Object-Oriented Software Engineering

Using UML, Patterns, and Java

Outline of this Class

- What is UML?
- A more detailed view on
  - ✓Use case diagrams
  - ✓Class diagrams
  - ✓Sequence diagrams
  - ✓Activity/Statecharts diagrams

UML Basic Notation: First Summary

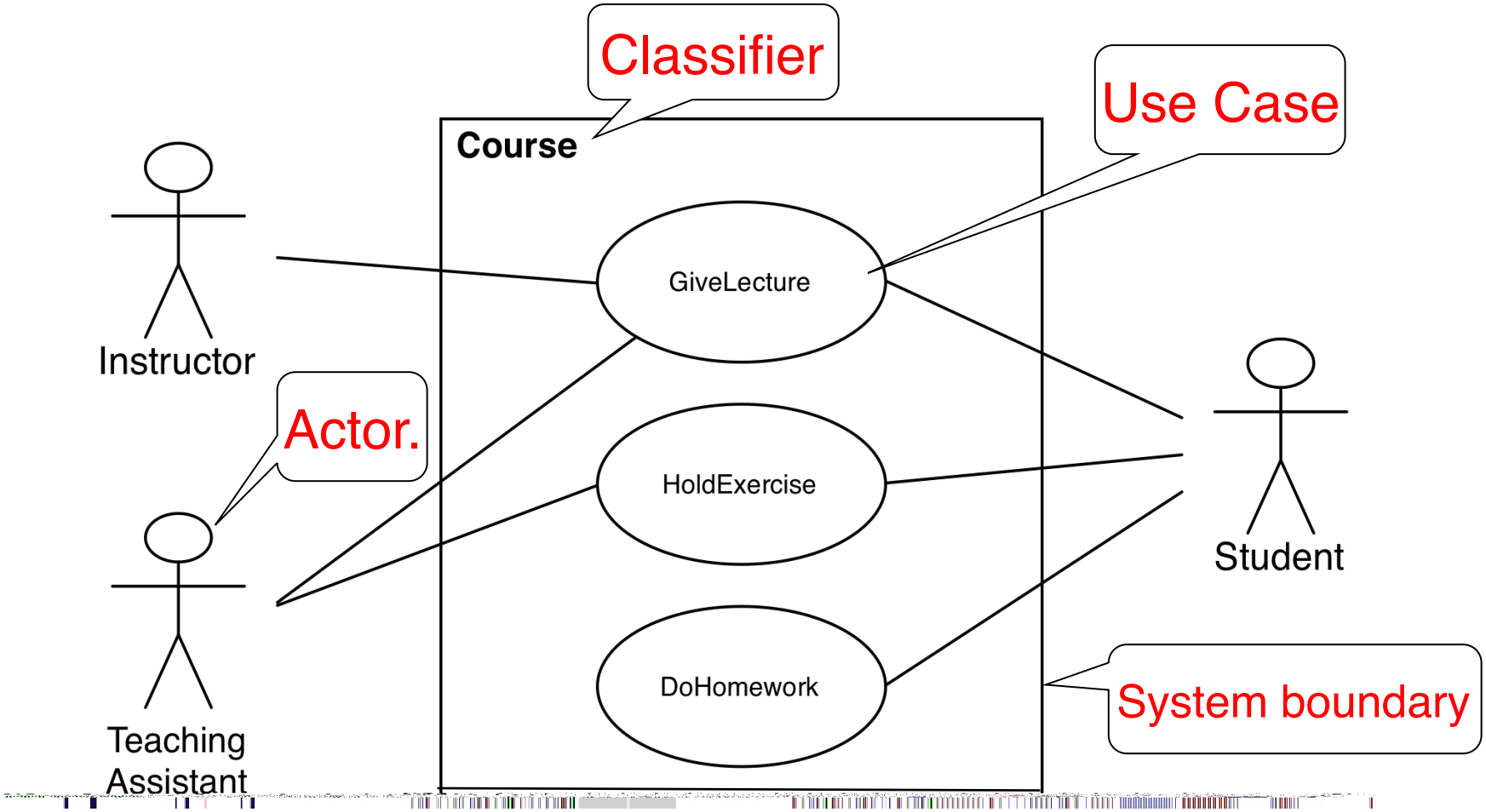
- UML provides a wide variety of notations for modeling many aspects of software systems
- UML diagrams cover the three fundamental models for software design:
  - Functional model: Use case diagrams
  - Object model: Class diagrams
  - Dynamic model: Sequence diagrams, statechart diagram
- Now we go into a little bit more detail…

UML First Pass

- Use case diagrams
  - Describe the functional behavior of the system as seen by the user
- Class diagrams
  - Describe the static structure of the system: Objects, attributes, associations
- Sequence diagrams
  - Describe the dynamic behavior between objects of the system
- Statechart diagrams
  - Describe the dynamic behavior of an individual object
- Activity diagrams
  - Describe the dynamic behavior of a system, in particular the workflow.

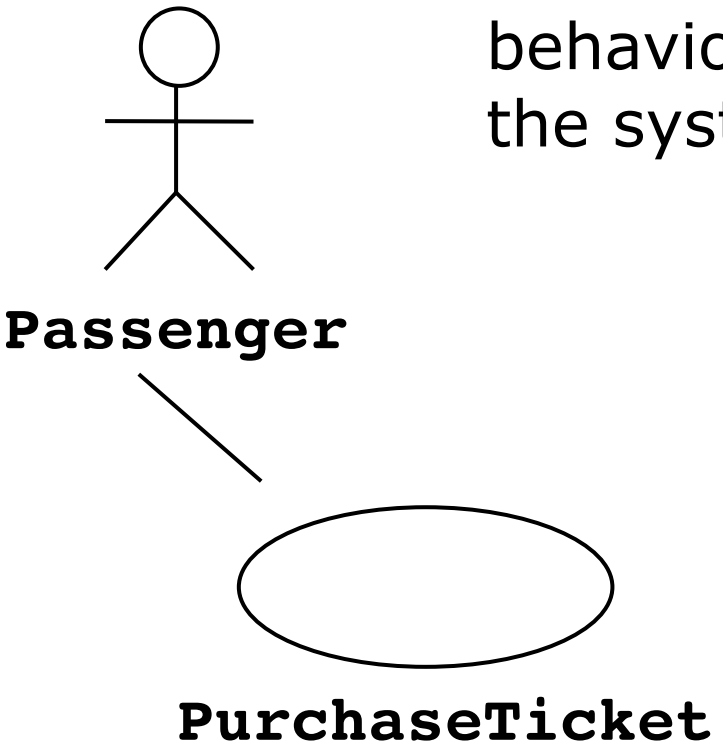
UML Use Case Diagram

UML first pass: Use case diagrams



Use case diagrams represent the functionality of the system from user's point of view

UML Use Case Diagrams



Used during requirements elicitation and analysis to represent external behavior ("visible from the outside of the system")

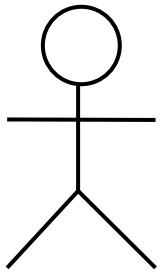
An ***Actor*** represents a role, that is, a type of user of the system

A ***use case*** represents a class of functionality provided by the system

Use case model:

The set of all use cases that completely describe the functionality of the system.

Actors



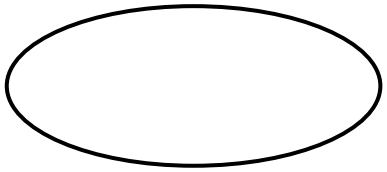
Passenger

- An actor is a model for an external entity which interacts (communicates) with the system:
  - User
  - External system (Another system)
  - Physical environment (e.g. Weather)
- An actor has a unique name and an optional description
- Examples:
  - **Passenger**: A person in the train
  - **GPS satellite**: An external system that provides the system with GPS coordinates.

Optional Description

Name

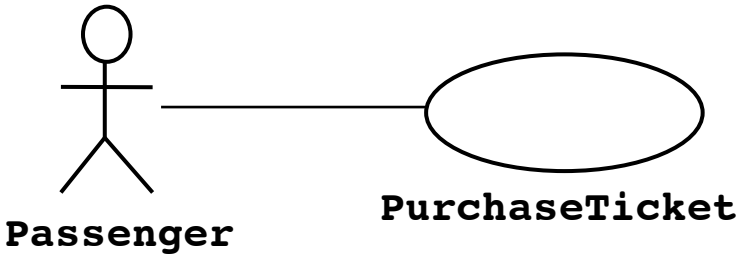
Use Case



PurchaseTicket

- A use case represents a class of functionality provided by the system
- Use cases can be described textually, with a focus on the event flow between actor and system
- The textual use case description consists of 6 parts:
  1. Unique name
  2. Participating actors
  3. Entry conditions
  4. Exit conditions
  5. Flow of events
  6. Special requirements.

Textual Use Case Description Example

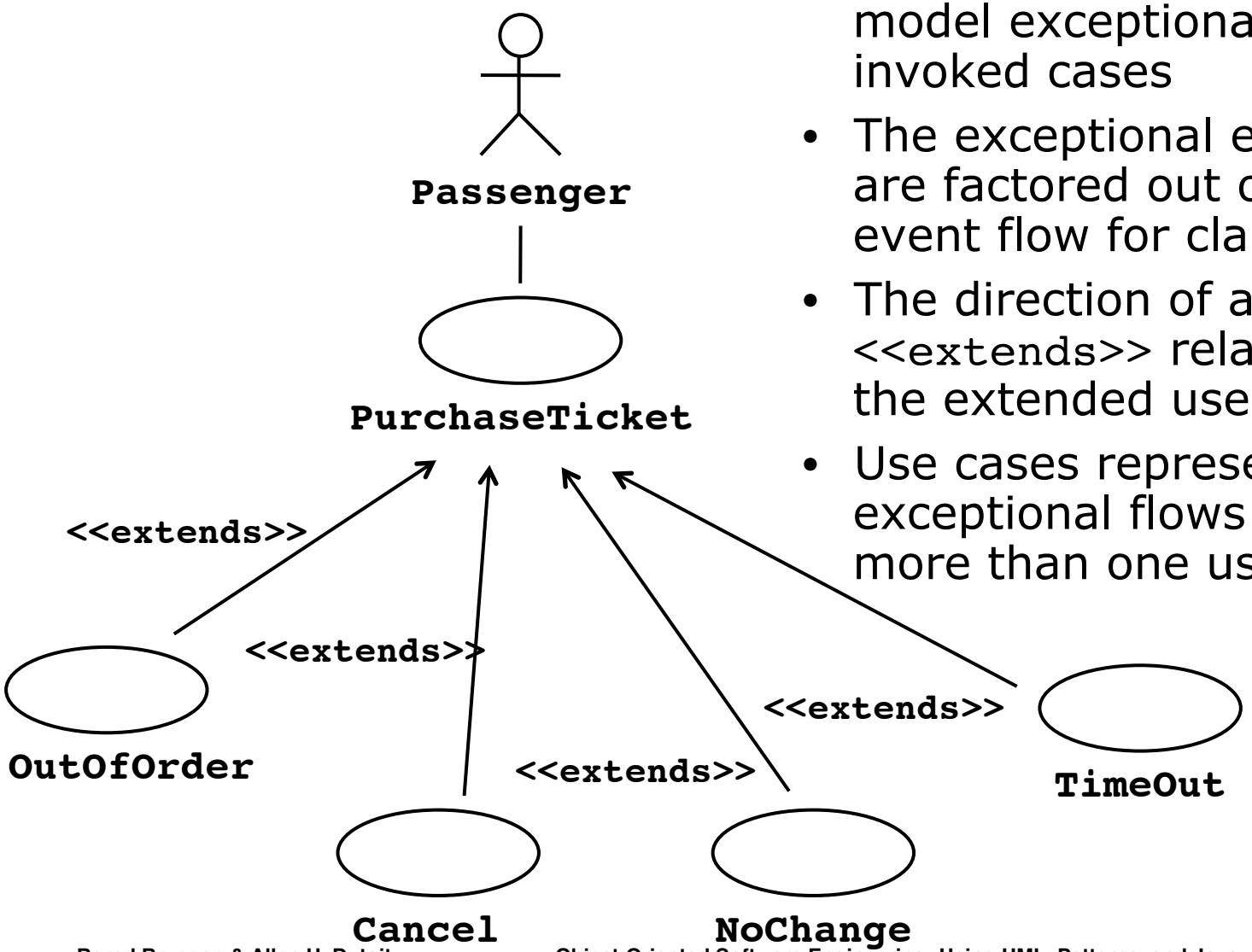


1. *Name:* `Purchase ticket`
2. *Participating actor:* `Passenger`
3. *Entry condition:*
   - (**GOOD**) `Passenger` selects an option from the display
   - (**WRONG**) `Passenger` stands in front of ticket distributor
   - (**Very WRONG**) `Passenger` has sufficient money to purchase ticket
4. *Exit condition:*
   - `Passenger` has ticket
   - (**Better**): System delivered ticket
5. *Flow of events:*
   1. `Passenger` selects the number of zones to be traveled
   2. Ticket Distributor displays the amount due
   3. `Passenger` inserts money, at least the amount due
   4. Ticket Distributor returns change
   5. Ticket Distributor issues ticket
6. *Special requirements:* *None.*

Use Cases can be related

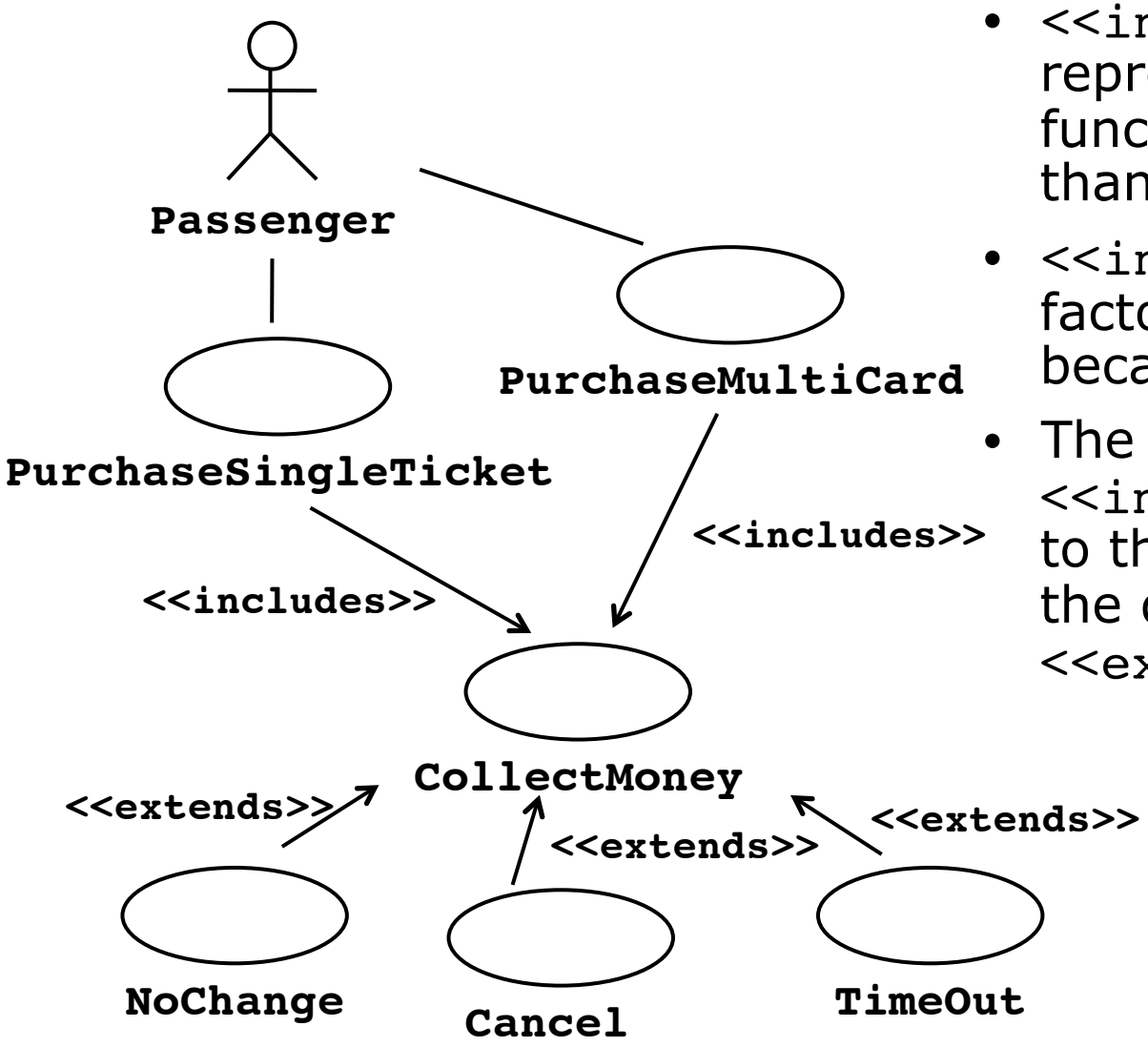
- Extends Relationship
  - To represent seldom invoked use cases or exceptional functionality
- Includes Relationship
  - To represent functional behavior common to more than one use case.

