FRIENDgram</i> al <i>FieldOfficer</i></p>
Exit Condition	5. Il <i>FieldOfficer</i> riceve l'avviso di ricezione e le risposte selezionate

# Identificazione degli Oggetti Entity dallo use case ReportEmergency

Dispatcher	Ufficiale di polizia che gestisce <i>Incidenti</i> . Un <i>Dispatcher</i> apre, documenta e chiude <i>Incident</i> in risposta a Report di Emergenza e altra comunicazione con <i>FieldOfficers</i> . I <i>Dispatcher</i> sono identificati dal numero del badge
EmergencyReport	Report iniziale su un <i>Incident</i> da un <i>FieldOfficer</i> a un <i>Disptacher</i> . Un <i>EmergencyReport</i> solitamente determina la creazione di un <i>Incident</i> da parte di un <i>Dispatcher</i> . Un <i>EmergencyReport</i> è composto da un livello di emergenza, un tipo (fuoco, stradale, ..), un luogo e una descrizione
FieldOfficer	Personale di un ufficio di polizia o dei vigili del fuoco in servizio. Un <i>FieldOfficer</i> può essere allocato al più ad un <i>Incident</i> alla volta. I <i>FieldOfficer</i> sono identificati da badge
Incident	Situazione che richiede l'attenzione di un <i>FieldOfficer</i> . Un <i>Incident</i> può essere riportato nel sistema da un <i>FieldOfficer</i> o da qualcuno anche esterno al sistema. Un <i>Incident</i> è composto da una descrizione, una risposta, uno status (aperto, chiuso, documentato), una locazione, e un numero di <i>FieldOfficer</i>

# Identificare gli Oggetti Boundary

- Gli oggetti Boundary rappresentano l' **interfaccia** del sistema con gli attori
  - In ogni use case, ogni attore interagisce almeno con un oggetto Boundary
  - L' oggetto Boundary colleziona informazione dall' attore e la traduce in una forma che può essere usata sia dagli oggetti Control che Entity
- Gli oggetti Boundary modellano l' interfaccia senza descriverne gli aspetti visuali!!!
  - Non ha senso parlare di “item di menu” o “scroll bar”
  - Lo sviluppo dell' interfaccia è solitamente di tipo prototipale e iterativo:
    - i test di usabilità fanno evolvere continuamente l' interfaccia  
➔ non sarebbe né pratico né utile modificare il modello di analisi ad ogni modifica dell' interfaccia

# Euristiche per Identificare gli Oggetti Boundary

- Identifica i **controlli della UI** di cui l'utente ha bisogno per iniziare lo use case (*ReportEmergencyButton*)
- Identifica **form** di cui l'utente ha bisogno per inserire dati nel sistema (*ReportEmergencyForm*)
- Identifica **avvisi e messaggi** che il sistema usa per rispondere all'utente (*AcknowledgmentNotice*)
- Quando più attori sono coinvolti in uno use case, identifica gli attori che sono **terminali** (*DispatcherStation*) per riferirti alla UI in considerazione
- Non modellare aspetti visuali della UI con oggetti Boundary (meglio mock-up)
- Usa ***sempre*** i termini dell'utente finale per descrivere l'interfaccia, non usare termini del dominio di implementazione

## Identificazione degli Oggetti *Boundary* dallo use case *ReportEmergency*

<i>AcknowledgmentNotice</i>	Avviso usato per mostrare l'acknowledgment del <i>Dispatcher</i> al <i>FieldOfficer</i>
<i>DispatcherStation</i>	Computer usato dal <i>Dispatcher</i>
<i>ReportEmergencyButton</i>	Bottone usato dal <i>FieldOfficer</i> per iniziare lo use case <i>ReportEmergency</i>
<i>EmergencyReportForm</i>	Form usata per l'input del <i>ReportEmergency</i> . Questa form è presentata al <i>FieldOfficer</i> sul <i>FieldOfficerStation</i> quando la funzione "Report Emergency" è selezionata. <i>EmergencyReportForm</i> contiene campi per specificare tutti gli attributi di un report di emergenza e un bottone (o altro controllo) per sottomettere la form completata.
<i>FieldOfficerStation</i>	Computer usato dal <i>FieldOfficer</i>
<i>IncidentForm</i>	Form usata per la creazione di <i>Incident</i> . Questa form è presentata al <i>Dispatcher</i> sul <i>DispatcherStation</i> quando è ricevuto l' <i>EmergencyReport</i> . Il <i>Dispatcher</i> usa anche questa form per allocare le risorse e notificare il report del <i>FieldOfficer</i>

# Identificare gli Oggetti Control

- Gli oggetti *Control* sono responsabili del **coordinamento** degli oggetti *Boundary* e *Entity*
  - Si preoccupano di collezionare informazione dagli oggetti *Boundary* e inviarla agli oggetti *Entity*
- Di solito non hanno una controparte nel mondo reale
- Spesso esiste una **stretta relazione** tra oggetti *Control* e use case:
  - Un oggetto *Control* è creato all'inizio dello use case e cessa di esistere alla fine
- **Euristiche**
  1. Identifica un oggetto *Control* per ogni use case
  2. Identifica un oggetto *Control* per ogni attore in uno use case
  3. La vita di un oggetto *Control* dovrebbe corrispondere alla durata di uno use case o di una sessione utente. Se è difficile identificare l'inizio e la fine dell'attivazione di un oggetto *Control*, il corrispondente use case probabilmente non ha delle entry e exit condition ben definite

# Identificare gli Oggetti Control dallo use case ReportEmergency

- Il flusso di controllo dello use case ReportEmergency viene modellato con due oggetti *Control* uno per ogni attore
  - *ReportEmergencyControl* per *FieldOfficer*
  - *ManageEmergencyControl* per *Dispatcher*
- Tale decisione deriva dalla consapevolezza che *FieldOfficerStation* e *DispatcherStation* sono due sottosistemi che comunicano su un link asincrono
  - Questa decisione potrebbe essere rimandata all'attività di design, comunque renderla visibile in fase di analisi consente di focalizzare l'attenzione su comportamenti eccezionali, come la perdita di comunicazione tra due stazioni
- Nel modellare lo use case ReportEmergency sono state modellate le stesse funzionalità usando oggetti *Boundary*, *Entity* e *Control*
  - Si è passati da una prospettiva “flusso di eventi” ad una “strutturale”
  - È aumentato il livello di dettaglio della descrizione
  - Sono stati selezionati termini standard per riferirci alle entità principali del dominio di applicazione e del sistema



# Oggetti Control dallo use case ReportEmergency

<b><i>ReportEmergencyControl</i></b>	<p>Gestisce la funzione <i>ReportEmergency</i> sulla <i>FieldOfficerStation</i>.</p> <p>Questo oggetto è creato quando il <i>FieldOfficer</i> seleziona il bottone “Report Emergency”. Crea un <i>EmergencyReportForm</i> e lo presenta a <i>FieldOfficer</i>. Dopo la sottomissione della form, questo oggetto colleziona l’informazione dalla form, crea un <i>EmergencyReport</i>, e lo inoltra al <i>Disptacher</i>. L’ oggetto Control quindi aspetta una notifica dal <i>DisptacherStation</i>. Quando riceve la notifica, l’ oggetto <i>ReportEmergencyControl</i> crea un <i>AcknowlegmentNotice</i> e lo mostra al <i>FieldOfficer</i></p>
<b><i>ManageEmergencyControl</i></b>	<p>Gestisce la funzione <i>ReportEmergency</i> sulla <i>DisptacherStation</i>. Questo oggetto è creato quando viene ricevuto un <i>EmergencyReport</i>. Quindi crea un <i>IncidentForm</i> e lo presenta al <i>Disptacher</i>. Quando il <i>Disptacher</i> ha creato un <i>Incident</i>, allocato <i>Resources</i>, e sottomesso una notifica, <i>ManageEmergencyControl</i> inoltra la notifica a <i>FieldOfficerStation</i></p>

# Mappare Use case in Oggetti con Sequence Diagram

- Un Sequence Diagram
  - mostra come il comportamento di uno use case (o scenario) è distribuito tra i suoi oggetti partecipanti
    - infatti vengono assegnate responsabilità a ogni oggetto in termini di un insieme di operazioni
  - Illustra la **sequenza di interazioni** tra gli oggetti necessaria per realizzare uno use case
    - non ci occupiamo di questioni di implementazioni, come l'efficienza!
- Non è adatto alla comunicazione con il cliente
- Per gli esperti è intuitivo e più preciso degli use case
- Fornisce una prospettiva diversa che consente di **individuare oggetti mancanti e aree non chiare** nelle specifiche

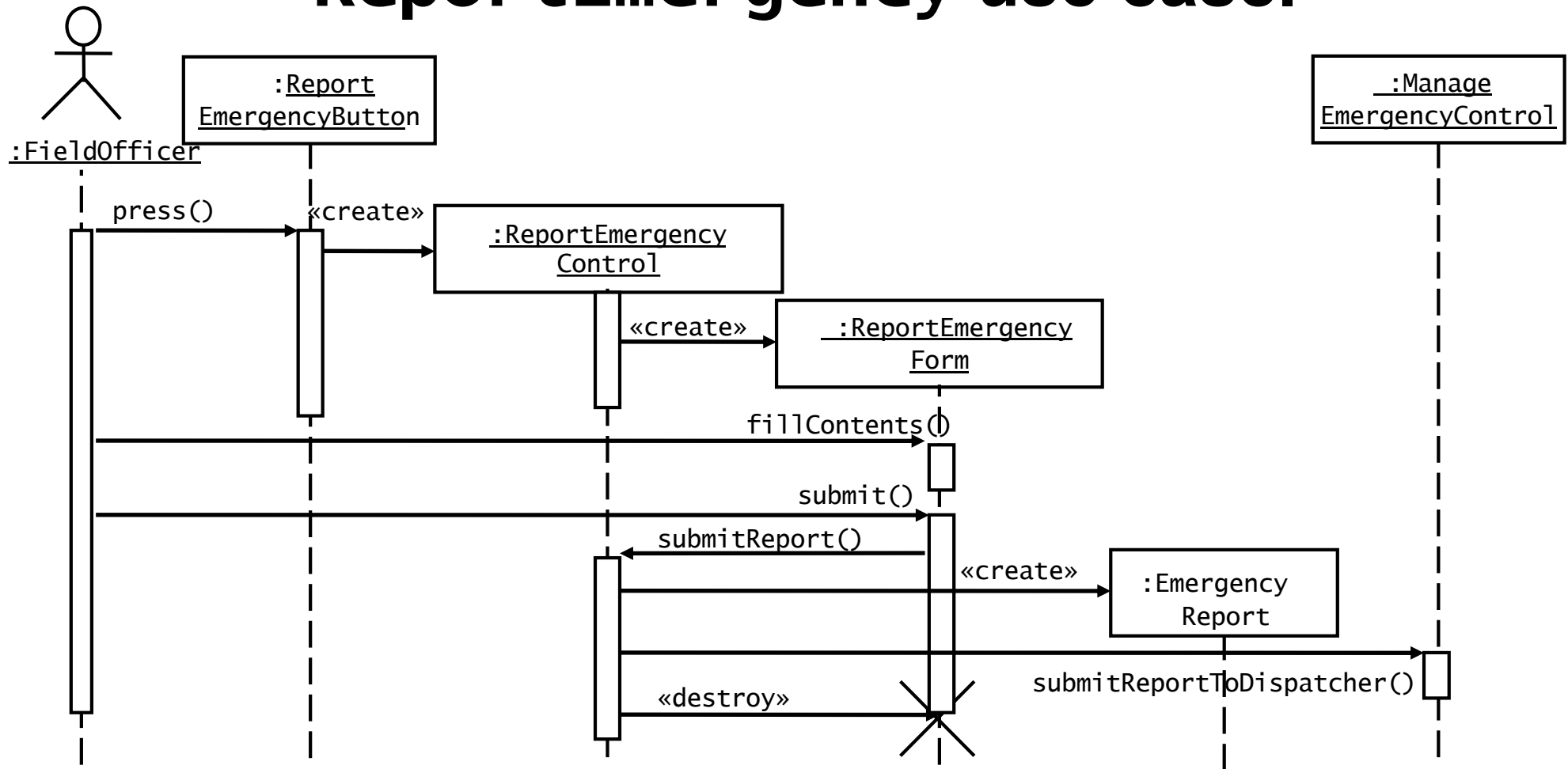
# Sequence Diagram

- Le colonne rappresentano gli oggetti che partecipano nello use case
  - La colonna più a sinistra rappresenta l'attore che inizia lo use case
  - La seconda colonna -> oggetto Boundary con cui l'attore interagisce per iniziare lo use case
  - La terza colonna -> oggetto Control che gestisce il resto dello use case
  - Gli oggetti Control creano altri oggetti Boundary e possono interagire con altri oggetti Control
- Le frecce orizzontali tra le colonne rappresentano messaggi o stimoli inviati da un oggetto ad un altro