The <<*extends*>> Relationship



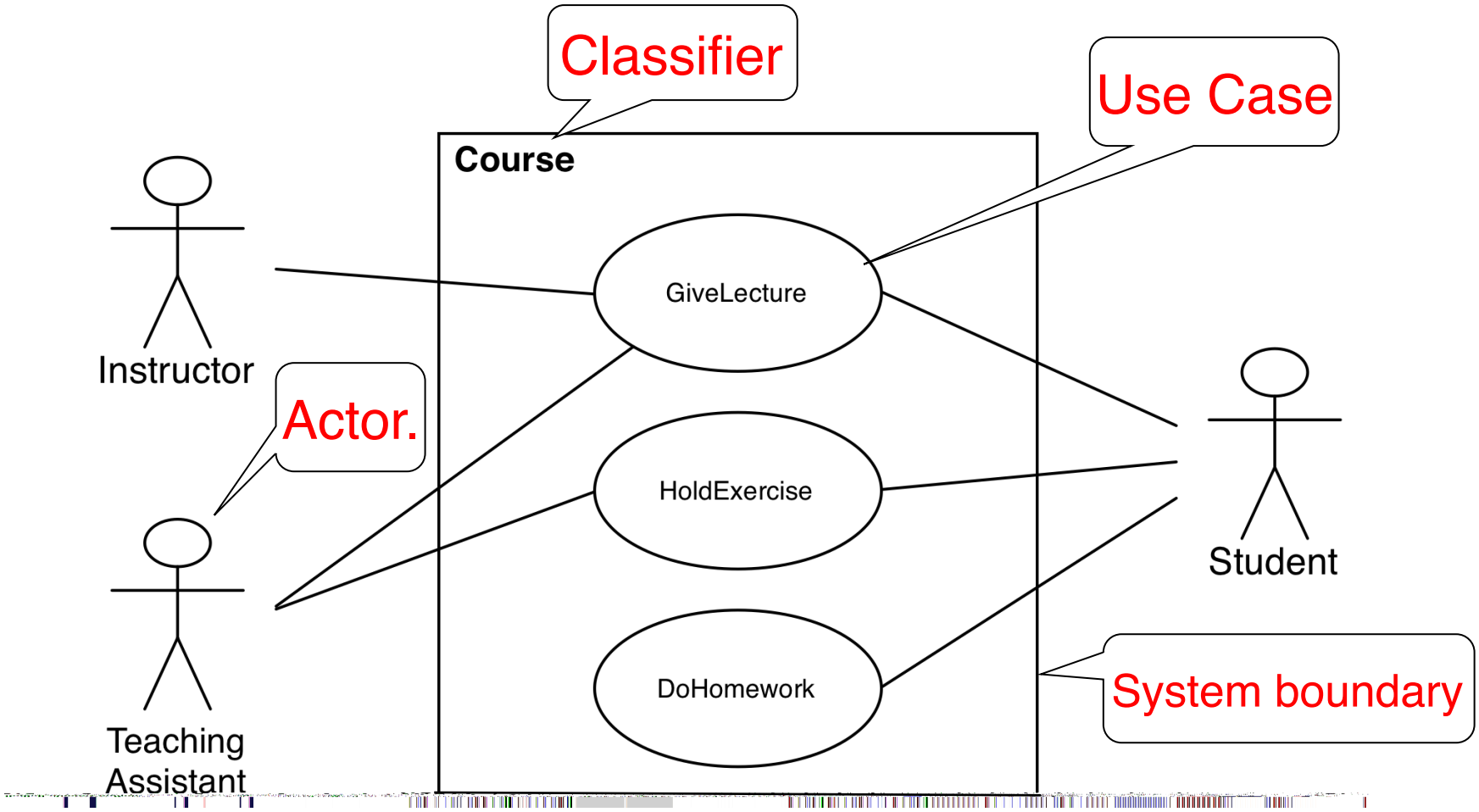
- `<<extends>>` relationships model exceptional or seldom invoked cases
- The exceptional event flows are factored out of the main event flow for clarity
- The direction of an `<<extends>>` relationship is to the extended use case
- Use cases representing exceptional flows can extend more than one use case.

The *<<includes>>* Relationship

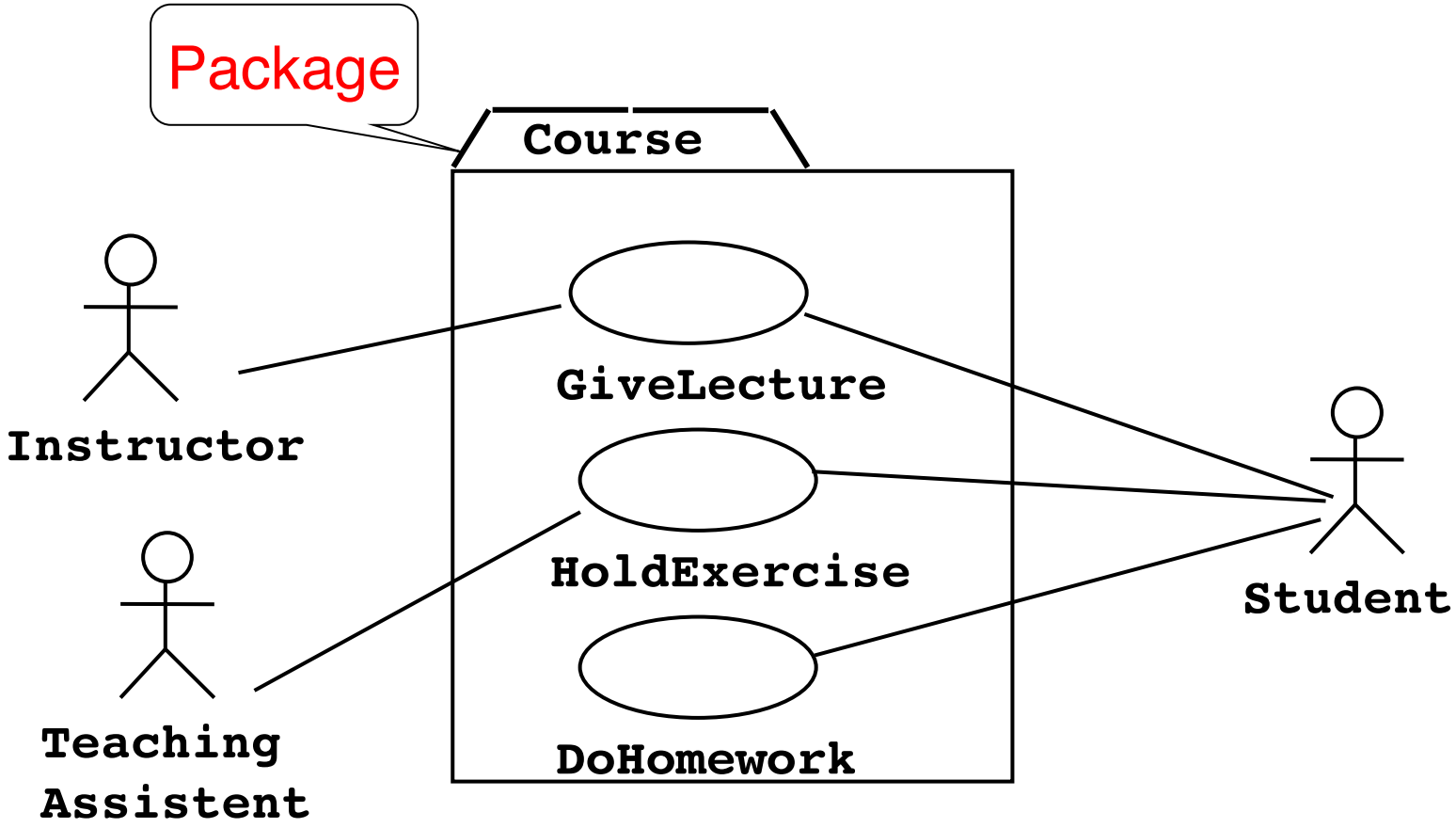


- <<includes>> relationship represents common functionality needed in more than one use case
- <<includes>> behavior is factored out for reuse, not because it is an exception
- The direction of a <<includes>> relationship is to the using use case (unlike the direction of the <<extends>> relationship).

Use Case Models can be packaged

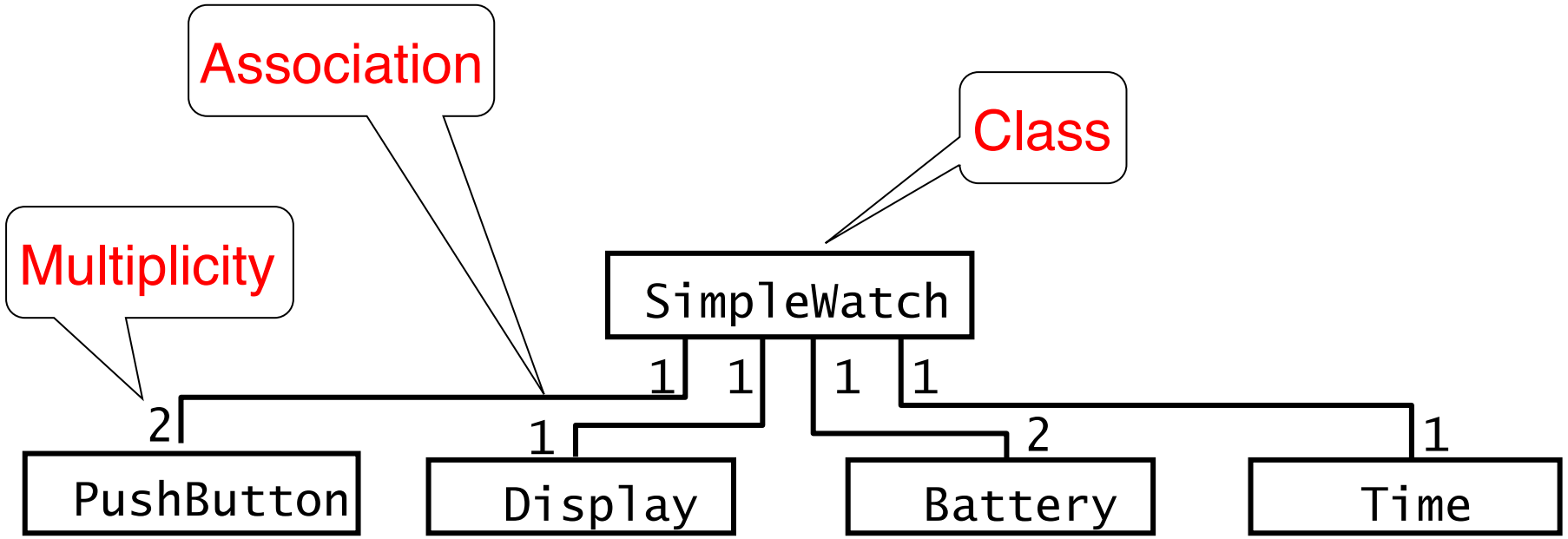


Historical Remark: UML 1 used packages



UML Class Diagram

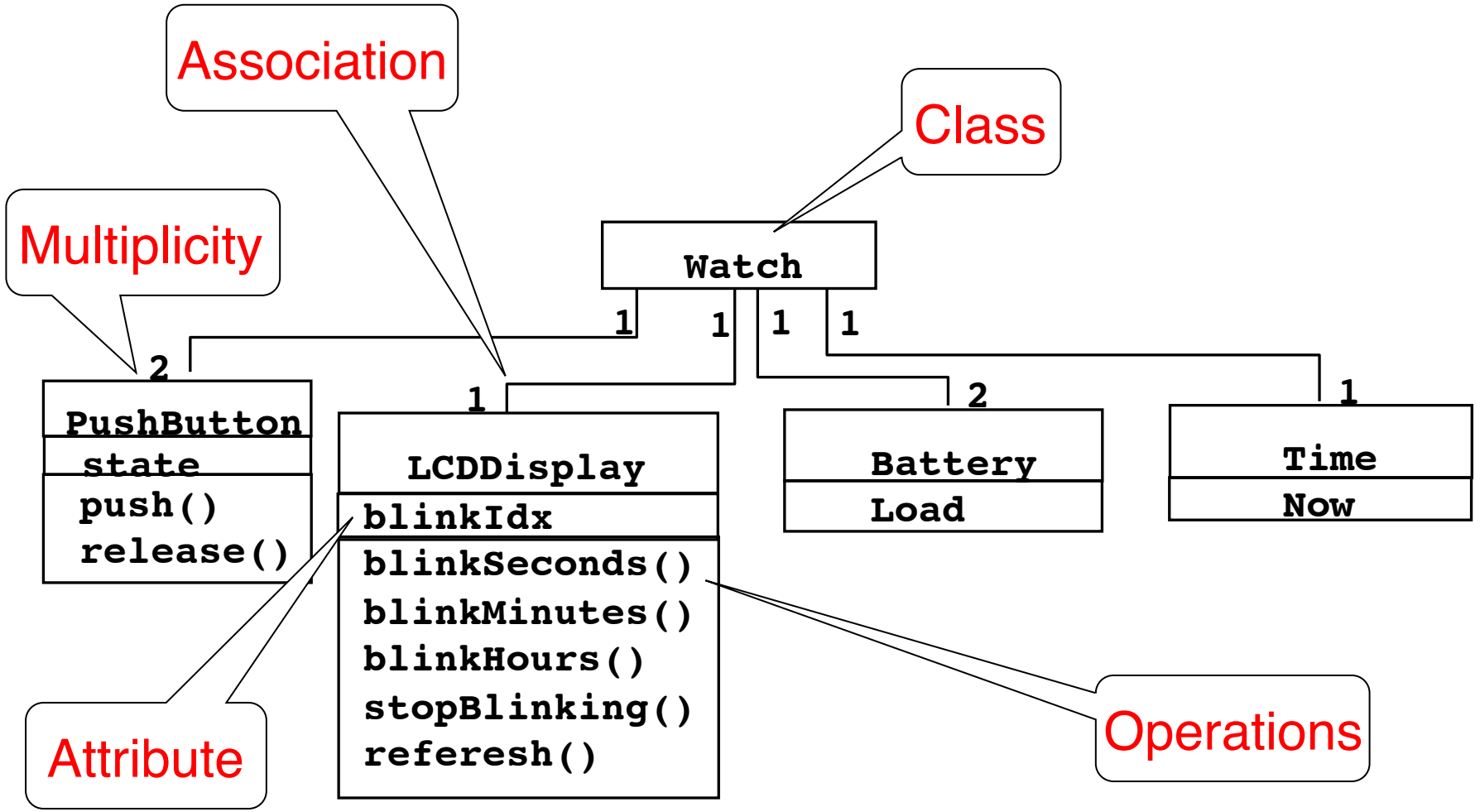
UML first pass: Class diagrams



Class diagrams represent the structure of the system

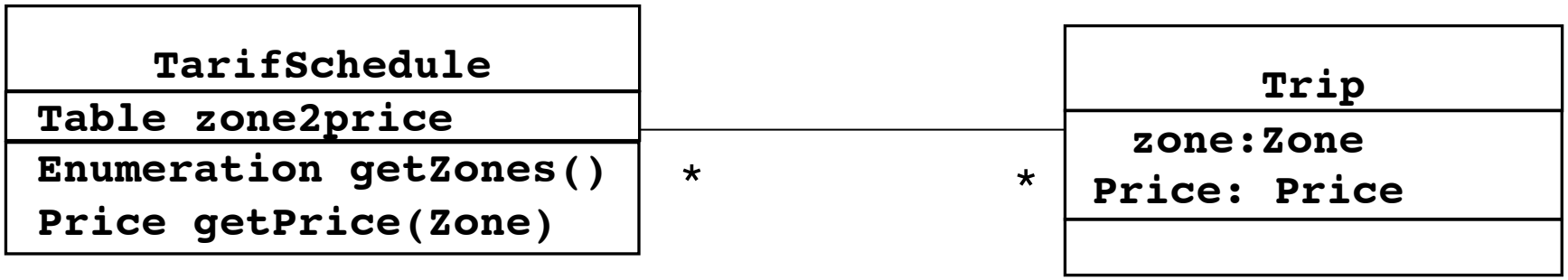
UML first pass: Class diagrams

Class diagrams represent the structure of the system

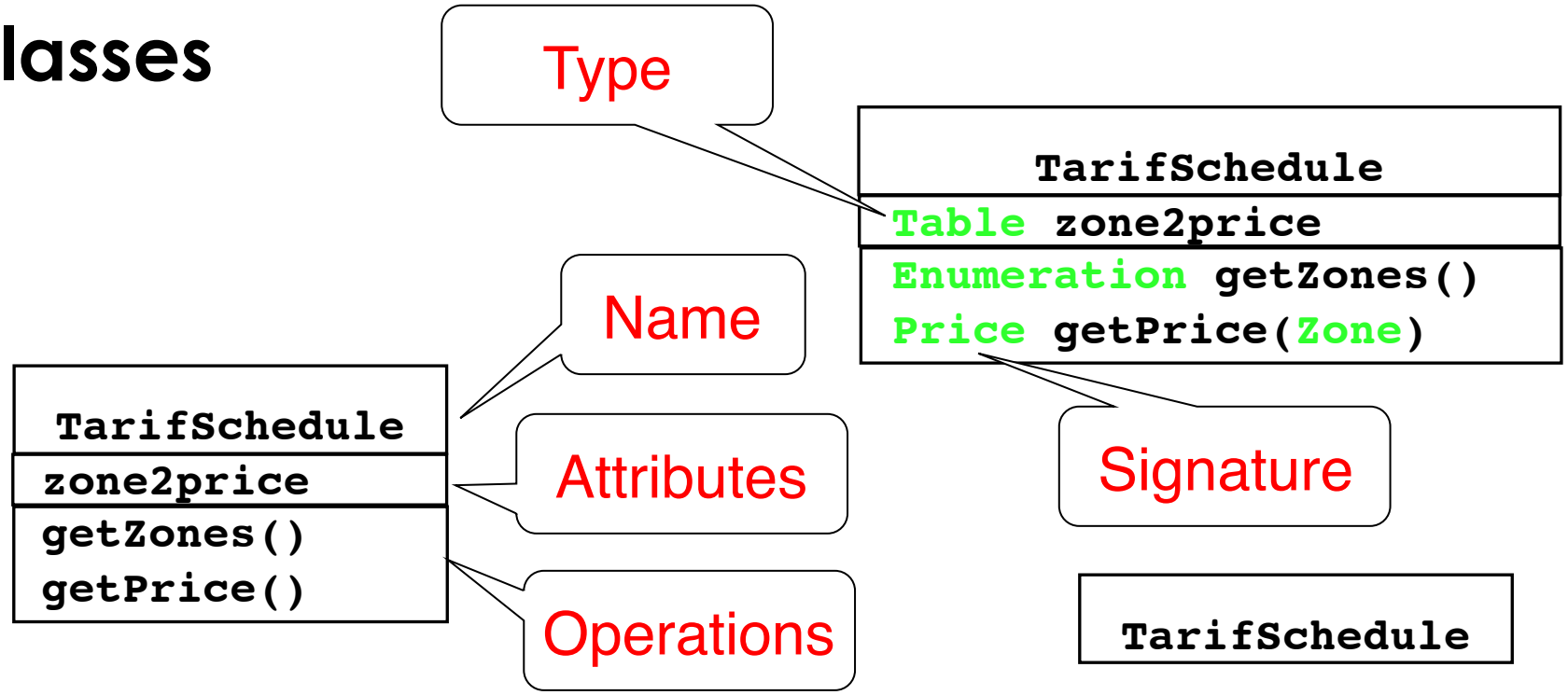


Class Diagrams

- Class diagrams represent the structure of the system
- Used
  - during requirements analysis to model application domain concepts
  - during system design to model subsystems
  - during object design to specify the detailed behavior and attributes of classes.



Classes



- A ***class*** represents a concept
- A class encapsulates state ***(attributes)*** and behavior ***(operations)***

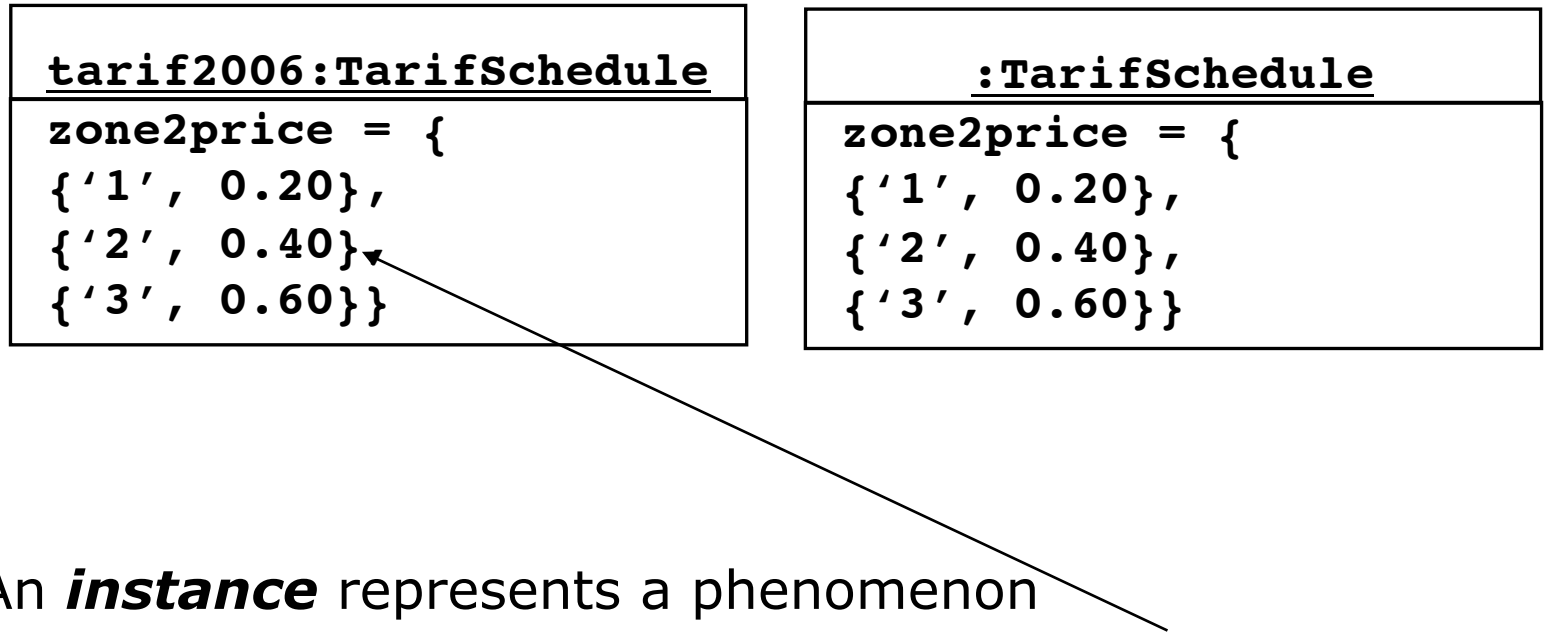
  Each attribute has a ***type***
  Each operation has a ***signature***

The class name is the only mandatory information

Actor vs Class vs Object

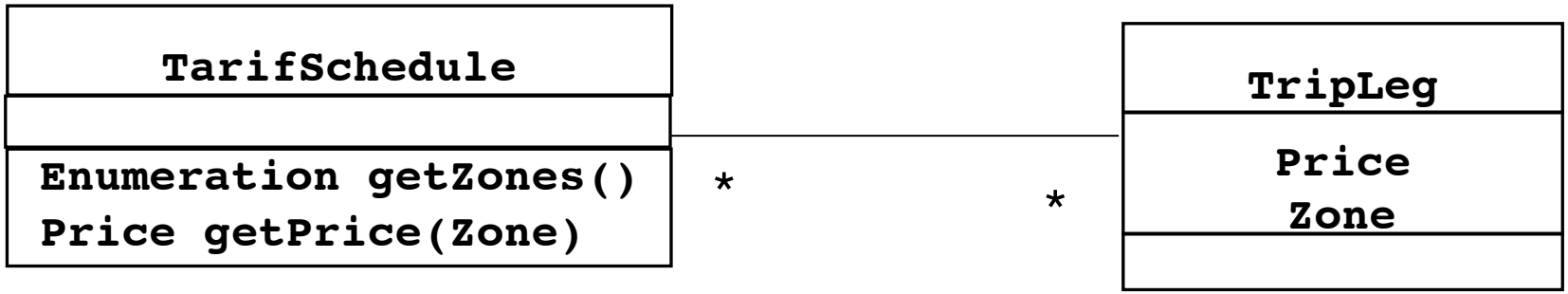
- **Actor**
  - An entity outside the system to be modeled, interacting with the system (“Passenger”)
- **Class**
  - An abstraction modeling an entity in the application or solution domain
  - The class is part of the system model (“User”, “Ticket distributor”, “Server”)
- **Object**
  - A specific instance of a class (“Joe, the passenger who is purchasing a ticket from the ticket distributor”).

Instances



- An ***instance*** represents a phenomenon
- The attributes are represented with their ***values***
- The name of an instance is underlined
- The name can contain only the class name of the instance (anonymous instance)

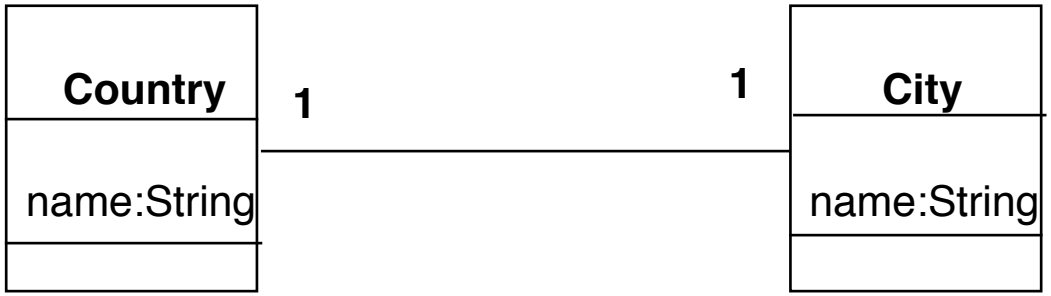
Associations



Associations denote relationships between classes

The multiplicity of an association end denotes how many objects the instance of a class can legitimately reference.

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



1-to-1 association

1-to-many association

Many-to-many Associations

- A stock exchange lists many companies.
- Each company is identified by a ticker symbol

From Problem Statement To Object Model

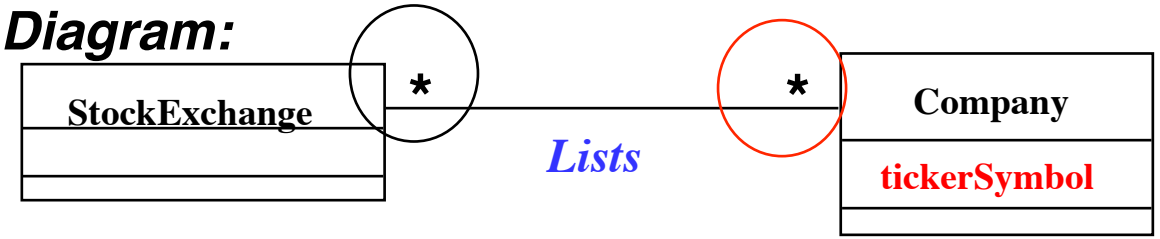
Problem Statement: A stock exchange lists many companies.
Each company is uniquely identified by a ticker symbol

Class Diagram:

From Problem Statement to Code

Problem Statement : A stock exchange lists many companies. Each company is identified by a ticker symbol

Class Diagram:



Java Code

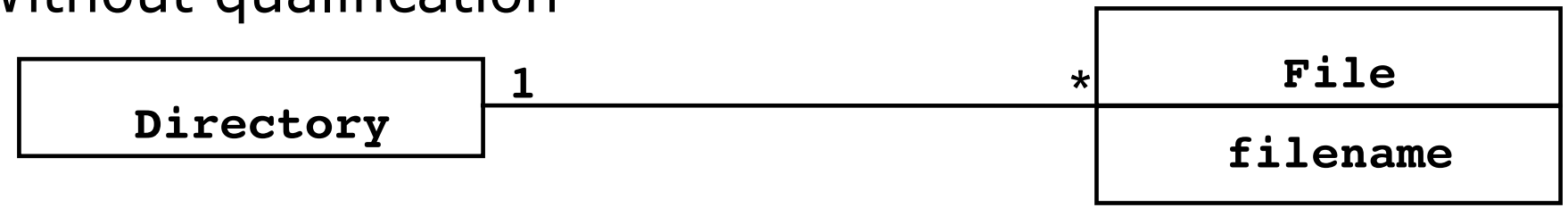
```
public class StockExchange
{
 private Vector m_Company = new Vector();
};

public class Company
{
 public int m_tickerSymbol;
 private Vector m_StockExchange = new Vector();
};
```

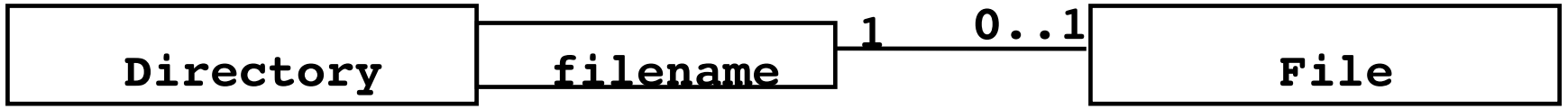
Associations are mapped to Attributes!