# Sequence Diagram (cont.)

- La ricezione di un messaggio determina l'attivazione di un'operazione
  - L'attivazione è rappresentata da un rettangolo da cui altri messaggi possono prendere origine
  - La lunghezza del rettangolo rappresenta il tempo durante il quale l'operazione è attiva
  - Un'operazione è un servizio fornito ad altri oggetti
- La vita degli oggetti
  - Il tempo procede verticalmente dal top al bottom
  - Al top del diagramma si trovano gli oggetti che esistono prima del 1° messaggio inviato
  - Oggetti creati durante l'interazione sono illustrati con il messaggio <<create>>
  - Oggetti distrutti durante l'interazione sono evidenziati con una croce
  - La linea tratteggiata indica il tempo in cui l'oggetto può ricevere messaggi

# Sequence diagram for the ReportEmergency use case.



# Identificazione di nuovo oggetto

- Lo use case `ReportEmergency` è incompleto: menziona l'esistenza di un messaggio di notifica ma non descrive l'informazione ad essa associata
- C'è necessità di chiarire con il cliente → l'oggetto `Acknowledgment` è aggiunto al modello di analisi e lo use case è raffinato
- L'oggetto `Acknowledgment` è creato prima dell'oggetto *Boundary* `AcknowledgmentNotice`

# Identificazione di nuovo oggetto

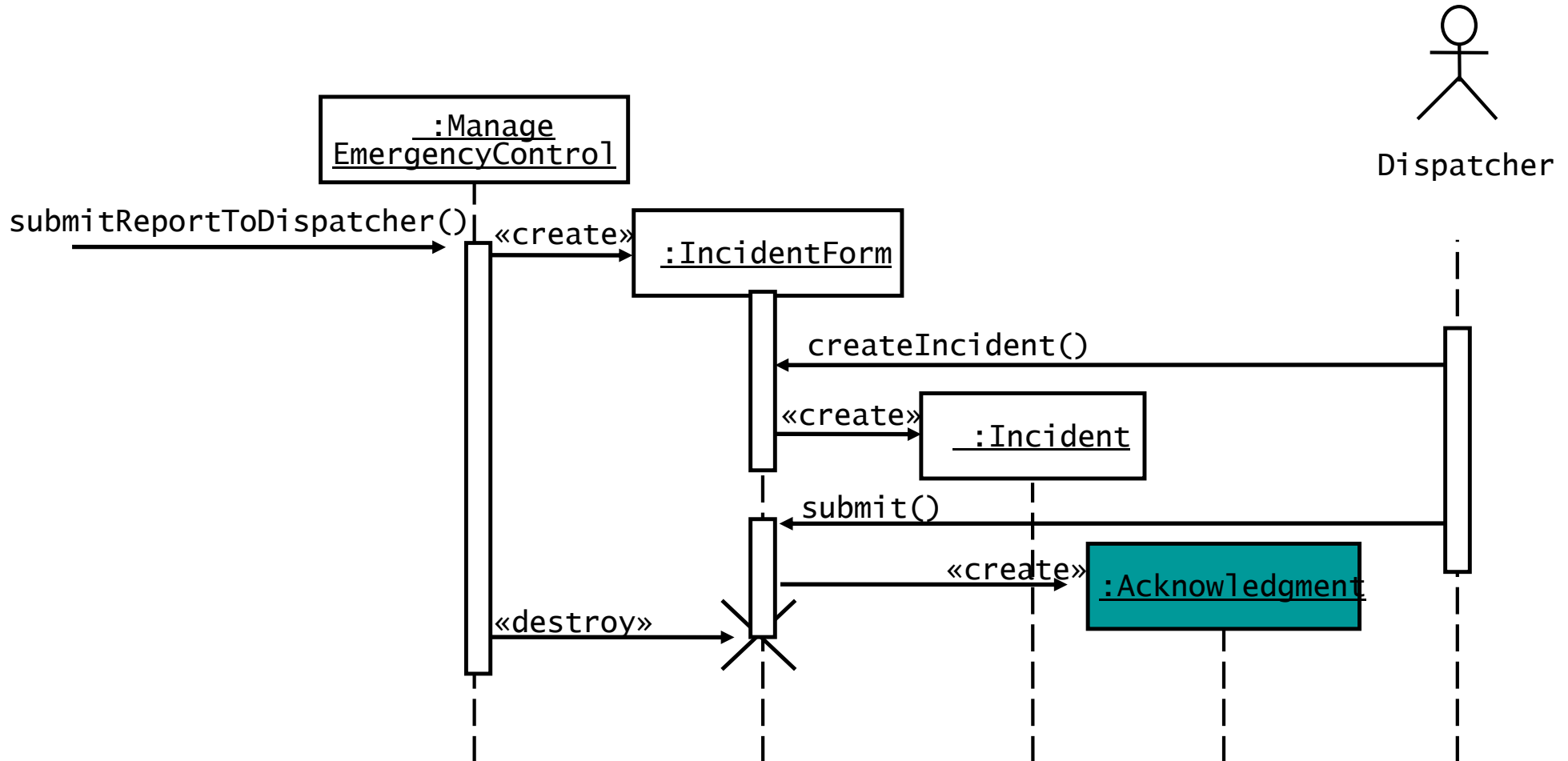
Acknowledgment	<p>Risposta di un Dispatcher a un EmergencyReport di un FieldOfficer.</p> <p>Inviando un Acknowledgment, il Dispatcher comunica al FieldOfficer che ha ricevuto l' EmergencyReport, crea un Incident, e assegna risorse.</p> <p>L' Acknowledgment contiene le risorse assegnate e il tempo stimato del loro arrivo</p>
----------------	--

# Use case ReportEmergency (Raffinato)

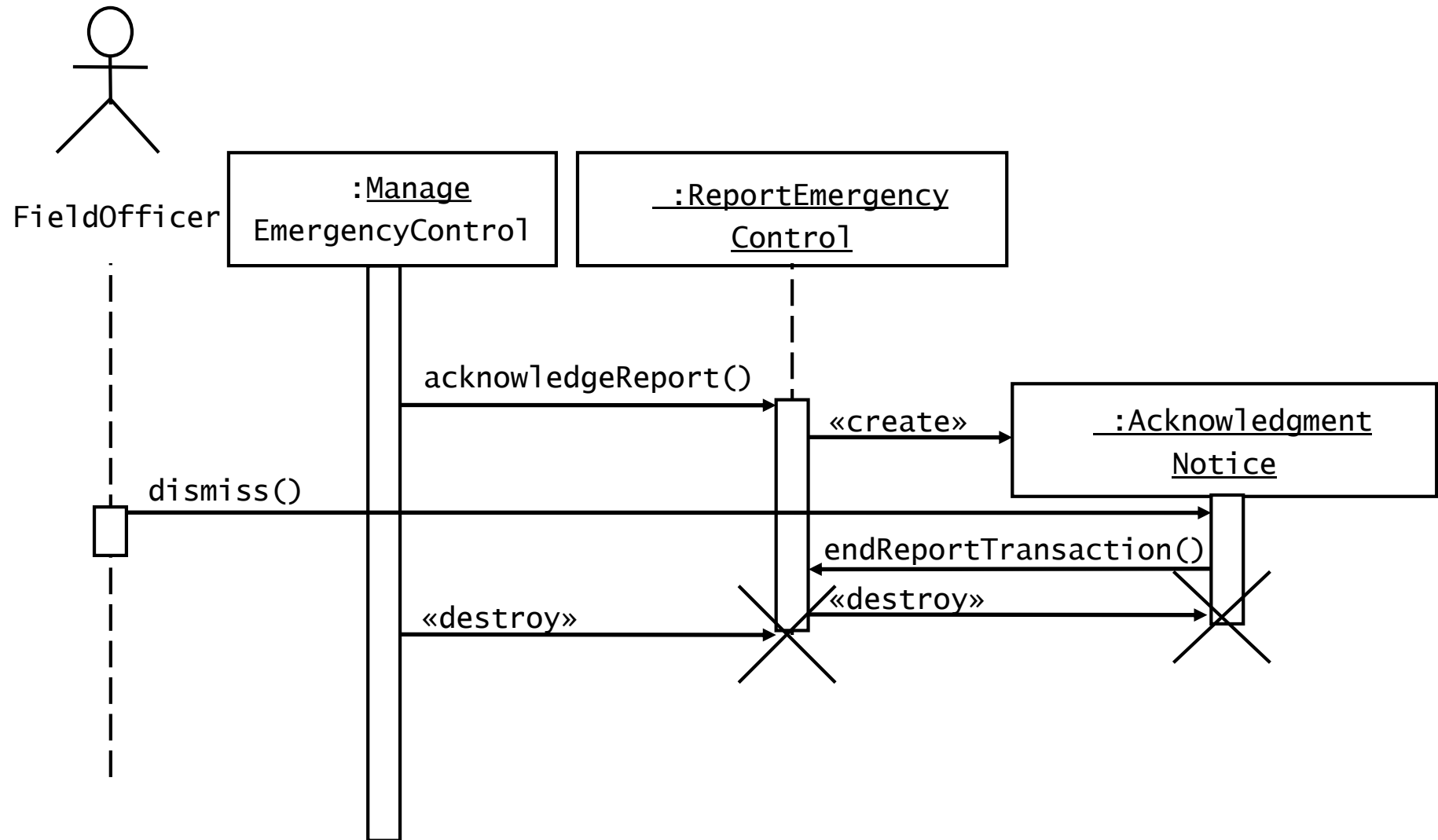
Nome Use Case	ReportEmergency
Entry Condition	1. Il <i>FieldOfficer</i> attiva la funzione “ReportEmergency” dal suo terminale
Flusso degli eventi	<p>2. <i>FRIEND</i> risponde presentando un form al <i>FieldOfficer</i> <i>Il form include un menu con i tipi di emergenza (emergenza generale, incendio) e locazione, descrizione dell'incidente, richiesta di risorse)</i></p> <p>3. Il <i>FieldOfficer</i> completa il form <i>specificando il tipo di emergenza e i campi descrizione</i>. Il <i>FieldOfficer</i> descrive anche possibili risposte alla situazione di emergenza <i>e richiede risorse specifiche</i>. Quando il form è completo, il <i>FieldOfficer</i> sottomette il form premendo il bottone “Send Report” a quel punto il Dispatcher è notificato</p> <p>4. Il <i>Dispatcher</i> rivede l'informazione sottomessa da <i>FieldOfficer</i> e crea un <i>Incident</i> nel database invocando lo use case <i>OpenIncident</i>. Tutta l'informazione contenuta nella form del <i>FieldOfficer</i> è inclusa automaticamente nell'incidente. Il <i>Dispatcher</i> seleziona una risposta allocando risorse all'incidente (con lo use case <i>AllocateResource</i>) e informa della ricezione del report dell'incidente inviando un breve messaggio al <i>FieldOfficer</i>. <b><i>L'acknowledgment indica al FieldOfficer che l'EmergencyReport è stato ricevuto, creato un Incident, e allocate risorse a Incident. L'acknowledgment include le risorse e il tempo stimato del loro arrivo.</i></b></p>
Exit Condition	5. Il <i>FieldOfficer</i> riceve l'avviso di ricezione e le risposte selezionate40



# Sequence diagram for the ReportEmergency use case (continued).



# Sequence diagram for the ReportEmergency use case (continued).



# Euristiche per disegnare un Sequence Diagram

- La 1° colonna dovrebbe corrispondere all'attore che inizia lo use case
- La 2° colonna dovrebbe essere un oggetto *Boundary* (che l'attore usa per iniziare lo use case)
- La 3° colonna dovrebbe essere l'oggetto *Control* che gestisce il resto dello use case
- Gli oggetti *Control* sono creati dagli oggetti *Boundary* che iniziano gli use case
- Oggetti *Control* e *Boundary* accedono a oggetti *Entity*
- Gli oggetti *Boundary* sono creati da oggetti *Control*
- Gli oggetti *Entity* non accedono **mai** agli oggetti *Control* e *Boundary* : ciò rende più facile condividere oggetti *Entity* tra più use case

# ***Cambiamenti al ReportEmergency***

- *New Entity object: Acknowledgment – Response of a Dispatcher to a FieldOfficer's EmergencyReport. Contains resources allocated, predicted arrival time ...*
- *Modify Step 4 of ReportEmergency flow of events' description: The acknowledgment indicates to the FieldOfficer that the EmergencyReport was received, an Incident created, and resources allocated to the Incident.*

# **Object** *Responsibilities*

- Sequence diagrams imply we distribute the behavior of the use case across participating objects.
- Responsibilities, under the form of operations, are assigned to objects
- Queste operazioni possono essere condivise tra diversi Use Case: ciò consente di eliminare ridondanza e migliorare la consistenza.
- During analysis, sequence diagrams only focus on high level behavior – implementation issues should not be addressed at this point

# ***Specifying Responsibilities***

- *Pre-condition*: Conditions under which operations can be executed and yield a correct result
- *Post-Condition*: Conditions that are guaranteed true after execution of an operation
- *Class invariant*: Conditions that must remain true, at all times, for any instance of a class
- *Contract*: All of the above are referred to as a contract
- Next course section will discuss how contracts can be precisely defined

# Analisi e Sequence Diagram

- Durante l'analisi i Sequence Diagram sono usati per individuare
  - nuovi oggetti
  - comportamenti mancanti
- Disegnare Sequence Diagram è un'attività laboriosa, quindi
  - Occorre dare priorità a quelle funzionalità problematiche o non ben specificate
  - Per le parti ben definite può essere utile solo per evitare di posticipare alcune decisioni chiave

# ***Cross-Checking***

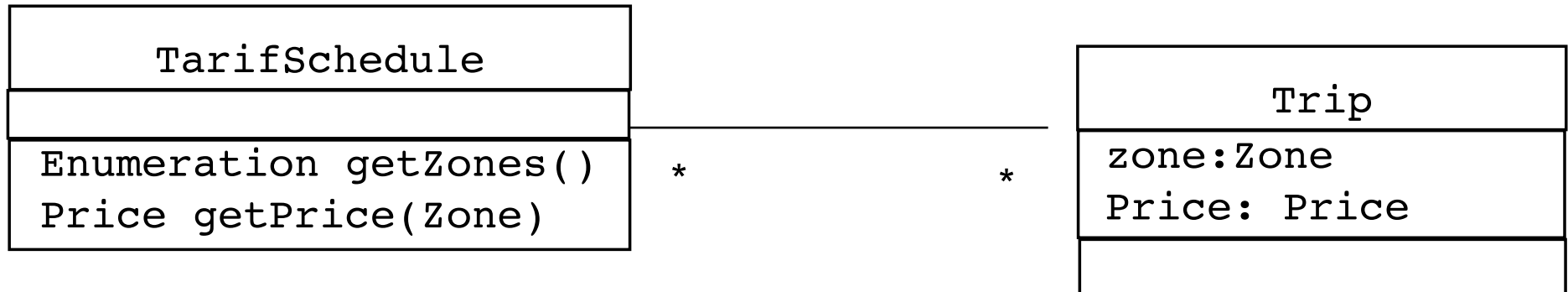
- Sequence diagrams can be used to help check the completeness / correctness of the use case model and class diagrams.
- Which Use Cases create this object? Which actors can access this information?
- Which Use Cases modify and destroy this object? Which actors initiate these Use Cases?
- Is this object Needed? (at least one Use Case depends on this information)



# Class Diagram

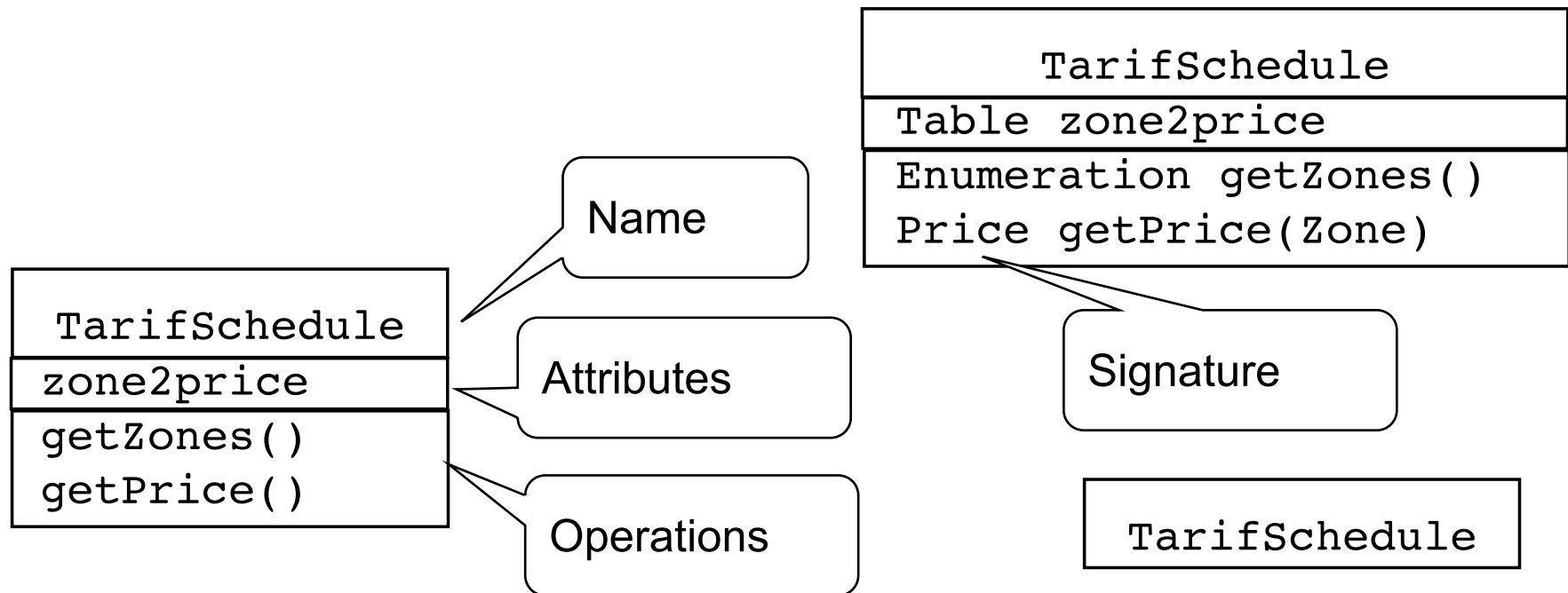
- Il sequence diagram consente di illustrare le interazioni tra gli oggetti
- Il **class diagram** mostra le relazioni tra gli oggetti
  - Associazioni
  - Ereditarietà
  - Aggregazione
- Identificare le Associazioni consente di identificare casi limiti (che devono essere chiariti con il cliente)
  - È intuitivo assumere che gli EmergencyReport siano scritti da un solo FieldOfficer
  - Il sistema può supportare che siano scritti da più di un FieldOfficer?
  - Ci possono essere EmergencyReport scritti da anonimi?
  - Queste questioni devono essere identificate durante l'analisi e chiarite con il cliente

# Class Diagrams



- Class diagrams represent the structure of the system.
- Used
  - during requirements analysis to model problem domain concepts
  - during system design to model subsystems and interfaces
  - during object design to model classes.

# Classes



- A **class** represent a concept
- A class encapsulates state (**attributes**) and behavior (**operations**).
- Each attribute has a **type**.
- Each operation has a **signature**.
- The class name is the only mandatory information.

# Instances

```
tarif_1974:TarifSchedule
```

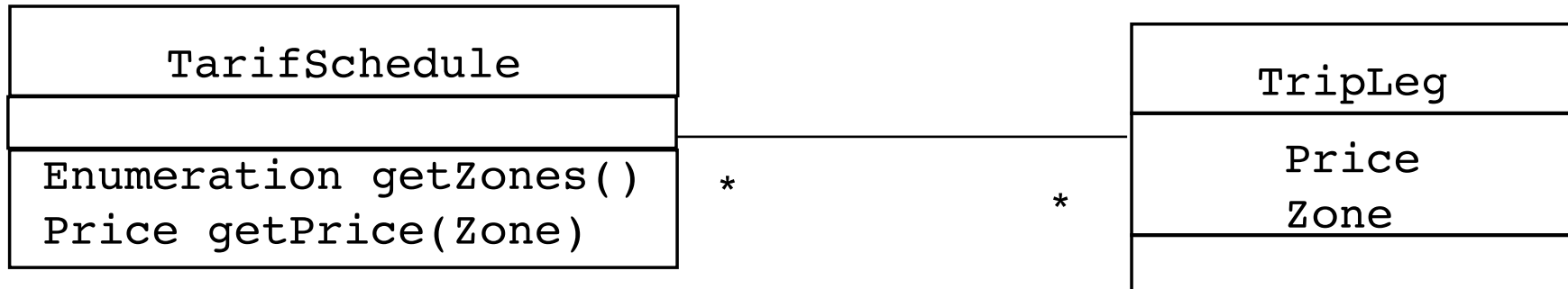
```
zone2price = {  
    {'1', .20},  
    {'2', .40},  
    {'3', .60}}
```

- An ***instance*** represents a phenomenon.
- The name of an instance is underlined and can contain the class of the instance.
- The attributes are represented with their ***values***.

# Actor vs Instances

- What is the difference between an *actor* , a *class* and an *instance*?
- Actor:
  - An entity outside the system to be modeled, interacting with the system (“Passenger”)
- Class:
  - An abstraction modeling an entity in the problem domain, must be modeled inside the system (“User”)
- Object:
  - A specific instance of a class (“Joe, the passenger who is purchasing a ticket from the ticket distributor”).

# Associations



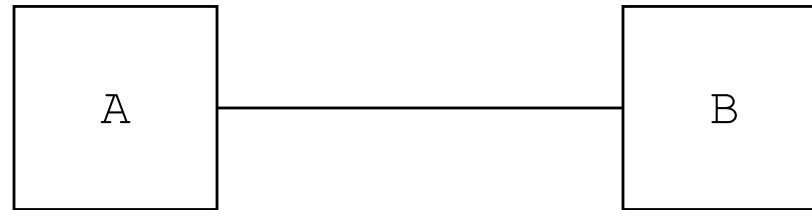
- Associations denote relationships between classes.
- The multiplicity of an association end denotes how many objects the source object can legitimately reference.

# Identificare Associazioni

- Link stabiliscono relazione tra oggetti
- **Link:**
  - Una connessione tra istanze di oggetti. Un link è come una tupla.
  - Un link è una istanza di associazione
- **Associazione:**
  - Fondamentalmente è un mapping bidirezionale.
  - Un'associazione descrive un insieme di link come una classe descrive un insieme di oggetti.

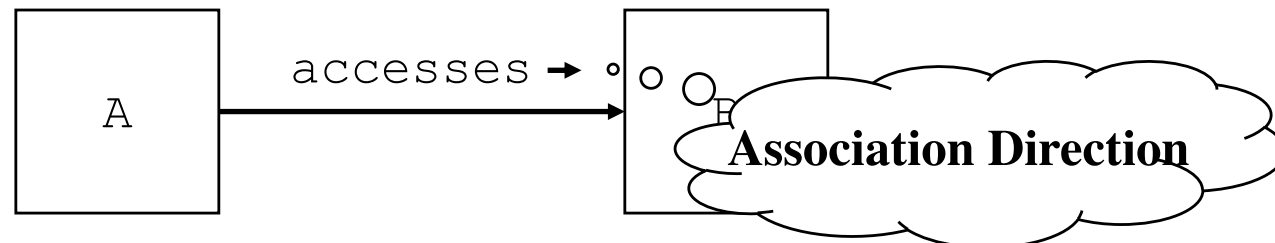
# Do UML associations have direction?

- An association between two classes is by default a bi-directional mapping.



- Class A can access class B and class B can access class A
- Both classes play the agent role.