Qualifiers

Without qualification

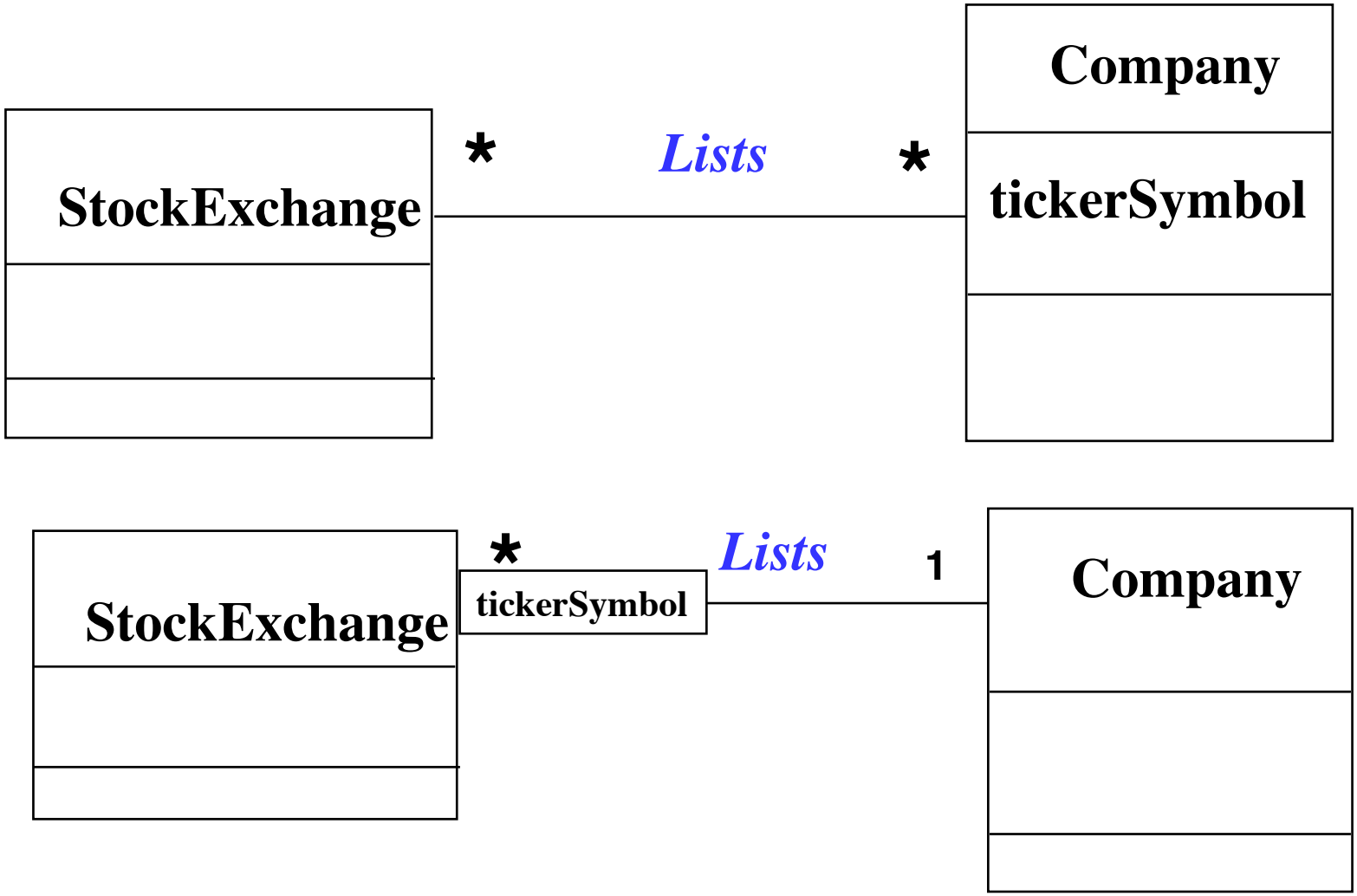


With qualification



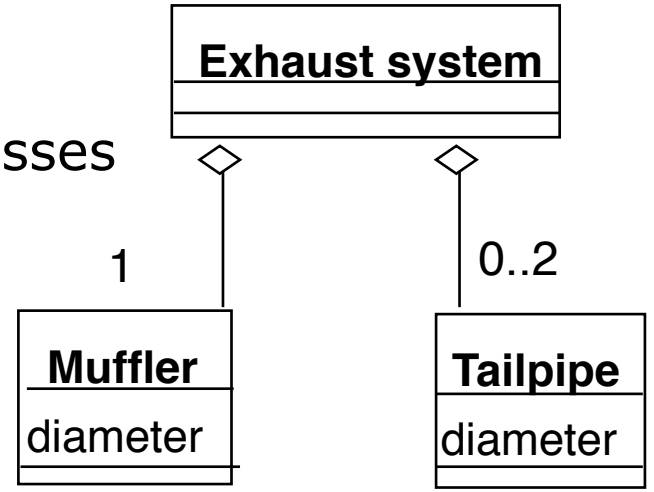
- Qualifiers can be used to reduce the multiplicity of an association

Qualification: Another Example

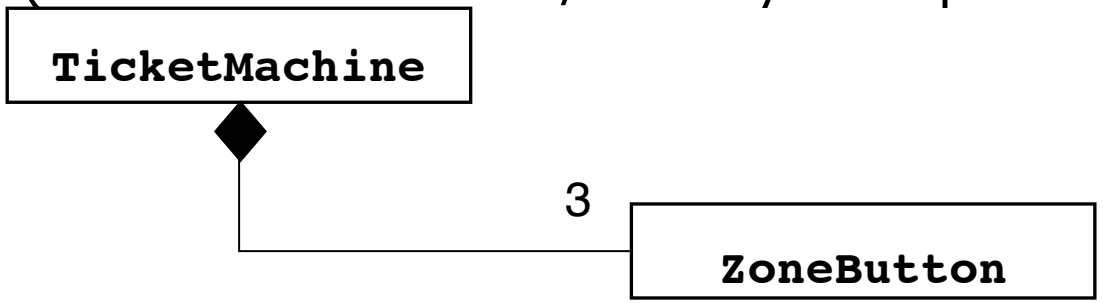


Aggregation

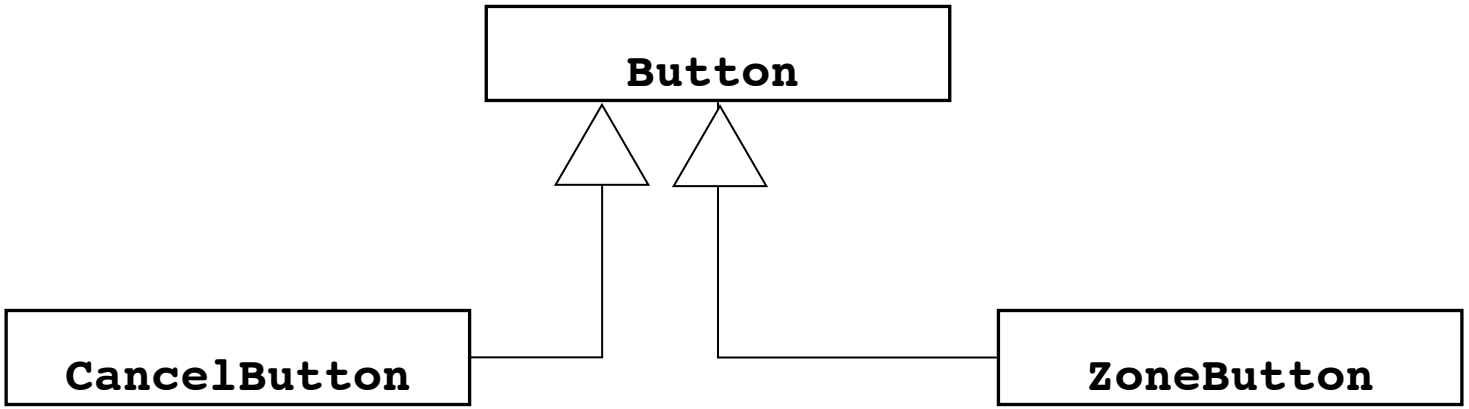
- An *aggregation* is a special case of association denoting a "consists-of" hierarchy
- The *aggregate* is the parent class, the components are the children classes



A solid diamond denotes *composition*: A strong form of aggregation where the *life time of the component instances* is controlled by the aggregate. That is, the parts don't exist on their won ("the whole controls/destroys the parts")

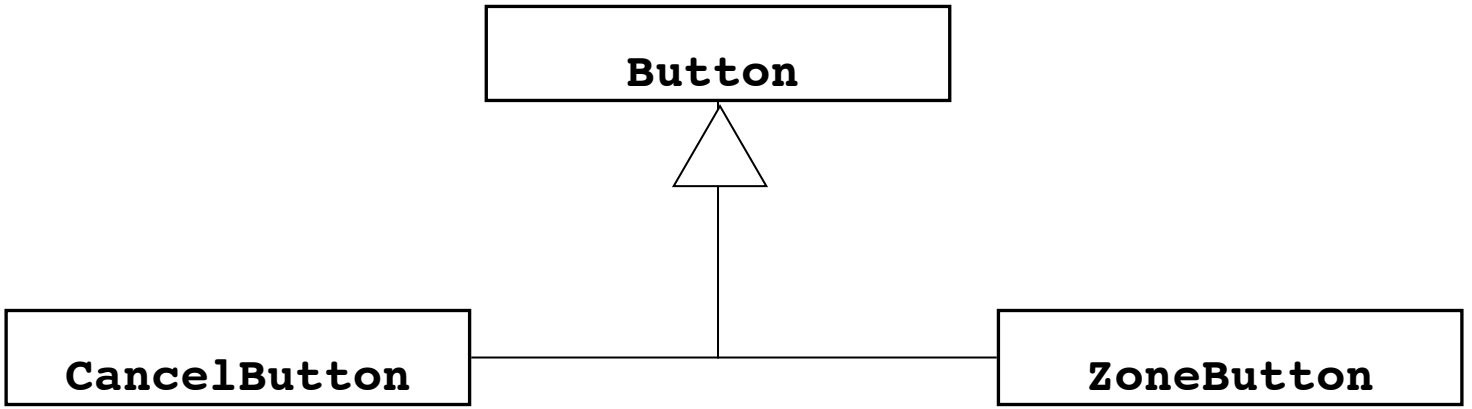


Inheritance



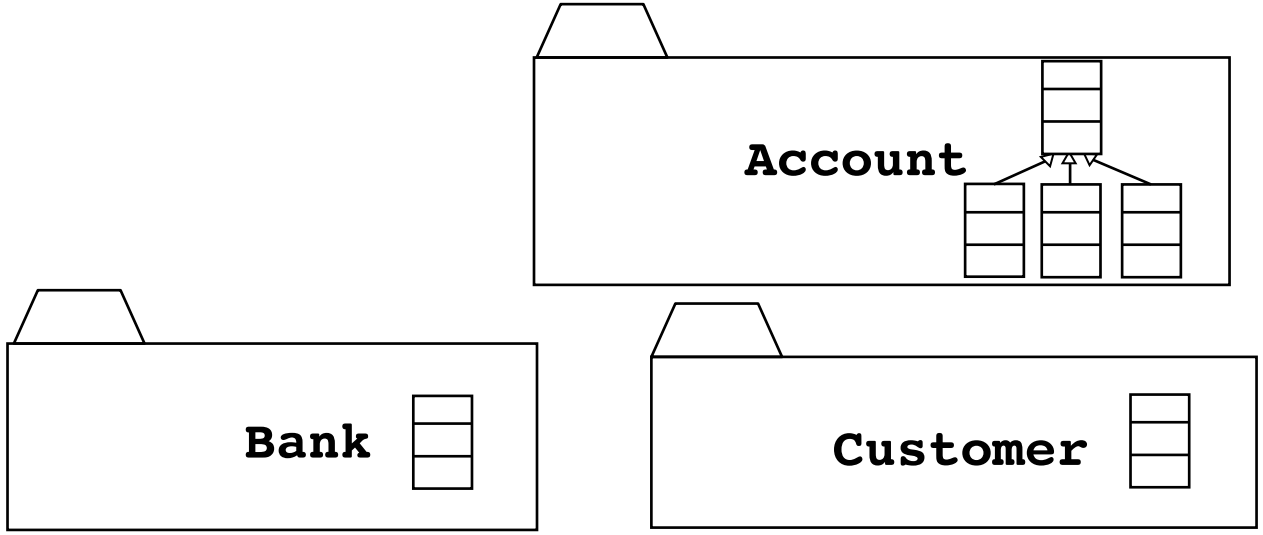
- *Inheritance* is another special case of an association denoting a "kind-of" hierarchy
- Inheritance simplifies the analysis model by introducing a taxonomy
- The **children classes** inherit the attributes and operations of the **parent class.**

Inheritance



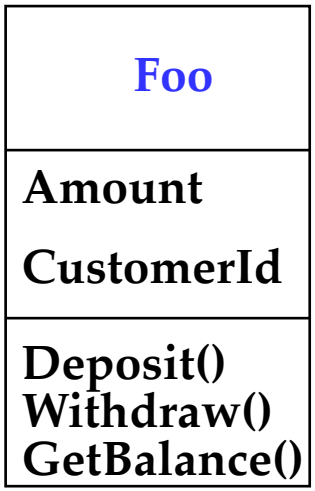
Packages

- Packages help you to organize UML models to increase their readability
- We can use the UML package mechanism to organize classes into subsystems



- Any complex system can be decomposed into subsystems, where each subsystem is modeled as a package.

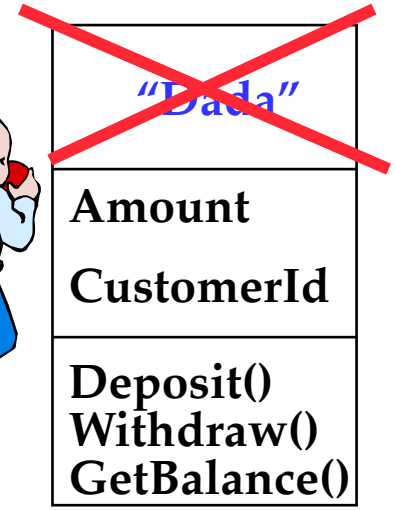
Object Modeling in Practice

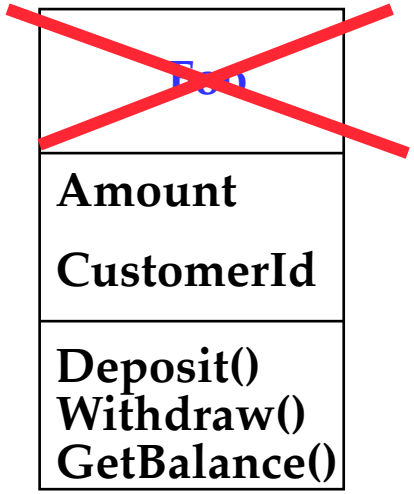


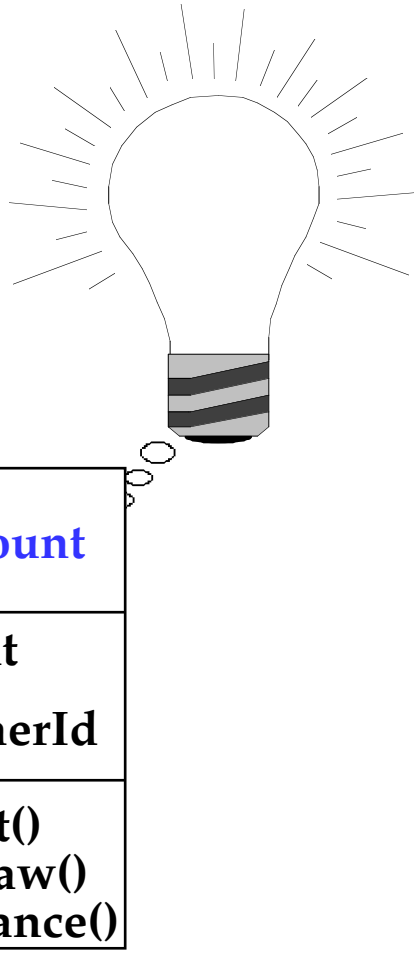
Class Identification: Name of Class, Attributes and Methods

Is Foo the right name?

Object Modeling in Practice: Brainstorming

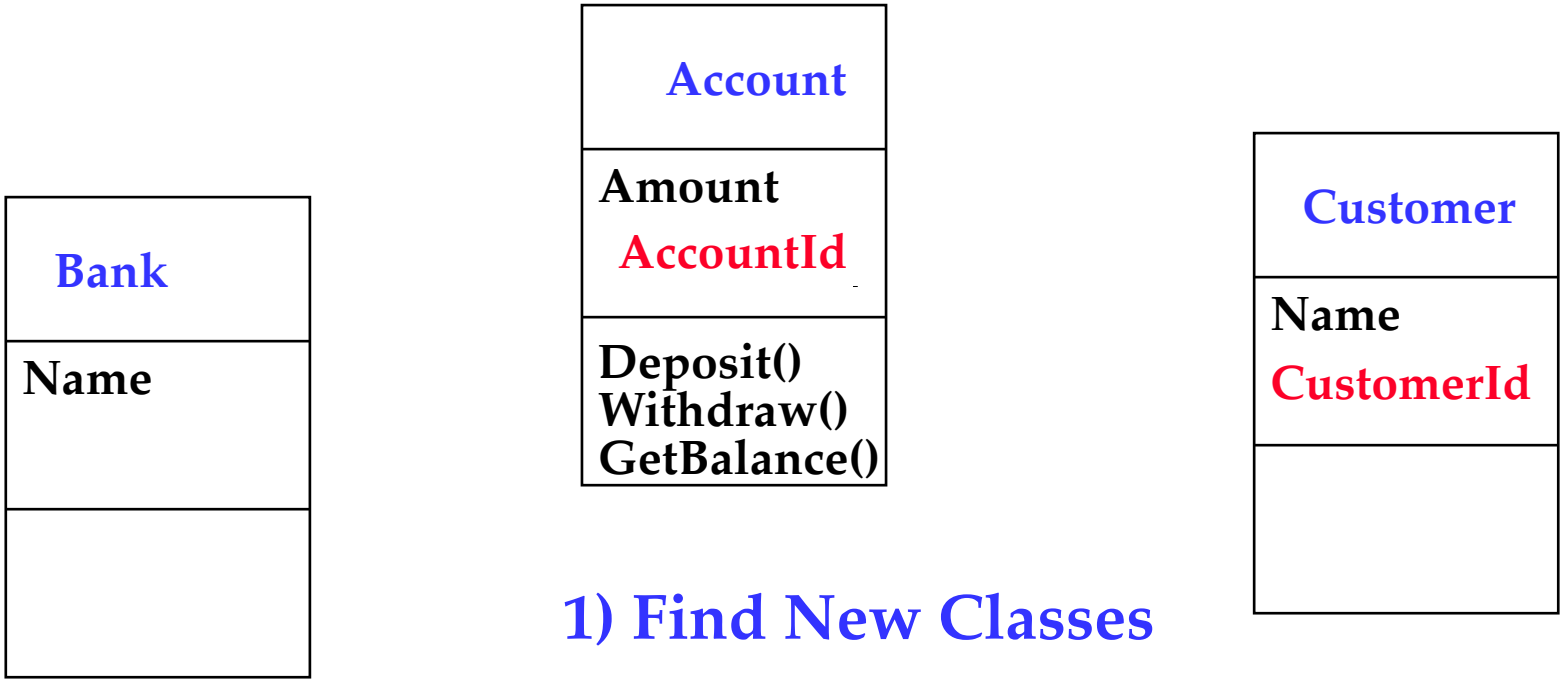






Is Foo the right name?

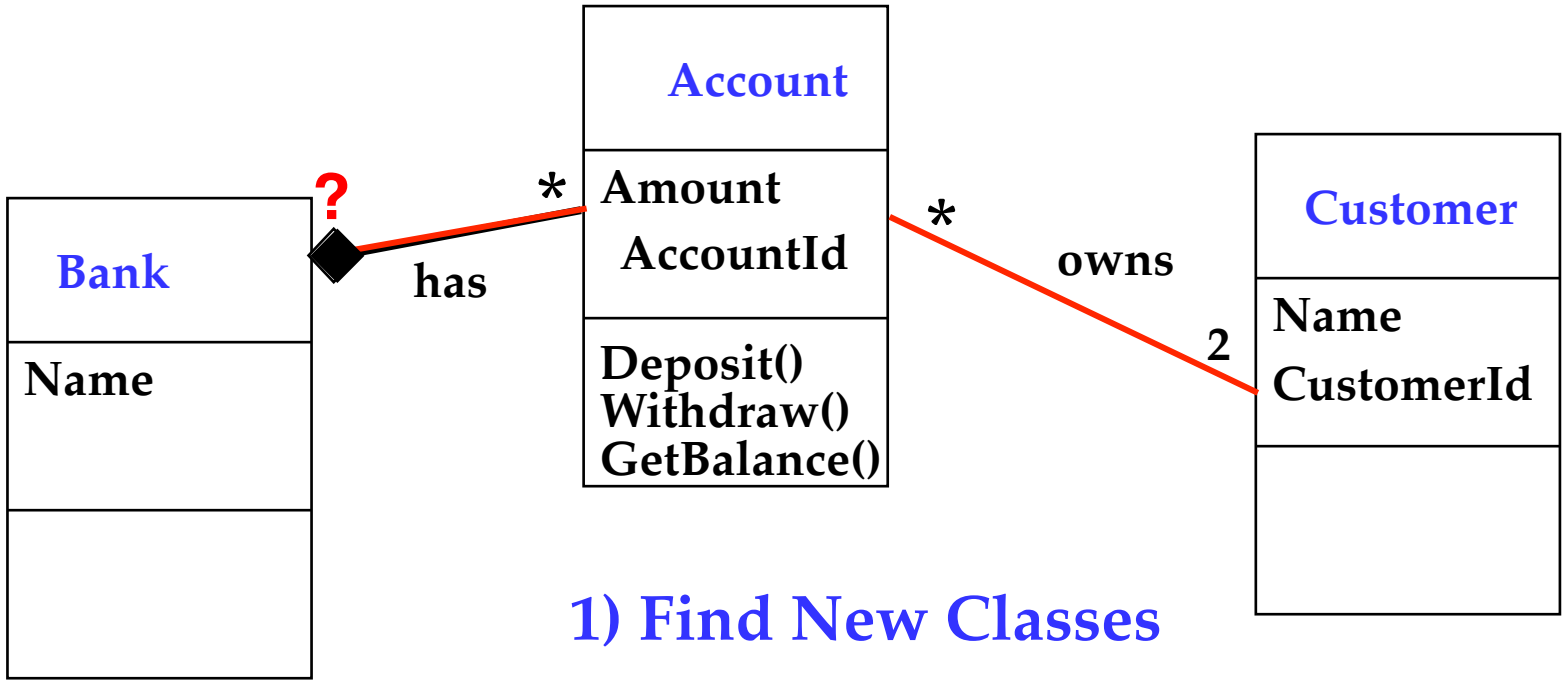
Object Modeling in Practice: More classes



1) Find New Classes

2) Review Names, Attributes and Methods

Object Modeling in Practice: Associations



1) Find New Classes

2) Review Names, Attributes and Methods

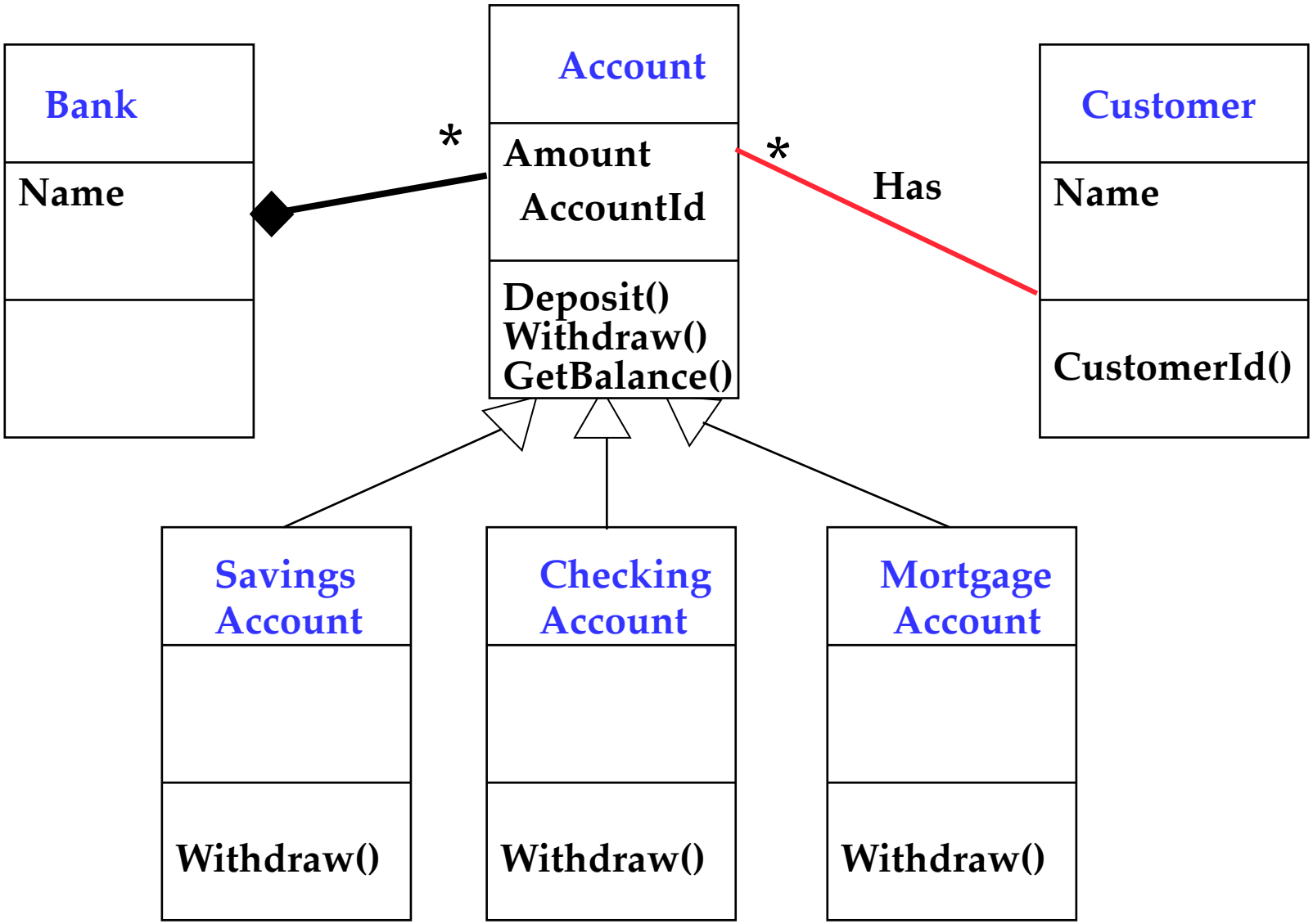
3) Find Associations between Classes

4) Label the generic assocations

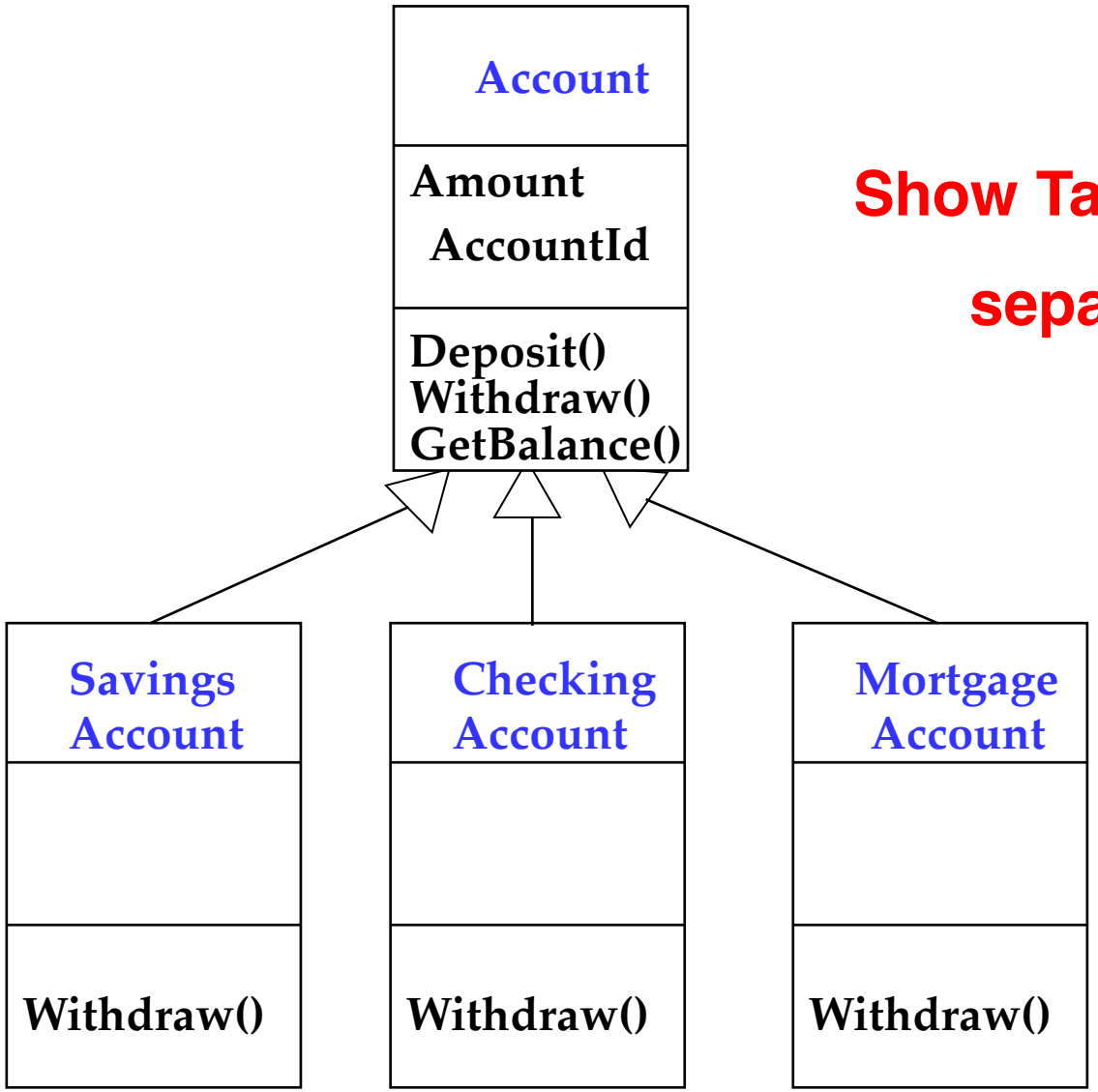
5) Determine the multiplicity of the assocations