If you make class A a client, and B a server, you can make the association directional. The arrowhead points to the server role:



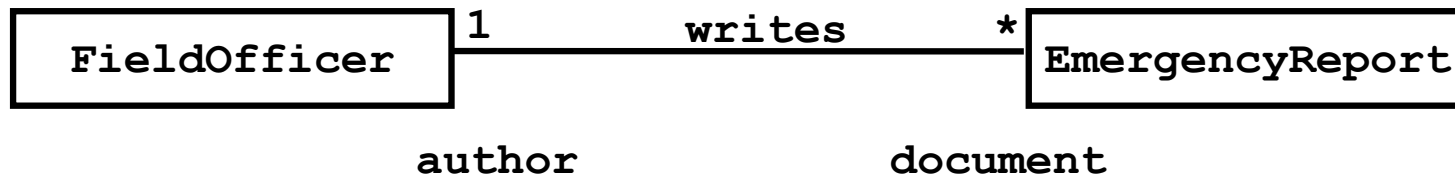
Class A ( the “client”) accesses class B (“the server”). B is also called *navigable*



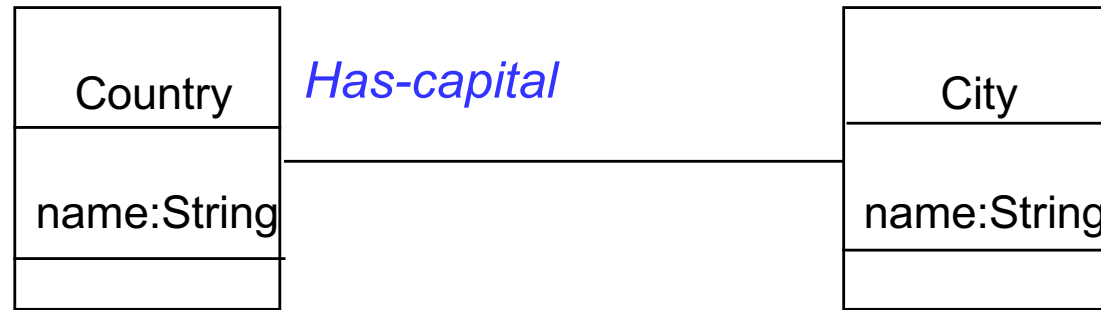
# Proprietà delle associazioni

- Nome
  - Unico e identificabile
- Ruolo
  - Identifica la funzione di ogni classe
- Molteplicità
  - 1-to-1, many-to-1, 1-to-many,

An example of association between the  
EmergencyReport and the  
FieldOfficer classes.



# 1-to-1 and 1-to-many Associations

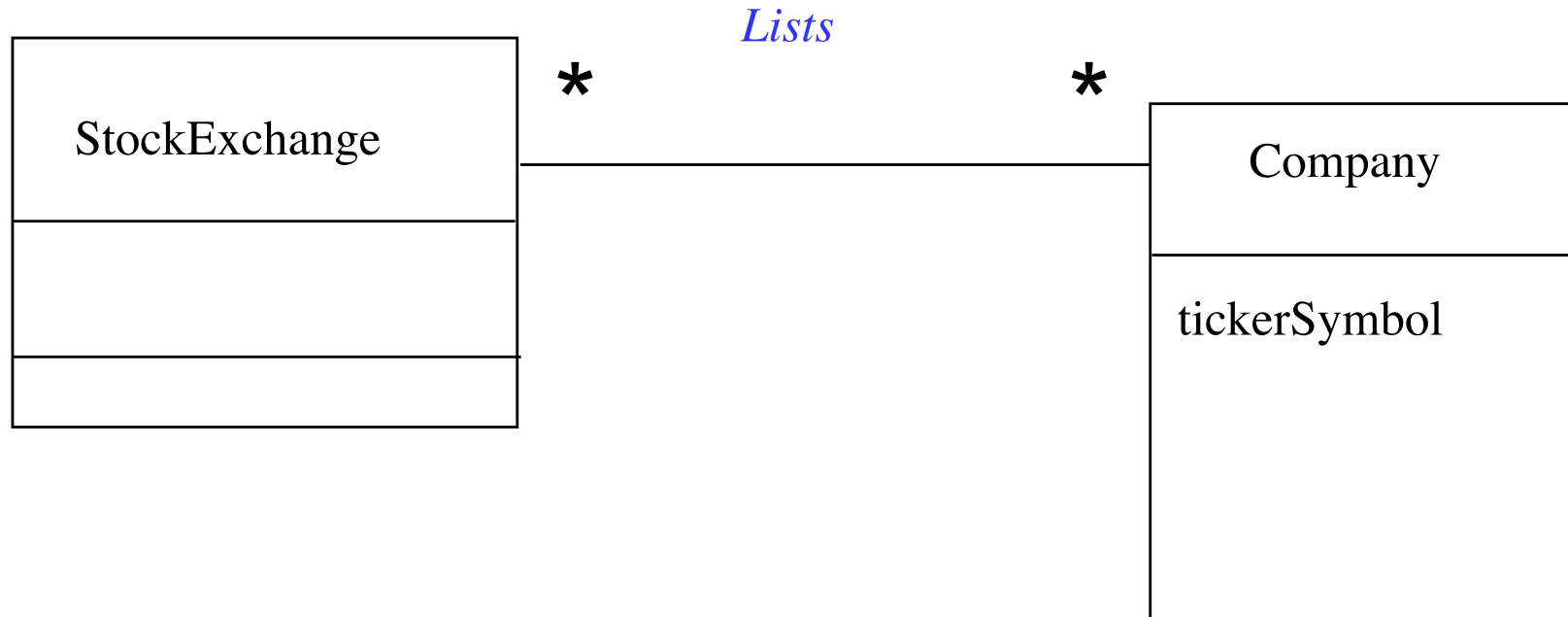


One-to-one association



One-to-many association

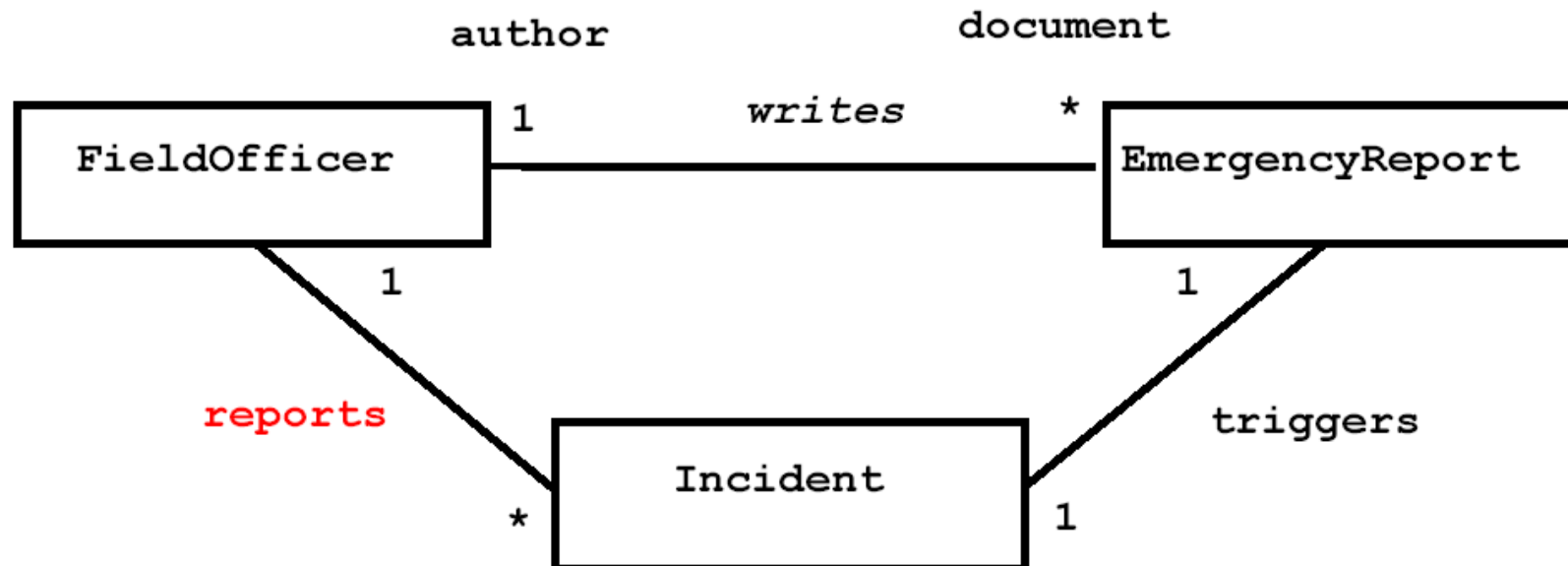
# Many-to-Many Associations



# Euristiche per Identificare le associazioni

- Esaminare i verbi nelle frasi (ha, è parte di, gestisce, riporta a, è iniziata da, è contenuta in, parla di, include...)
- Nominare in modo preciso i nomi delle associazioni e i ruoli
- Eliminare associazioni che possono essere derivate da altre associazioni
- Non preoccuparsi della molteplicità delle associazioni fino a quando l'insieme delle associazioni non è stabile
- Troppe associazioni rendono il modello degli oggetti “illeggibile”