6) Review associations

Practice Object Modeling: Find Taxonomies

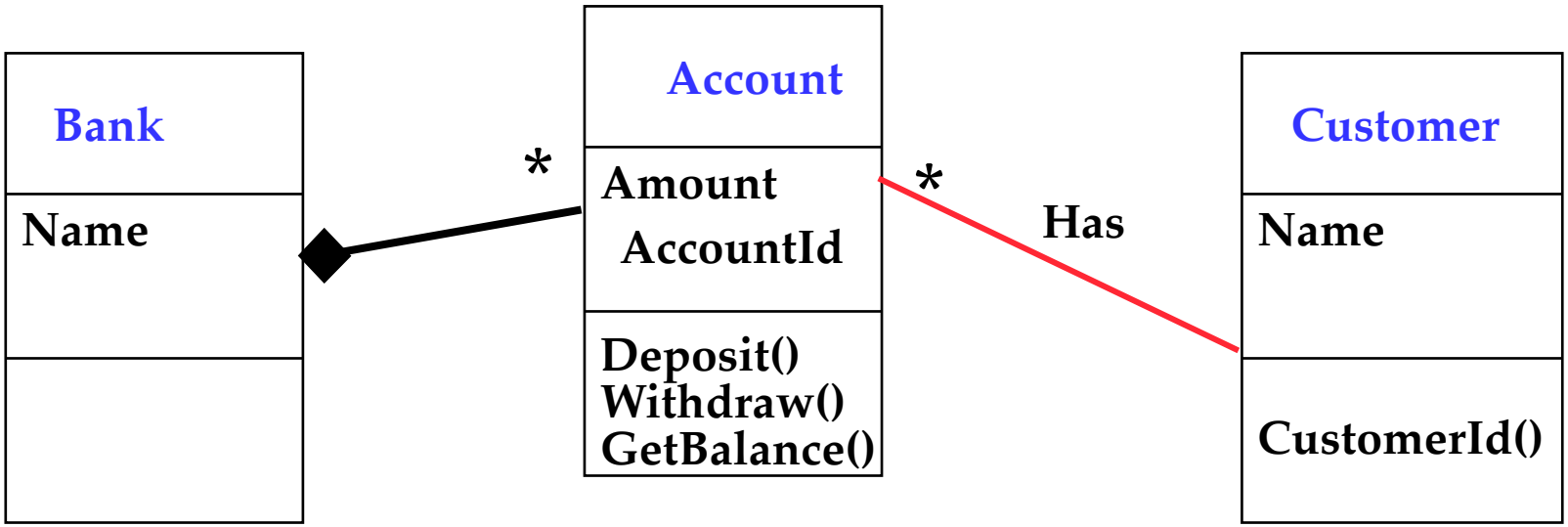


Practice Object Modeling: Simplify, Organize



Show Taxonomies separately

Practice Object Modeling: Simplify, Organize

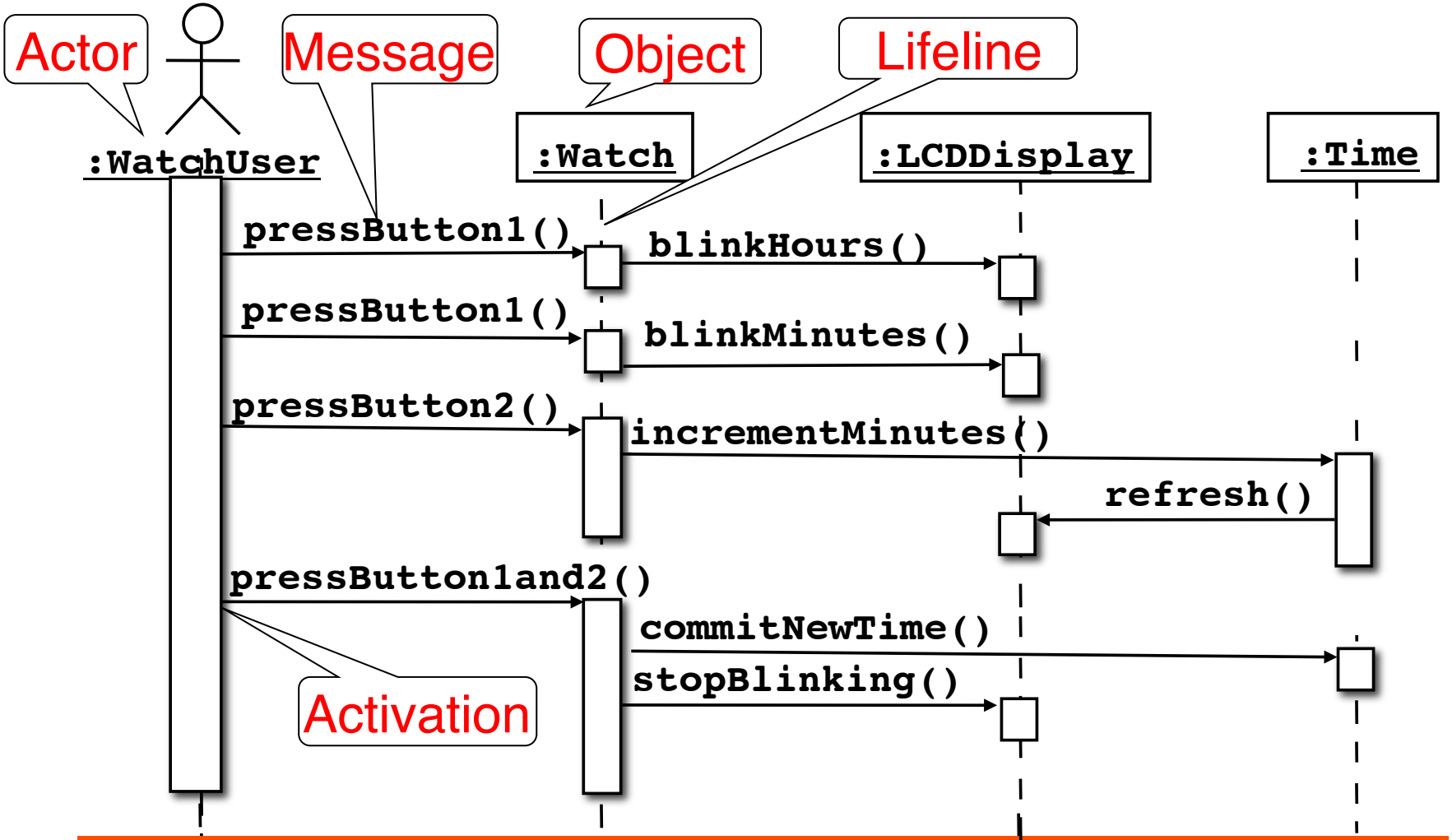


Use the 7+-2 heuristics

or better 5+-2!

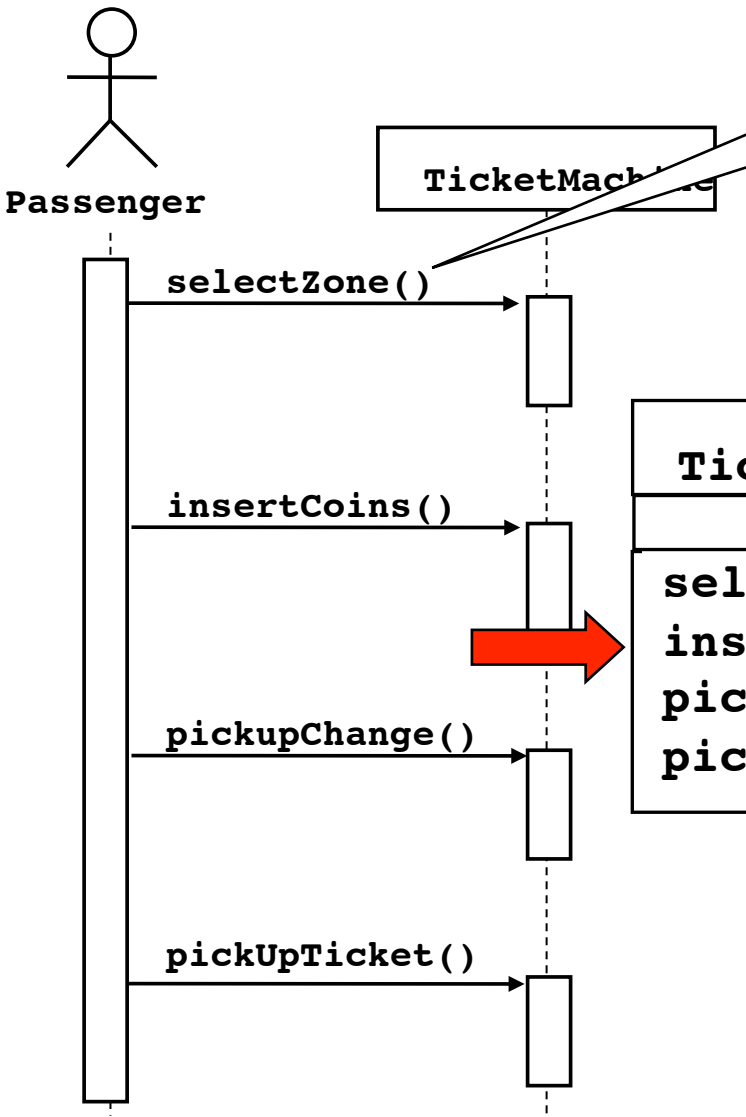
UML Sequence Diagram

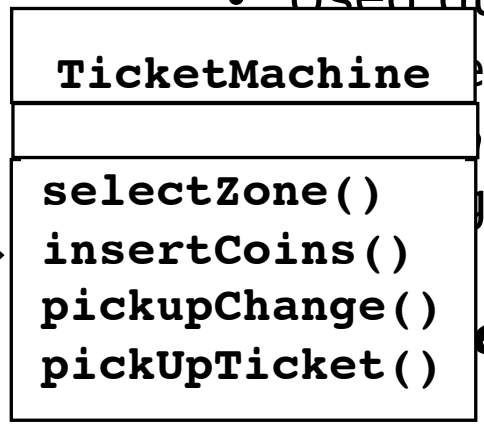
UML first pass: Sequence diagram



Sequence diagrams represent the behavior of a system as messages ("interactions") between *different objects*

Sequence Diagrams





Focus on Controlflow

Used during analysis

- To refine use case descriptions
- to find additional objects ("participating objects")

- Used during system design
- ...efine ...bsystem interfaces
- ...***ces*** ar...
- ...les. ***Act...***

Messages -> Operations on participating Object

- ***...es*** are represented by ... lines
- ***Messages*** are represented by arrows
- ***Activations*** are represented by narrow rectangles.

Scenarios, use case and sequence diagrams

- A scenario is an instance of a use case describing a concrete set of actions (*no alternative paths are in it*)
- A use case is an abstraction that describes all possible scenarios involving the described functionality.
- Scenarios are used as examples for illustrating common cases;
  - their focus is on understandability.
- Use cases are used to describe all possible cases;
  - their focus is on completeness.

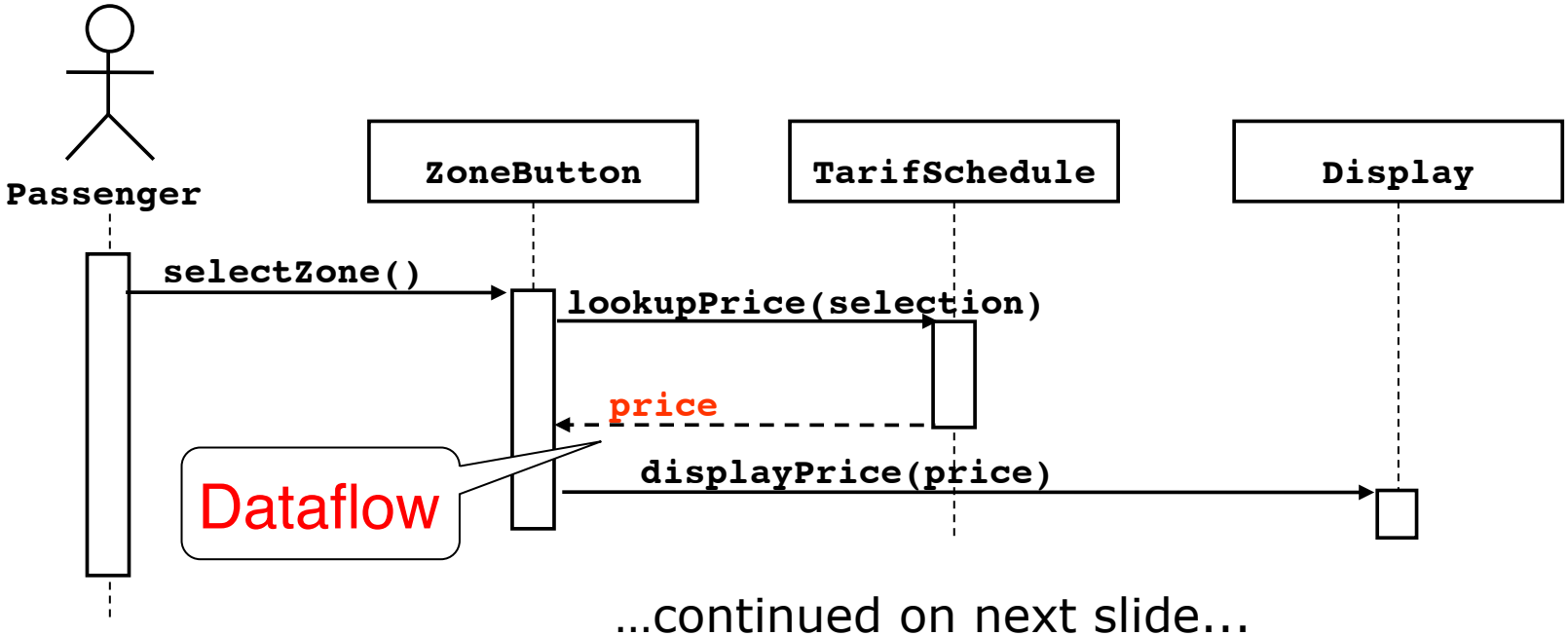
How to describe scenarios

- We describe a scenario using a template with three fields:
  - The **name** of the scenario enables us to refer to it unambiguously. The name of a scenario is underlined to indicate that it is an instance.
  - The **participating actor instances** field indicates which actor instances are involved in this scenario. Actor instances also have underlined names.
  - The **flow of events** of a scenario describes the sequence of events step by step.

Scenario: an example

Scenario name	`warehouseOnFire`
Participating actor instances	`bob, alice:FieldOfficer` `john:Dispatcher`
Flow of events	1. Bob, driving down main street in his patrol car, notices smoke coming out of a warehouse. His partner, Alice, activates the "Report Emergency" function from her `FRIEND` laptop. 2. Alice enters the address of the building, a brief description of its location (i.e., northwest corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene given that area appears to be relatively busy. She confirms her input and waits for an acknowledgment. 3. John, the `Dispatcher`, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the `Incident` site and sends their estimated arrival time (ETA) to Alice. 4. Alice receives the acknowledgment and the ETA.

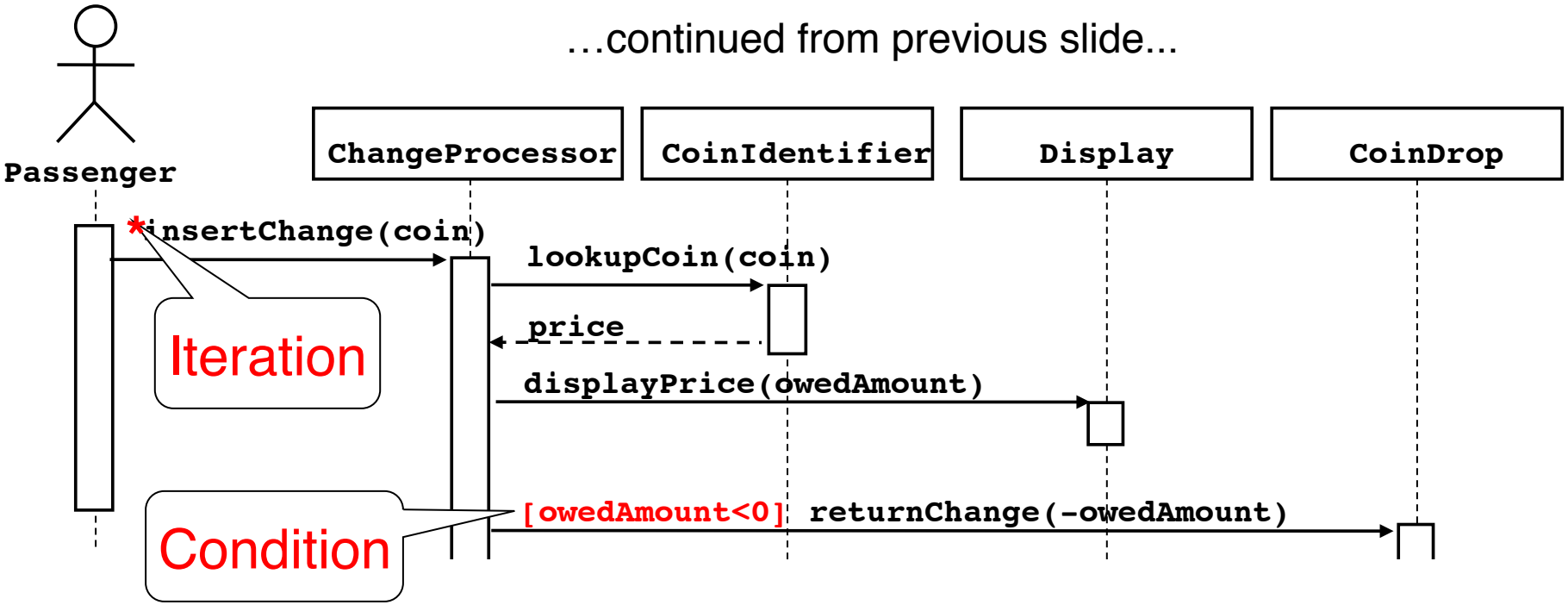
Sequence Diagrams can also model the Flow of Data



...continued on next slide...

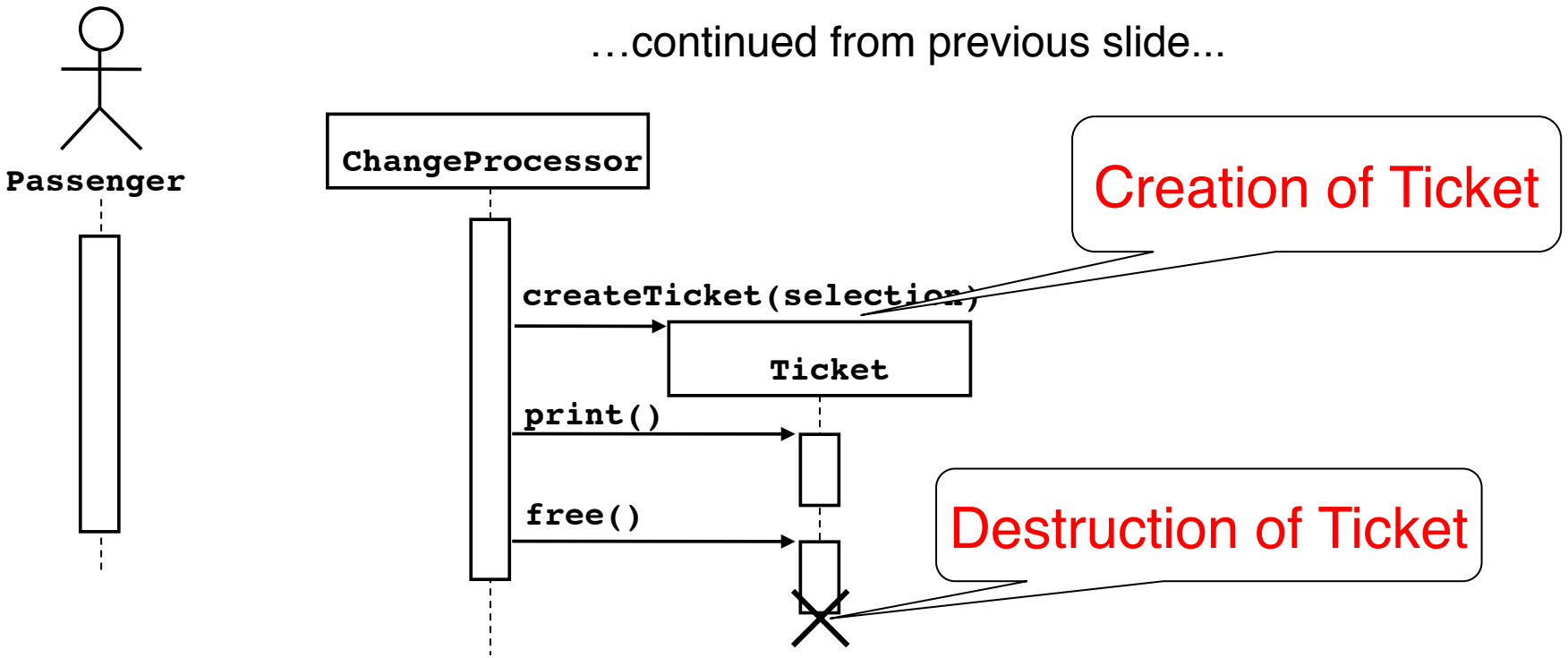
- The source of an arrow indicates the activation which sent the message
- Horizontal dashed arrows indicate data flow, for example return results from a message

Sequence Diagrams: Iteration & Condition



- Iteration is denoted by a * preceding the message name
- Condition is denoted by boolean expression in [ ] before the message name

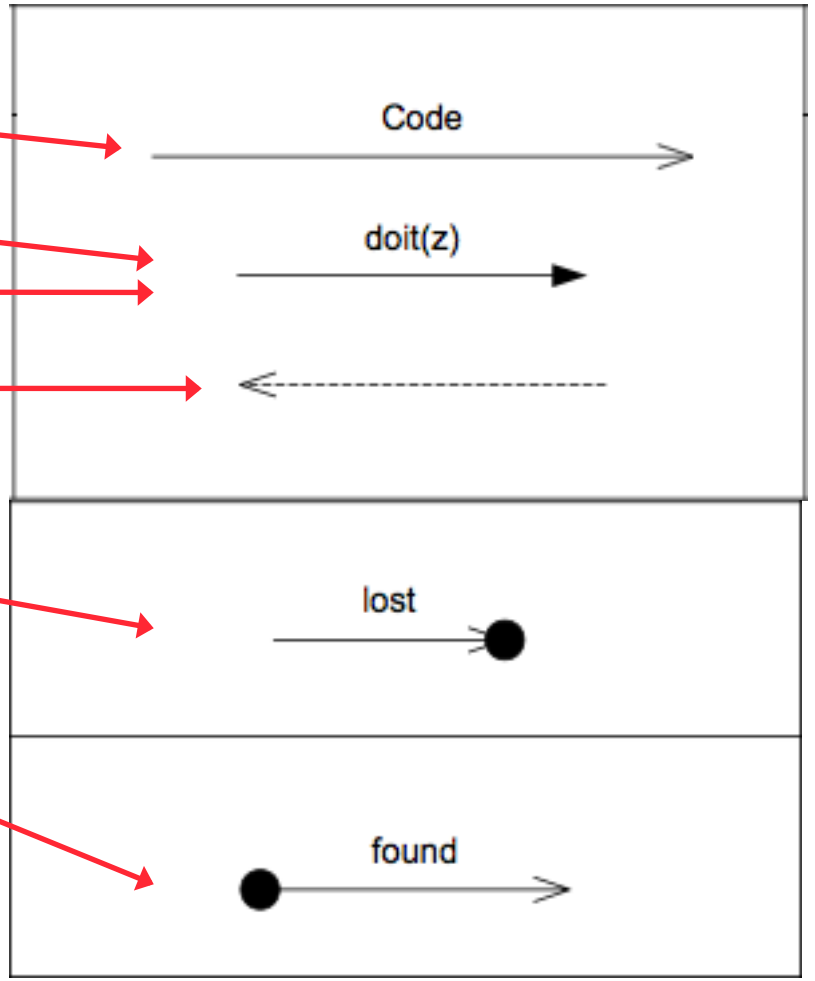
Creation and destruction



- Creation is denoted by a message arrow pointing to the object
- Destruction is denoted by an X mark at the end of the destruction activation
  - In garbage collection environments, destruction can be used to denote the end of the useful life of an object.

Message Types

- Asynchronous
- Synchronous
- Call and Object creation
- Reply
- Lost
- Found



Sequence Diagram Properties

- UML sequence diagram represent *behavior in terms of interactions*
- Useful to identify or find missing objects
- Time consuming to build, but worth the investment
- Complement the class diagrams (which represent structure).

Interaction Diagrams

Interaction Diagrams

- UML 2.0: New concept of **interaction fragments**

- Before we go into detail with interaction fragments, let's cover the concept of an **interaction.**

Interaction Diagrams

- Four types of interaction diagrams:
  - Sequence diagrams
  - We will not study the following (by now at least):
    - Communication diagrams
    - Interaction overview diagrams
    - Timing diagrams
- The basic building block of an interaction diagram is the interaction
  - An interaction is **a unit of behavior** that focuses on the **observable exchange of information** between connectable elements