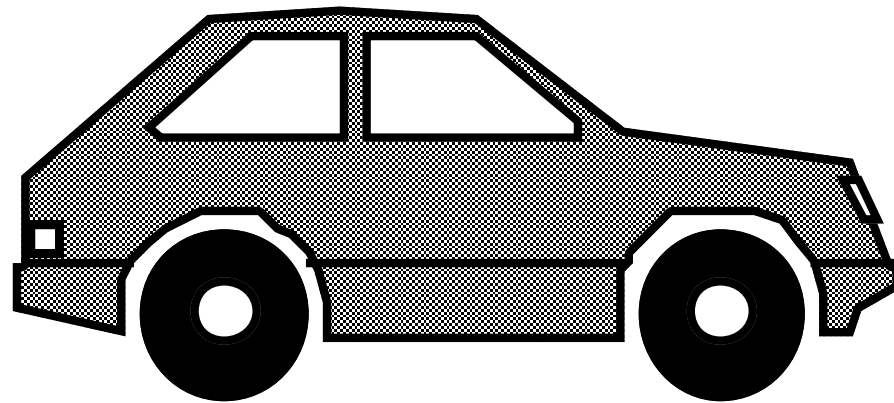
# ***FRIEND Example***



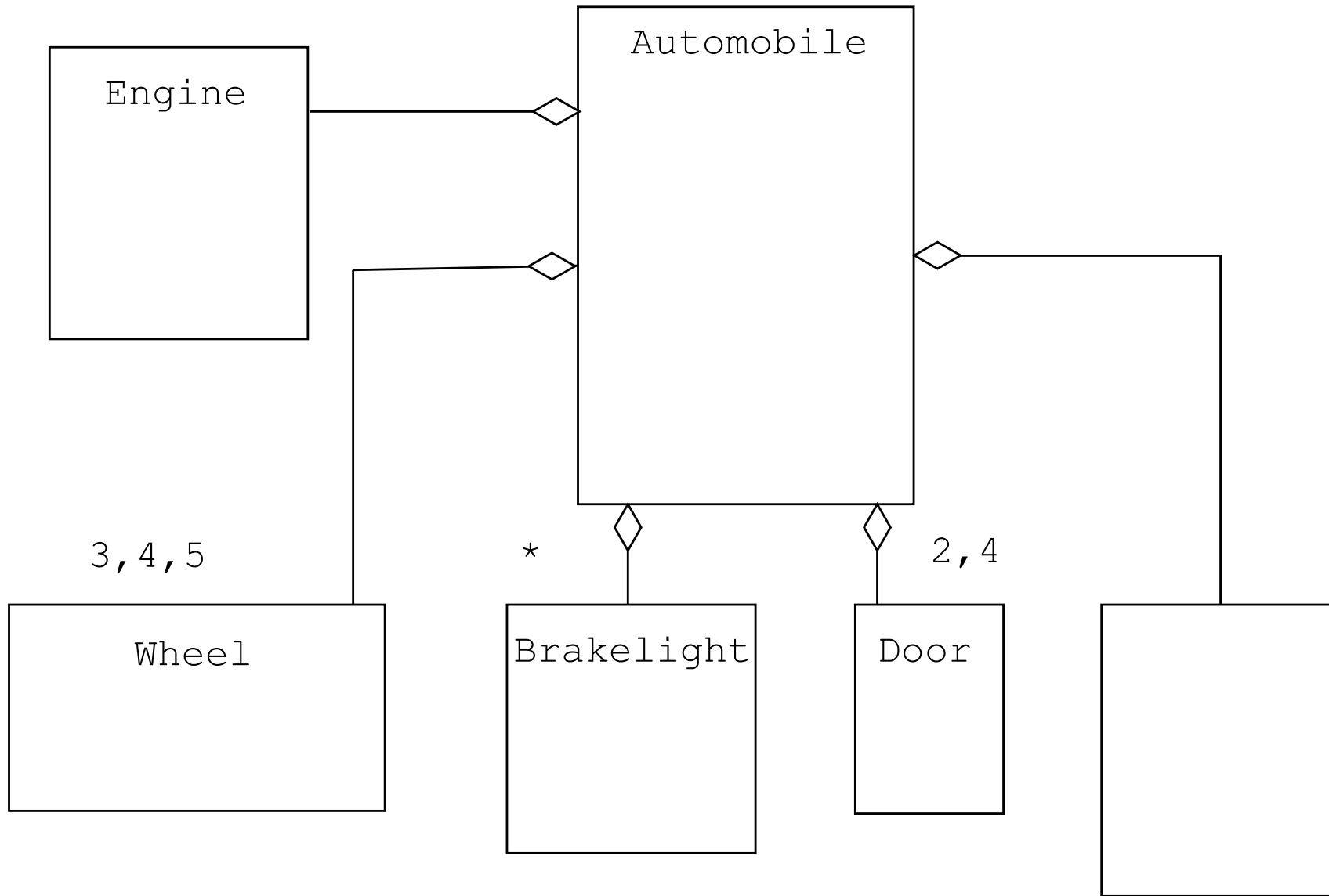
- Association necessary?

# Identificare le Aggregazioni

- Modella una gerarchia: “*È parte di*”
- notazione UML : come un’associazione ma con un rombo che indica la parte assemblata della relazione



# Esempio di aggregazione



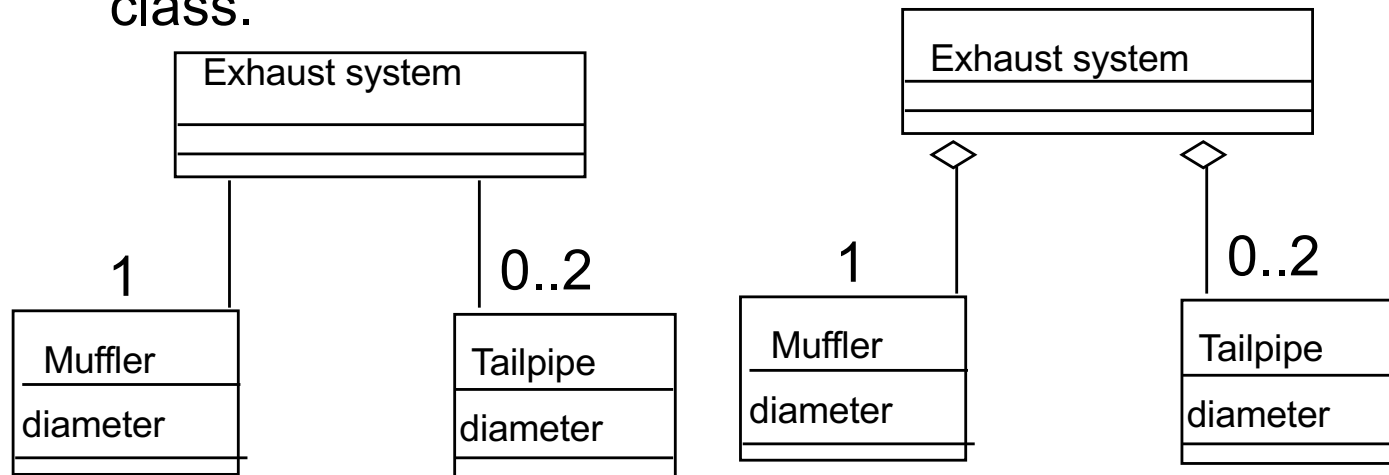


# Identificare aggregazioni

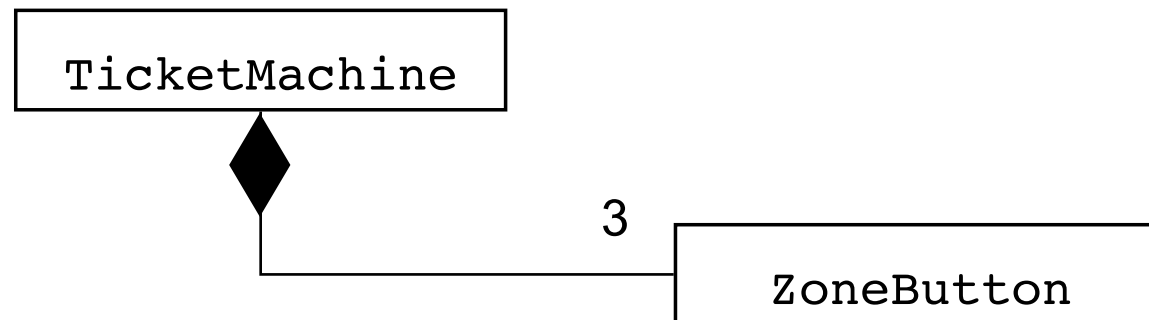
- Associazioni di aggregazione aggiungono informazioni al modello di analisi su come i concetti di contenimento nel dominio di applicazione possono essere organizzati
- Se non è sicuri che l'associazione che si sta descrivendo sia un'aggregazione (un concetto intero-parti) è meglio modellarla come associazione 1-Many e poi rivederla quando si ha maggiore conoscenza del dominio

# Aggregation

- An **aggregation** is a special case of association denoting a “consists of” hierarchy.
- The **aggregate** is the parent class, the **components** are the children class.

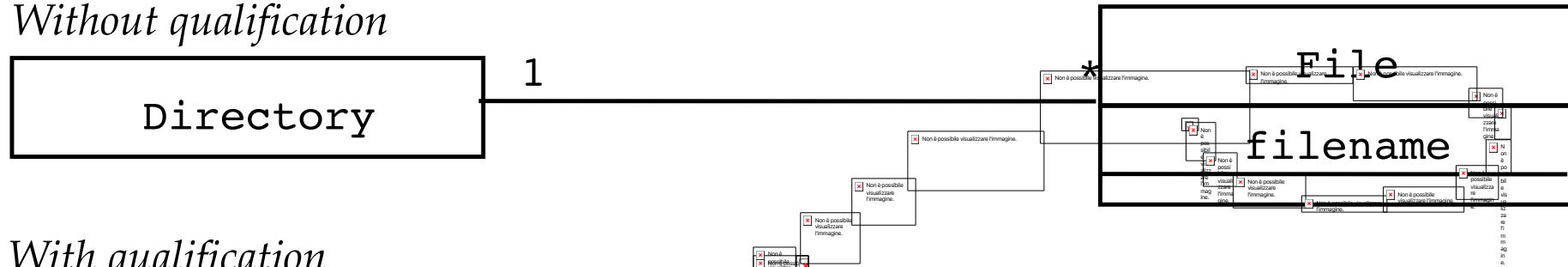


- A solid diamond denotes **composition**, a strong form of aggregation where components cannot exist without the aggregate. (Bill of Material)

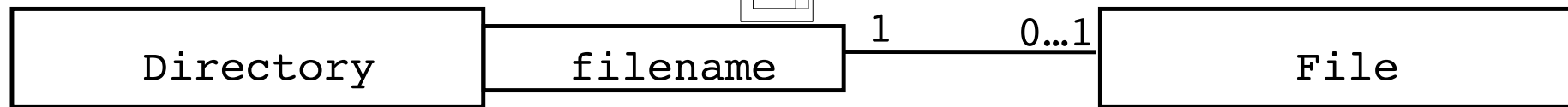


# Qualifiers

*Without qualification*

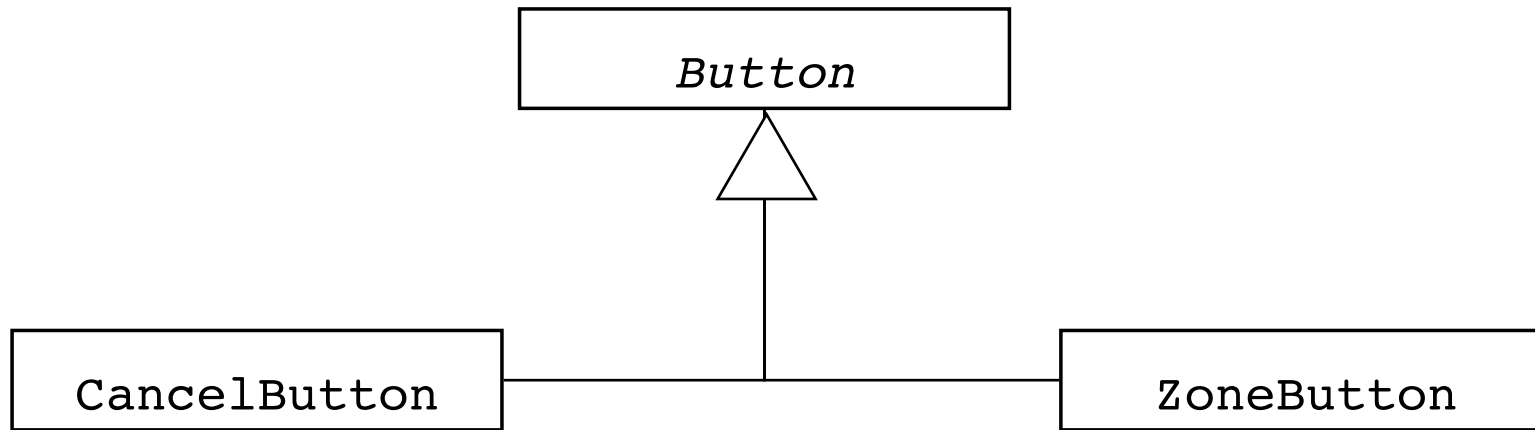


*With qualification*



- Qualifiers can be used to reduce the multiplicity of an association.

# Inheritance

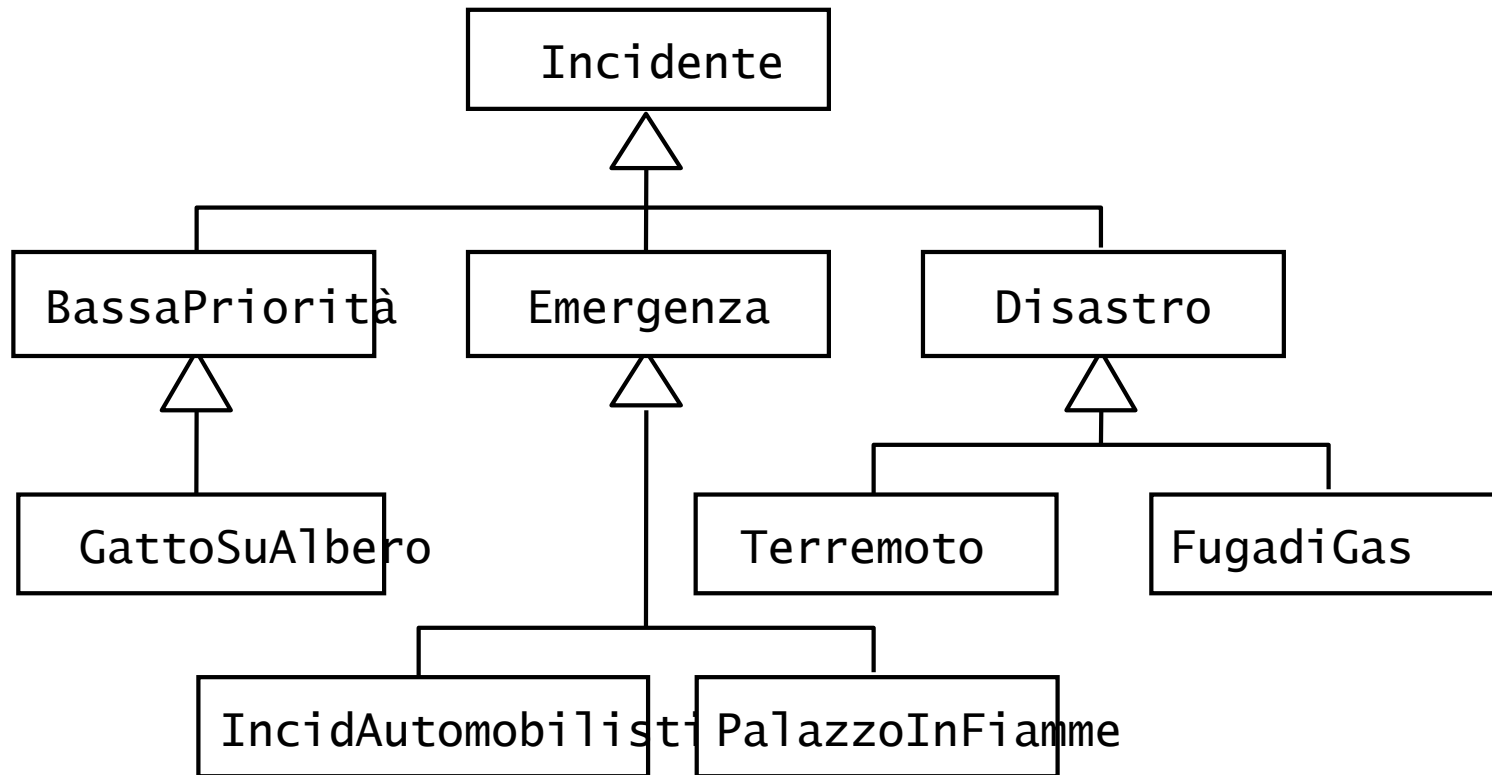


- The **children classes** inherit the attributes and operations of the **parent class**.
- Inheritance simplifies the model by eliminating redundancy.

# Generalizzazione e specializzazione

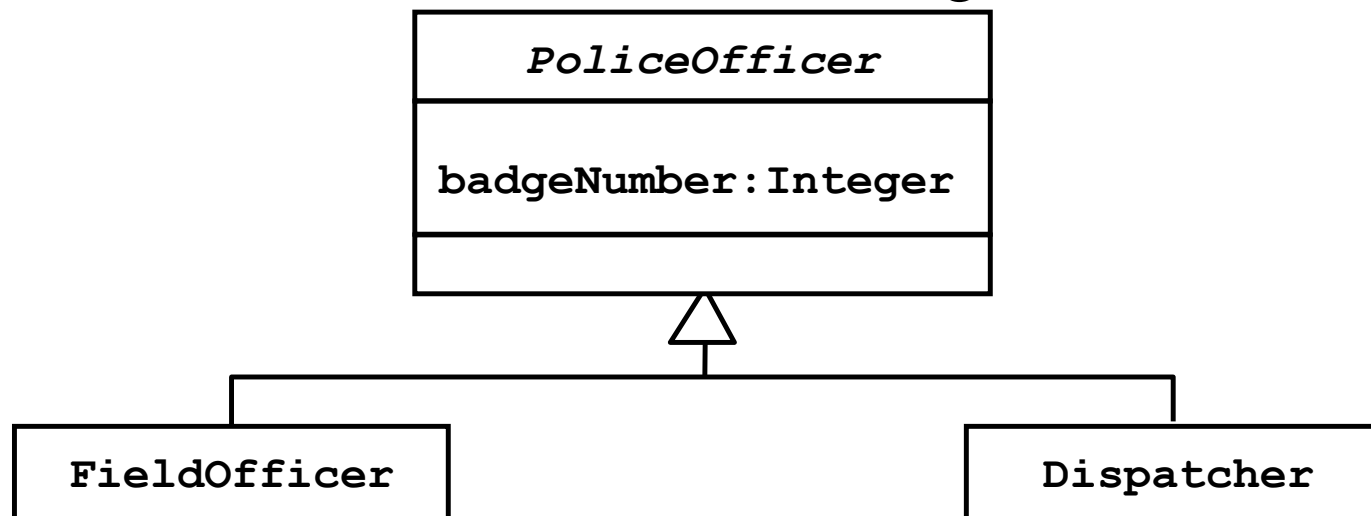
- *Ereditarietà*: consente di organizzare concetti in gerarchie:
  - Al top della gerarchia concetti più generali
  - Al bottom concetti più specializzati
- *Generalizzazione*: attività di modellazione che identifica concetti astratti da quelli di più basso livello
  - Es. Stiamo facendo reverse-engineering di un sistema di gestione delle emergenze e analizzando le videate per la gestione di incidenti autostradali e incendi. Osservando concetti comuni, creiamo un concetto astratto Emergenze
- *Specializzazione*: attività che identifica concetti più specifici da quelli di più alto livello
  - ES. Stiamo costruendo un sistema di gestione delle emergenze e stiamo discutendo le funzionalità con il cliente: il cliente introduce prima il concetto di incidente, quindi descrive tre tipi di incidenti: disastri, emergenze, incidenti a bassa priorità
- Come risultato sia della specializzazione che della generalizzazione abbiamo la specifica di ereditarietà tra concetti

# Un esempio di gerarchia.



# Modellare relazione di ereditarietà tra gli oggetti

- Modella una gerarchia; ***“E’ uno di”***
- Una notazione potente per condividere similarità tra classi preservando le loro differenze
- *Esempio:*
  - *Dispatchers* and *Fieldofficers*: both have *badgeNumber* to identify them within the city. They are both *PoliceOfficer*
  - Abstract *PoliceOfficer* class, containing common functionality and attributes
- Notazione UML : un arco con un triangolo



# Identificare Attributi

- Gli attributi sono proprietà individuali degli oggetti – Solo gli attributi rilevanti al sistema dovrebbero essere identificati
  - Nome, che lo identifica
  - Breve descrizione
  - Tipo, che descrive i valori possibili che può assumere
- Gli attributi possono essere aggiunti anche dopo che l'analisi è finita.
  - Non sono direttamente legati alle funzionalità del sistema
  - Gli sviluppatori in questa fase non devono spendere molto del loro tempo per l'identificazione degli attributi
  - Le associazioni dovrebbero essere identificate prima degli attributi per evitare confusione



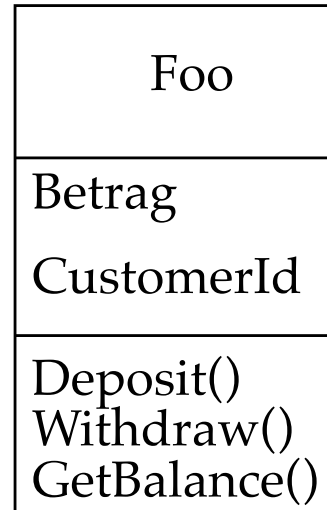
# Attributes of the EmergencyReport class.

EmergencyReport
<code>emergencyType:{fire,traffic,other}</code> <code>location:String</code> <code>description:String</code>

# ***Euristiche***

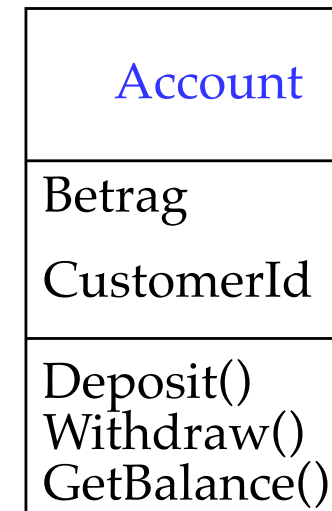
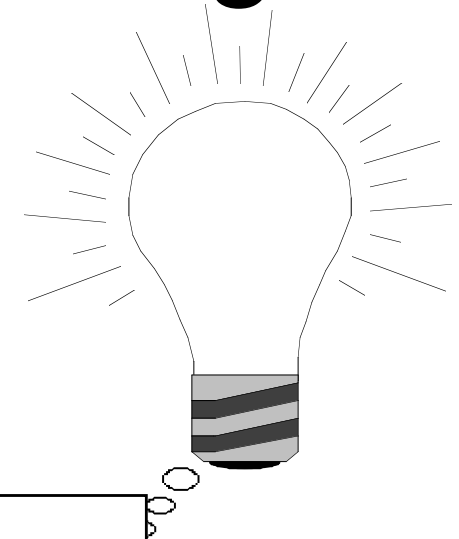
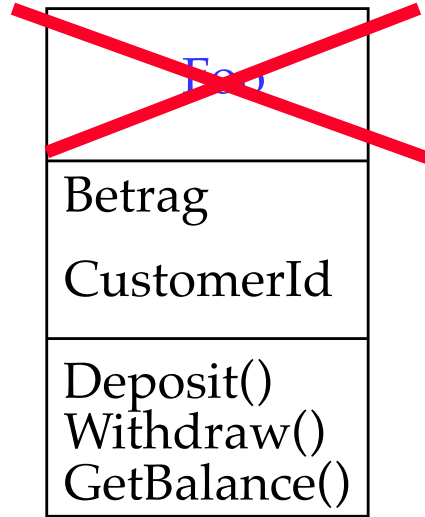
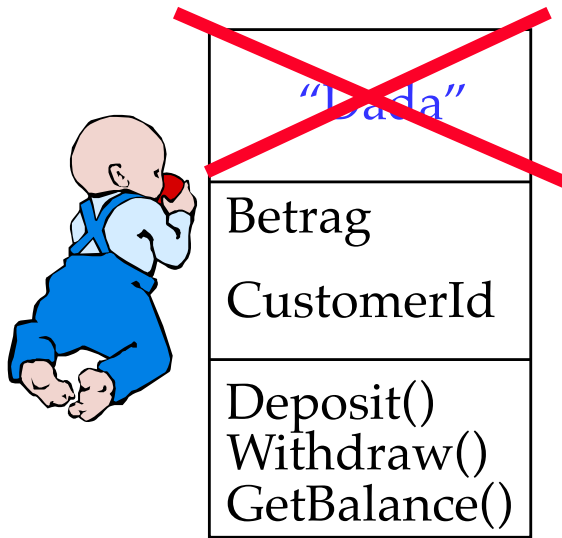
- Frasi possessive o aggettivi
- Rappresenta lo stato memorizzato come un attributo di un oggetto entità
- Guarda i contratti delle operazioni
- Descrivi ogni attributo nel dizionario dei dati
- Non perdere tempo a descrivere i dettagli fino a quando il modello a oggetti non è stabile

# Object Modeling in Practice: Class Identification



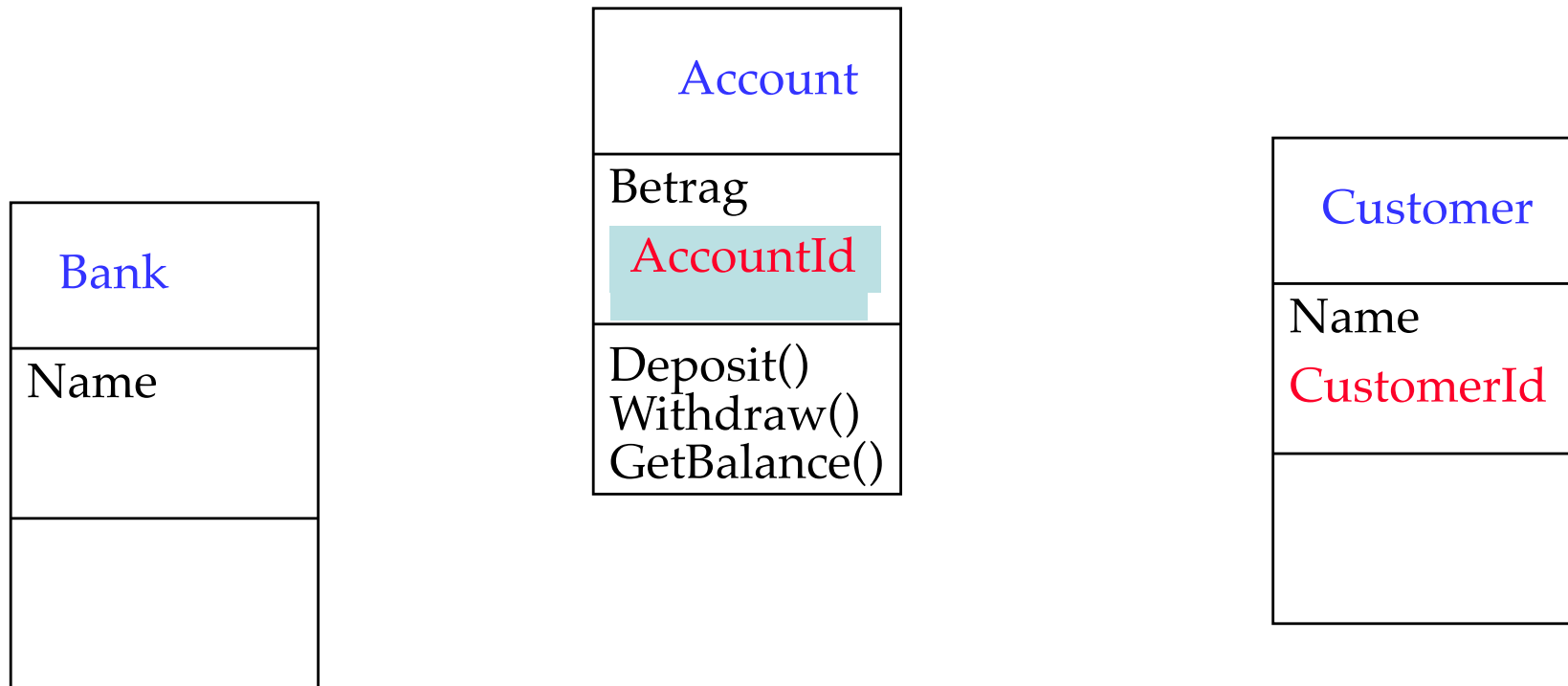
Class Identification: Name of Class, Attributes and Methods

# Object Modeling in Practice: Encourage Brainstorming



Naming is important!  
Is **Foo** the right name?

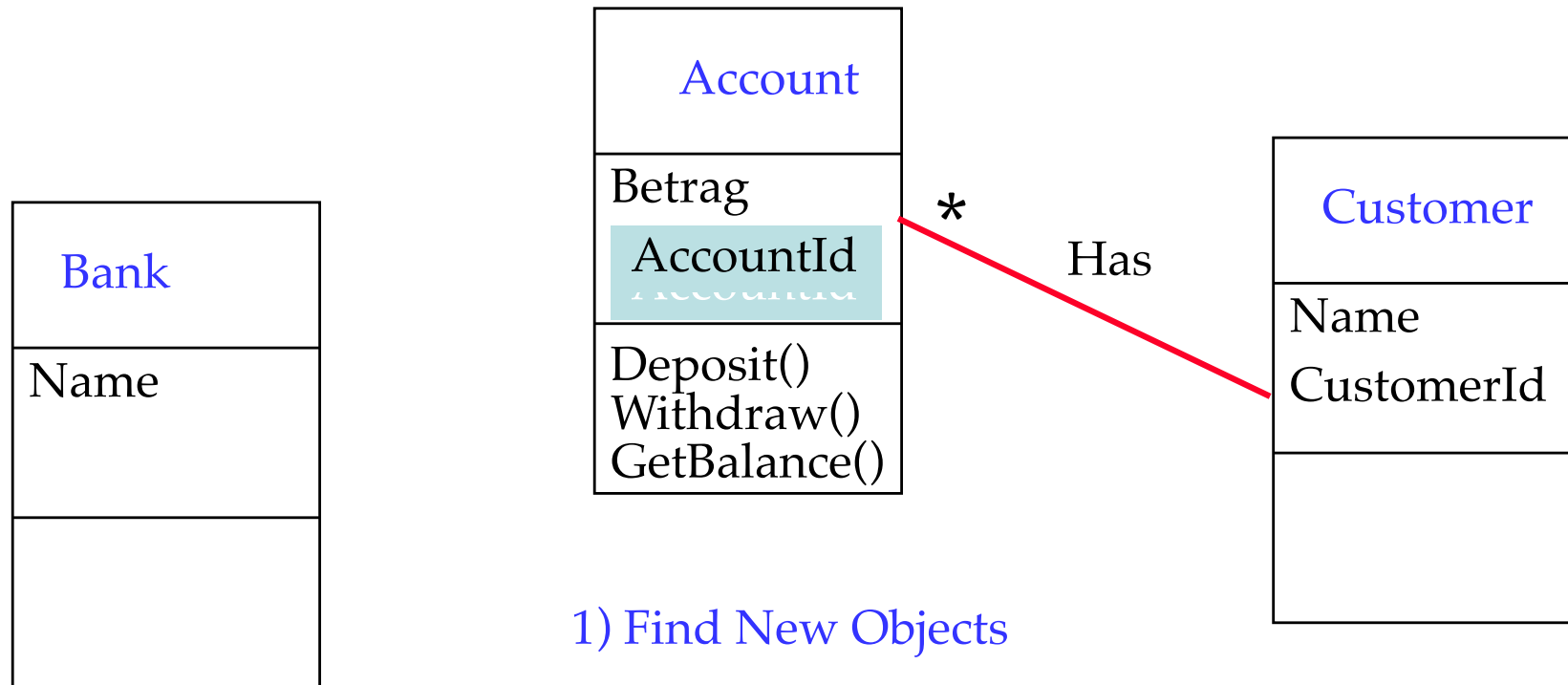
# Object Modeling in Practice ctd



1) Find New Objects

2) Iterate on Names, Attributes and Methods

# Object Modeling in Practice: A Banking System



1) Find New Objects

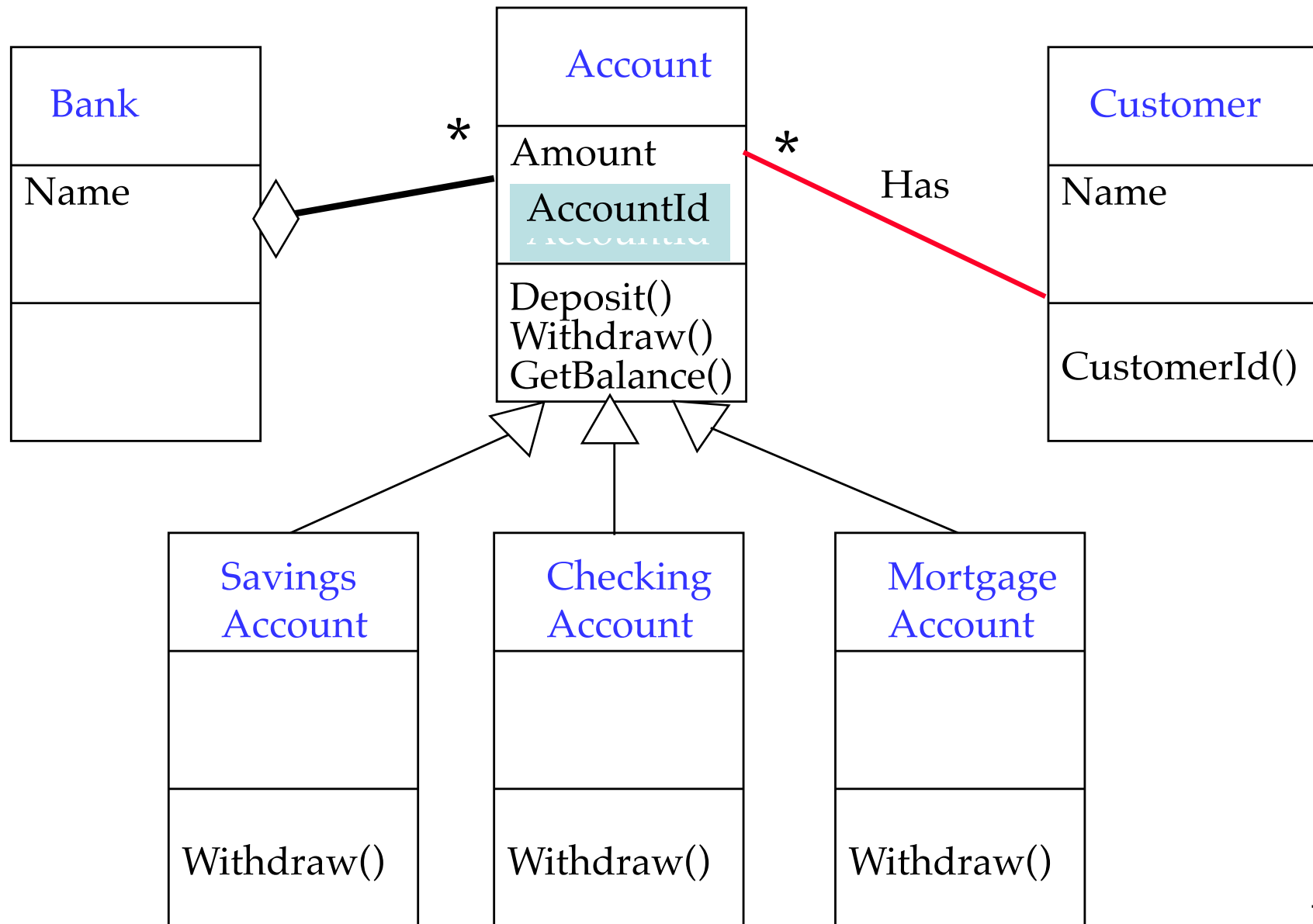
2) Iterate on Names, Attributes and Methods

3) Find Associations between Objects

4) Label the associations

5) Determine the multiplicity of the associations

# Practice Object Modeling: Iterate, Categorize!

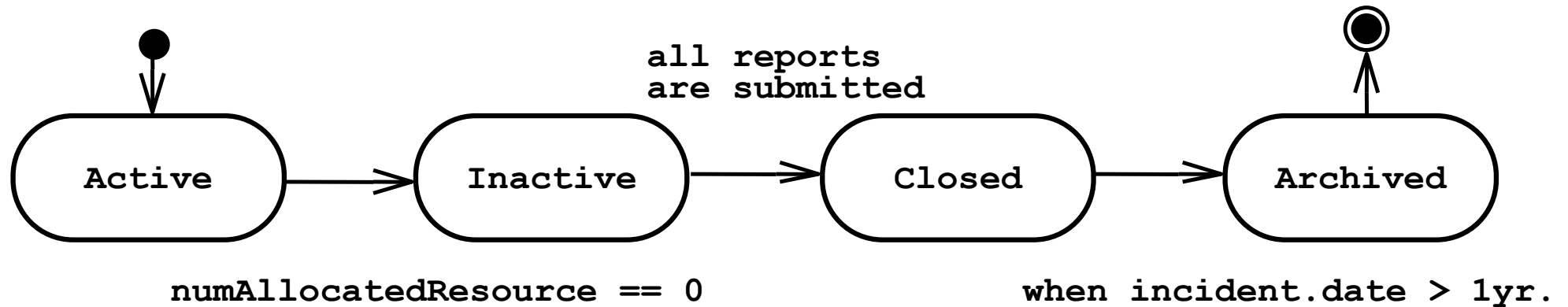


# Modellare il comportamento dei singoli oggetti

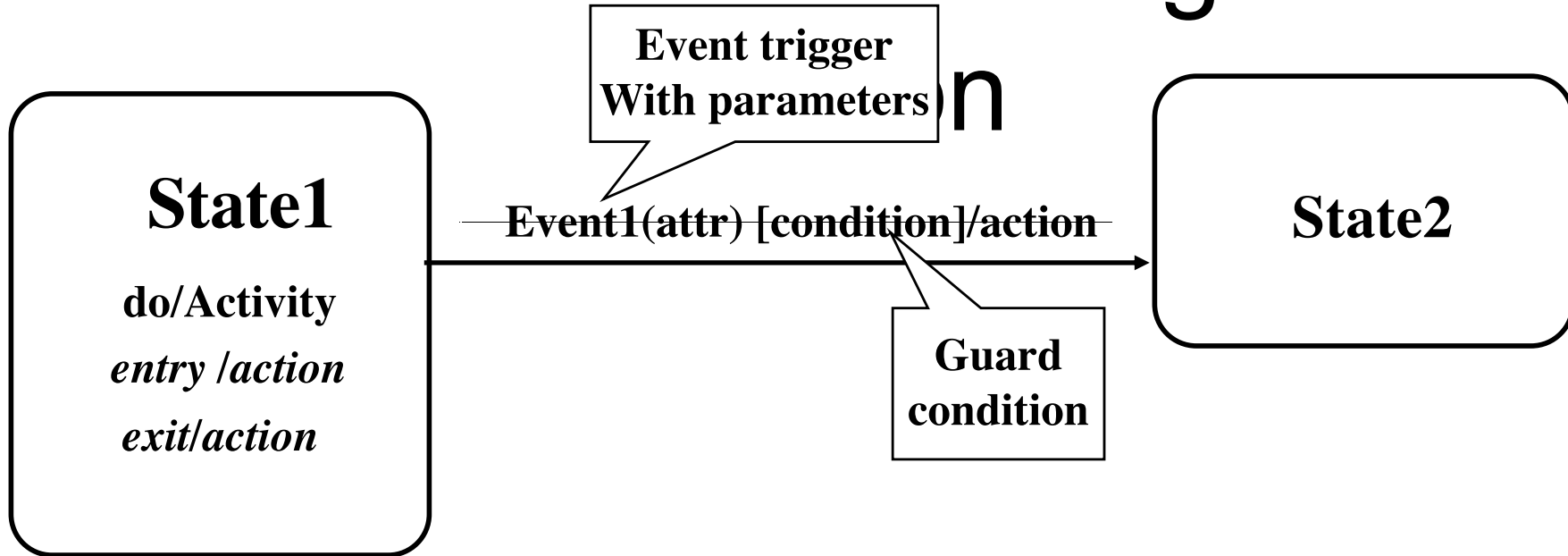
- I sequence diagram sono usati per “distribuire” il comportamento tra i diversi oggetti del sistema e rappresentano il comportamento del sistema dal punto di vista dell’ utente
- **Gli statechart rappresentano il comportamento dei singoli oggetti**
- Gli sviluppatori costruiscono una descrizione formale del comportamento degli oggetti, e di conseguenza identificano use case “tralasciati” nella prima fase
- Non è necessario costruire statechart per tutti gli oggetti del sistema (solo oggetti con una durata del ciclo di vita significativa):
  - è il caso degli oggetti control,
  - meno degli oggetti entity,
  - no per oggetti boundary.



# UML statechart for Incident.



# UML Statechart Diagram



- Notation based on work by Harel

# Rivedere il modello di analisi

- Il modello di analisi è costruito **incrementalmente e iterativamente** (molte iterazioni con il cliente e gli utenti sono necessarie)
- Quando il modello di analisi diventa **stabile** (non si hanno più cambiamenti, oppure sono minimali), il modello è riesaminato, prima dagli sviluppatori e poi insieme con il cliente e gli utenti
- L'obiettivo di questa attività di revisione è stabilire che la specifica risulta essere: corretta, completa, consistente e chiara

# Rivedere il modello di analisi

- Domande per assicurarsi la **correttezza**:
  - Il glossario è comprensibile per gli utenti?
  - Le classi astratte corrispondono a concetti ad alto livello?
  - Tutte le descrizioni concordano con le definizioni degli utenti?
  - Oggetti Entity e Boundary hanno nomi significativi?
  - Oggetti control e use case sono nominati con verbi significativi?
  - Tutti gli errori sono descritti e trattati?

# Rivedere il modello di analisi

- Domande per assicurarsi la **completezza**:
  - Per ogni oggetto: E' necessario per uno use case? In quale use case è creato? modificato? distrutto?
  - Per ogni attributo: quando è settato? Quale è il tipo?
  - Per ogni associazione: quando è attraversata? Perché ha una data molteplicità?
  - Per ogni oggetto control: ha le associazioni necessarie per accedere agli oggetti che partecipano nel corrispondente use case

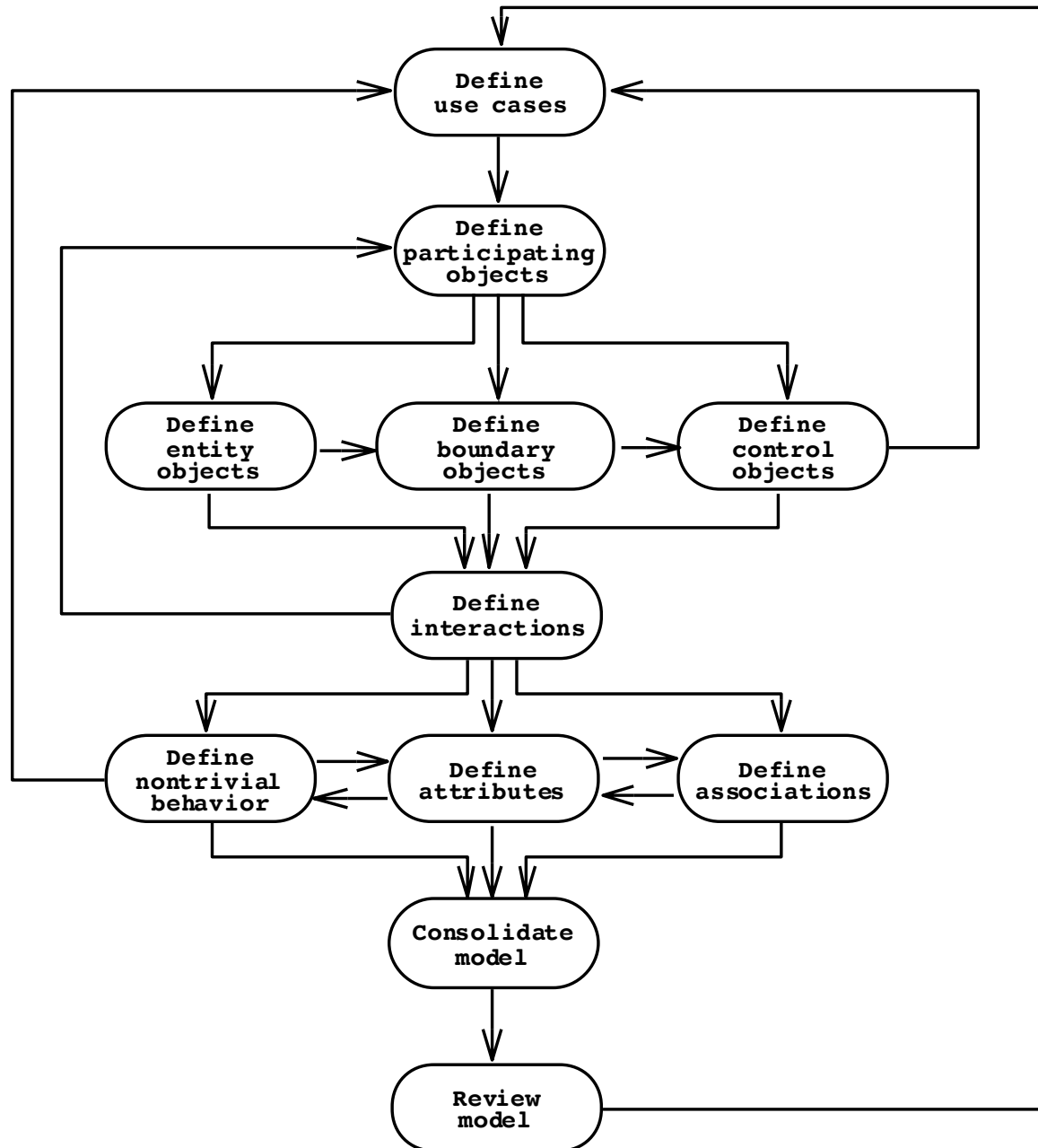
# Rivedere il modello di analisi

- Domande per assicurarsi la **consistenza**:
  - Ci sono classi o use case con lo stesso nome?
  - Ci sono entità con nomi simili e che denotano concetti simili?
  - Ci sono oggetti con attributi e associazioni simili che non sono nella stessa gerarchia di ereditarietà

# Rivedere il modello di analisi

- Domande per assicurarsi **realismo**:
  - Ci sono caratteristiche innovative nel sistema? Sono stati fatti studi o costruiti prototipi per valutarne la fattibilità?
  - Le richieste di performance e affidabilità possono essere assicurate? E' stato costruito un prototipo per assicurarsi della fattibilità? Tale prototipo è in esecuzione su qualche hardware particolare?

# Analysis: UML Activity Diagram





# Gestione dell' analisi

- Documentare l' analisi
- Assegnare responsabilità
- Comunicazione nella fase di analisi
- Iterazione nel modello di analisi
- Accordo con il cliente

# Documentare l'analisi

- Il documento di analisi dei requisiti ([RAD](#)) iniziato nella fase di raccolta dei requisiti viene completato
  - Le sezioni da 1 a 3.4.2 sono state scritte durante la raccolta dei requisiti
  - Le sezioni 3.4.3 e 3.4.4 sono scritte durante l'analisi
- Ovvero devono essere specificati
  - Object model
  - Dynamic model

# Documento di Analisi dei Requisiti (RAD)

## 1. Introduction

1.1. Purpose of the system

1.2. Scope of the system

1.3. Objectives and success criteria of the project

1.4. Definition, acronyms, and abbreviations

1.5. References

1.6. Overview

## 2. Current system

## 3. Proposed system

3.1. Overview

3.2. Functional requirements

3.3. Nonfunctional requirements

3.4. System models

3.4.1. Scenarios

3.4.2. Use case model

3.4.3. *Object model (during analysis)*

3.4.4. *Dynamic model (during analysis)*

3.4.5. User interface – navigational path and screen mock-up

## 4. Glossary

# Assegnare responsabilità

- Ci sono tre tipi di ruoli: **generazione di informazione**, **integrazione**, **revisione**
- **L'utente finale** è esperto del dominio di applicazione e **genera** informazioni sul sistema corrente
- Il **cliente**, un ruolo di **integrazione**, definisce lo scopo del sistema sulla base delle richieste dell'utente
- **L'analista** è lo sviluppatore con la maggiore conoscenza del dominio di applicazione e **genera** informazioni sul sistema da sviluppare. Ogni analista è responsabile per uno o più use case. Identifica gli oggetti, le associazioni, attributi, ...
- **L'architetto**, un ruolo di **integrazione**, unifica use case e oggetti dal punto di vista del sistema. Differenti analisti hanno differenti modi di modellare
- Il **manager delle configurazioni** è responsabile di mantenere la storia delle revisioni e la tracciabilità del RAD con gli altri documenti
- Il **reviewer** valida il RAD per correttezza, completezza, consistenza e chiarezza

# Comunicazione

- E' difficile la comunicazione nelle prime fasi. Fattori che contribuiscono ad aumentare tale difficoltà sono:
  - **Diversi background dei partecipanti**
  - **Diverse aspettative degli stakeholders**
    - **Cliente:** max investimento
    - **Utente:** supportare il proprio lavoro senza che questo incida sul suo ruolo
    - **Manager:** rispettare i tempi
  - **Nuovi team:** imparare a lavorare insieme
  - **Evoluzione del sistema:** le richieste, il linguaggio, tutto è fluttuante
- **Alcune linee guida possono aiutare a gestire la complessità**
  - Definisci un ambiente chiaro
    - Definisci bene i ruoli
    - Adotta forum pubblici e privati, con un database di discussione visibile al cliente e uno no
    - Lo sviluppatore non deve interferire con le politiche interne cliente/utente
  - Definisci chiari obiettivi e criteri di successo (sia per cliente che per lo sviluppatore) Sez. 1.3 del RAD
  - Brainstorming
    - Mettere gli stakeholder tutti insieme può aiutare a eliminare barriere di comunicazione, generando un ambiente in cui tutti collaborano alla individuazione delle soluzioni e alla loro revisione
    - Evitare di usare linguaggi ad hoc, usa uno standard come UML (use case e scenario per comunicare con il cliente e object diagram, sequence diagram e statechart per comunicare con gli altri sviluppatori)
    - L'ultima release del documento dovrebbe essere disponibile a tutti i partecipanti: mantiene una versione aggiornata on-line per propagare i cambiamenti nell'ambito del progetto

# Accordo con il cliente

- L' accordo con il cliente rappresenta l' accettazione del modello di analisi
- Cliente e sviluppatori convergono su funzioni e caratteristiche che avrà il sistema
- In più si accordano su
  - Lista di priorità
  - Processo di revisione
  - Lista di criteri per accettare o rigettare il sistema
  - Scheduling delle attività e budget

# An example of a revision process