Example of an Interaction: Sequence Diagram

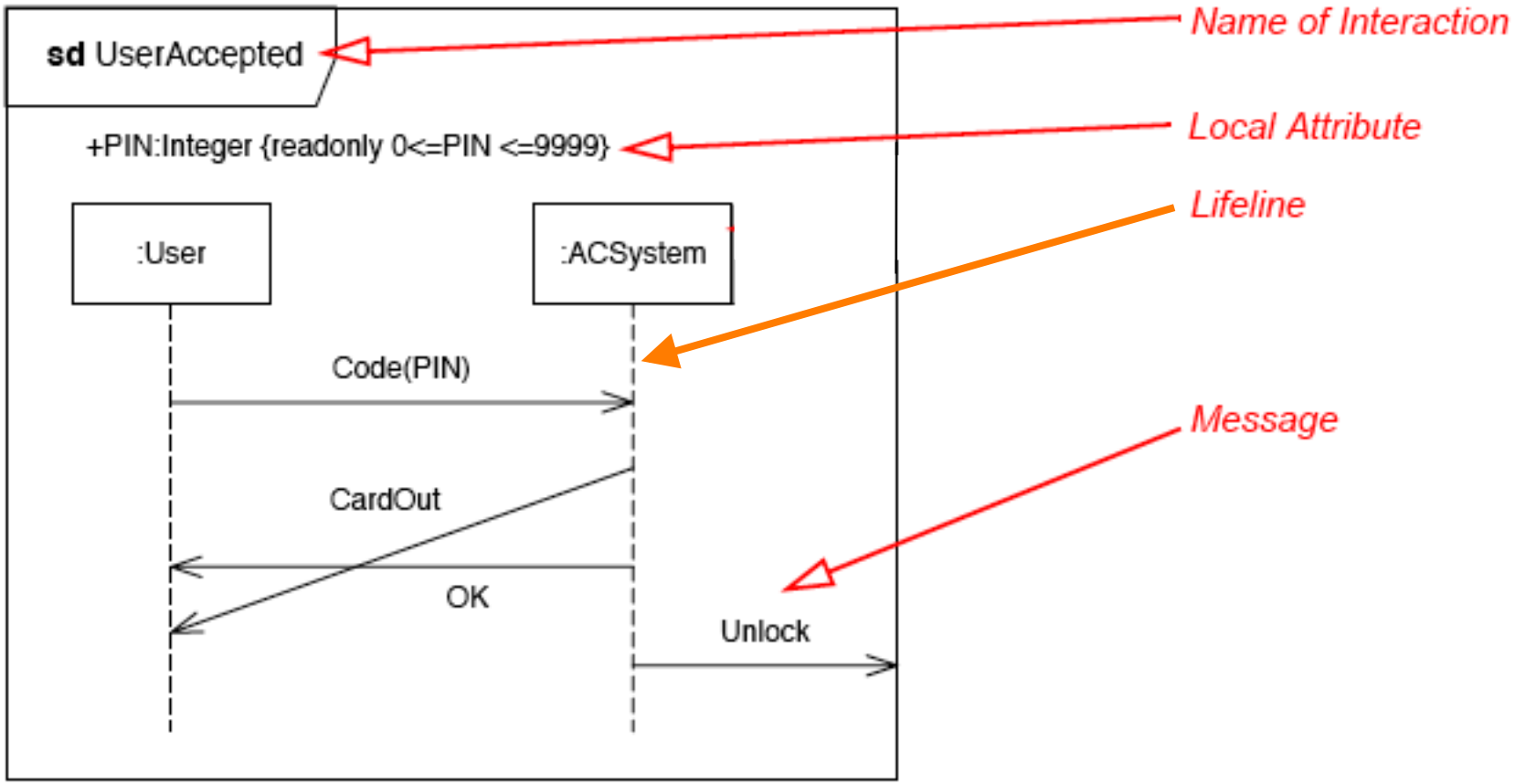


Figure 14.16 - An example of an Interaction in the form of a Sequence Diagram

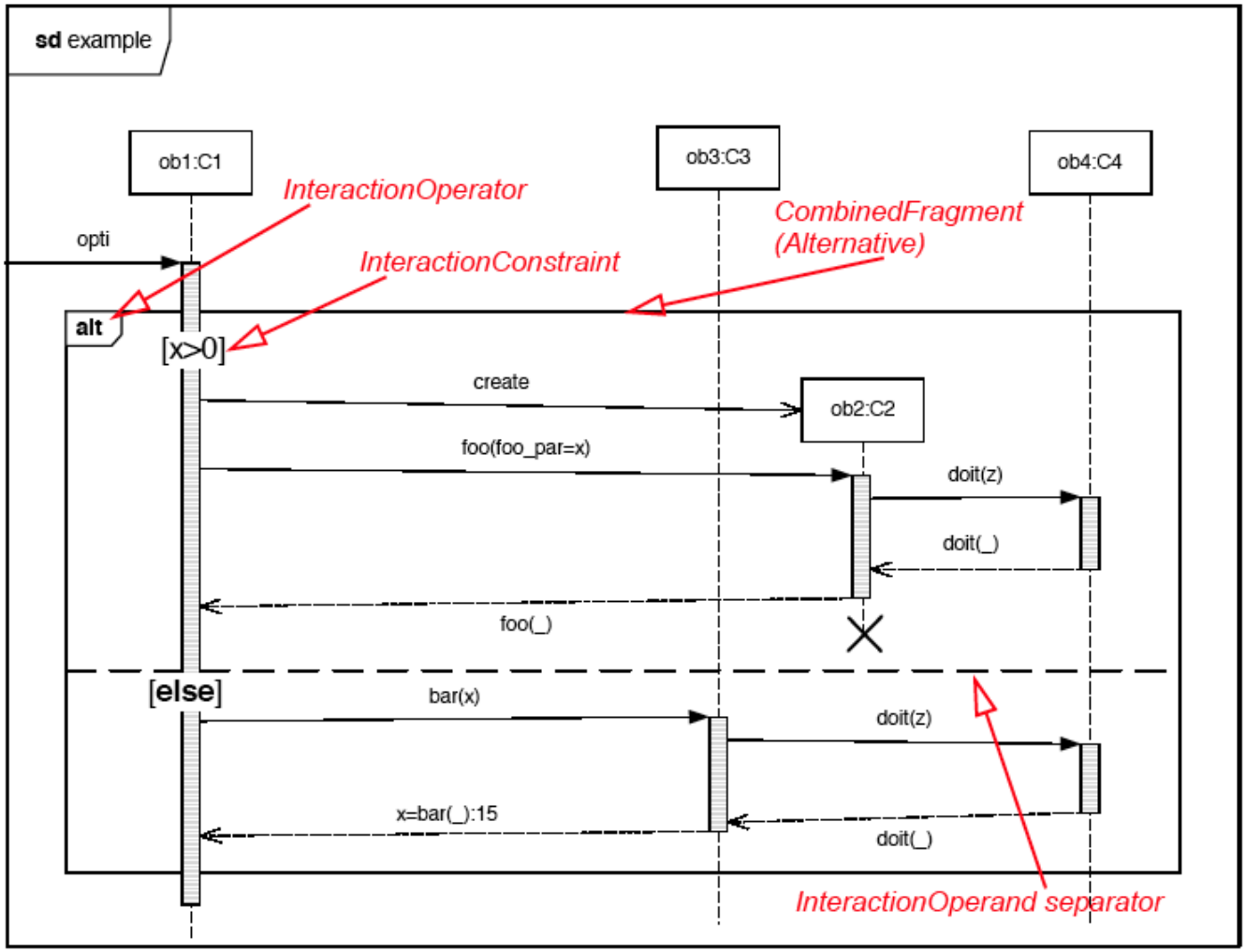
Interaction Fragment

- Interaction Fragment
  - Is a piece of an **interaction**
  - Acts like an interaction itself
- Combined Fragment
  - Is a subtype of interaction fragment
  - defines an expression of interaction fragments
- An expression of interaction fragments is defined by

  ⇨ an interaction operator and interaction operands.

Example of a Combined Fragment using the alt operator

- The interaction operator alt indicates a choice of behavior between interaction fragments



Alt Operator

- The interaction operator **alt** indicates a choice of behavior between interaction fragments
  - At most one interaction fragment (that is, an InteractionOperand) is chosen
  - The chosen interaction fragment must have an explicit or implicit guard expression that evaluates to true at this point in the interaction
    - A guard can be
      - a boolean expression (called InteractionConstraint)
      - `else` (a reserved word)
  - If the fragment has no guard expression, `true` is implied.

Interaction Operators

- The following operators are allowed in the combination of interaction fragments:
  - alt
  - opt
  - par
  - loop
  - critical
  - neg
  - assert
  - strict
  - seq
  - Ignore
  - consider

Opt and Break Operators

option:

The interaction operator **opt** designates a choice of behavior where either the (sole) operand happens or nothing happens.

break:

The interaction operator **break** represents a breaking scenario: The operand is a scenario that is performed instead of the remainder of the enclosing interaction fragment.

Parallel and Critical Operator

par

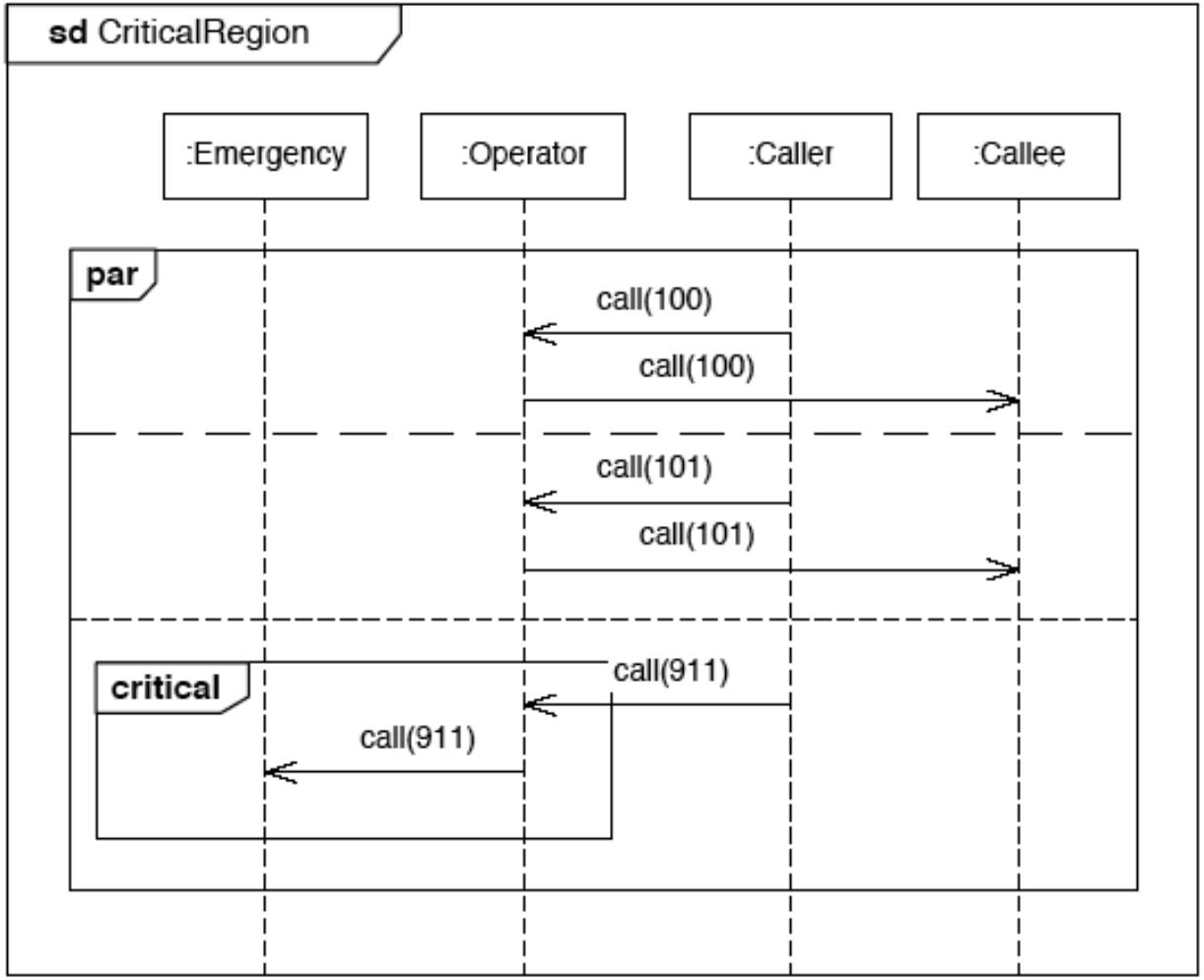
The interaction operator **par** designates a parallel merge between the behaviors of the operands of a combined fragment.

critical

The interaction operator **critical** designates that the combined fragment represents a critical region.

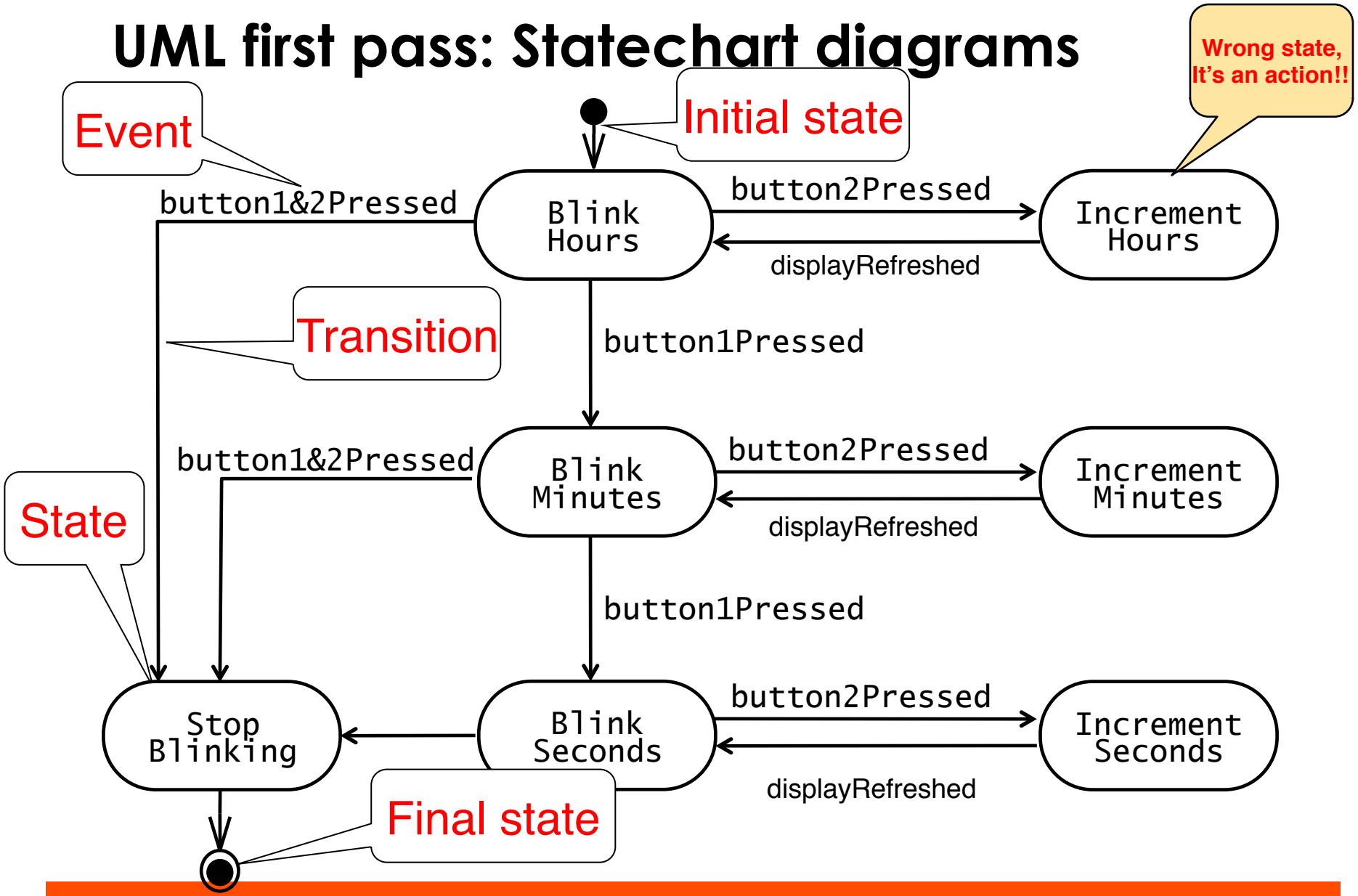
Example of a Critical Region

Problem statement: The telephone Operator must make sure to forward a 911-call from a Caller to the Emergency system before doing anything else. Normal calls can be freely interleaved.

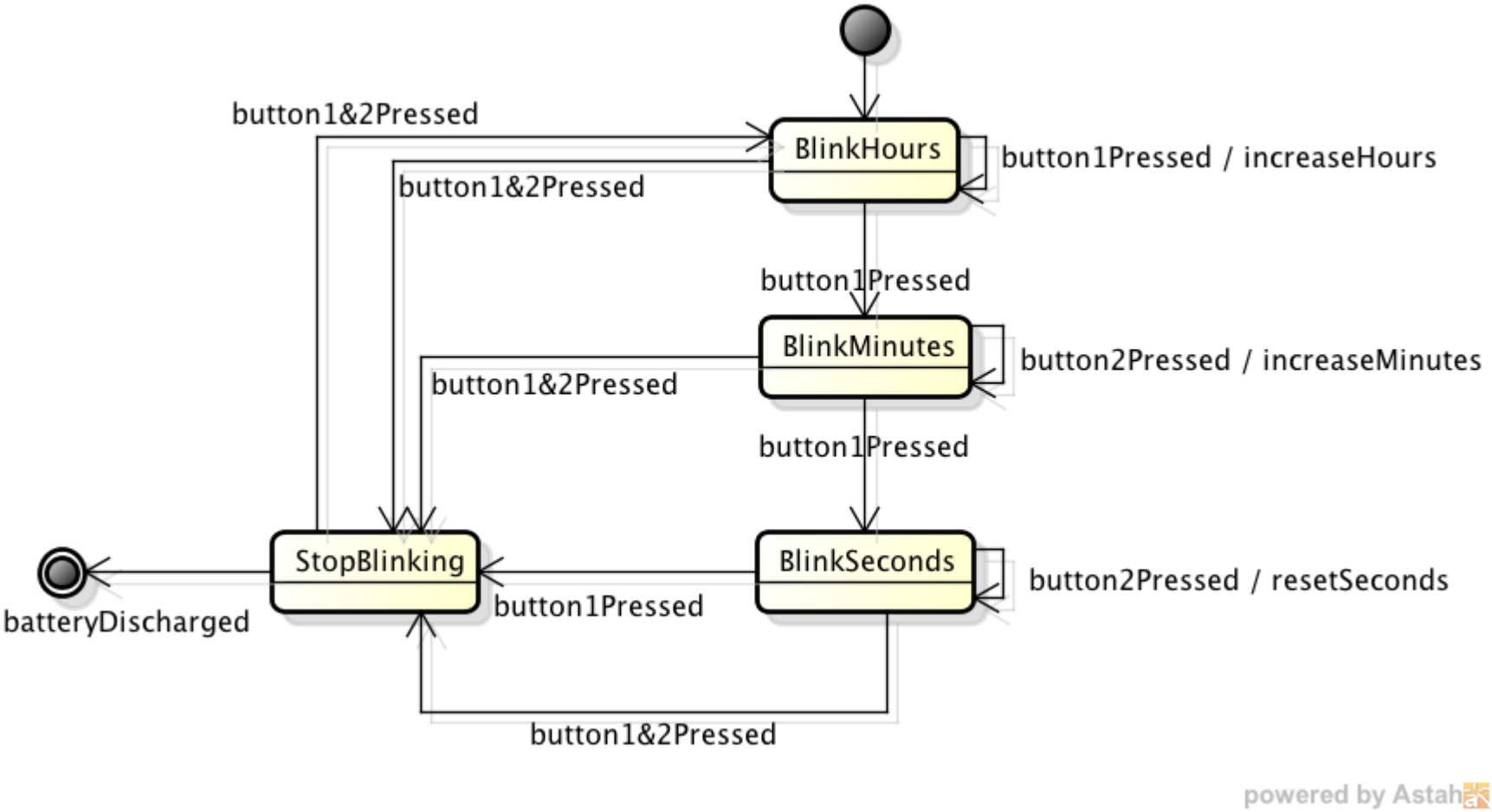


UML Statechart Diagram

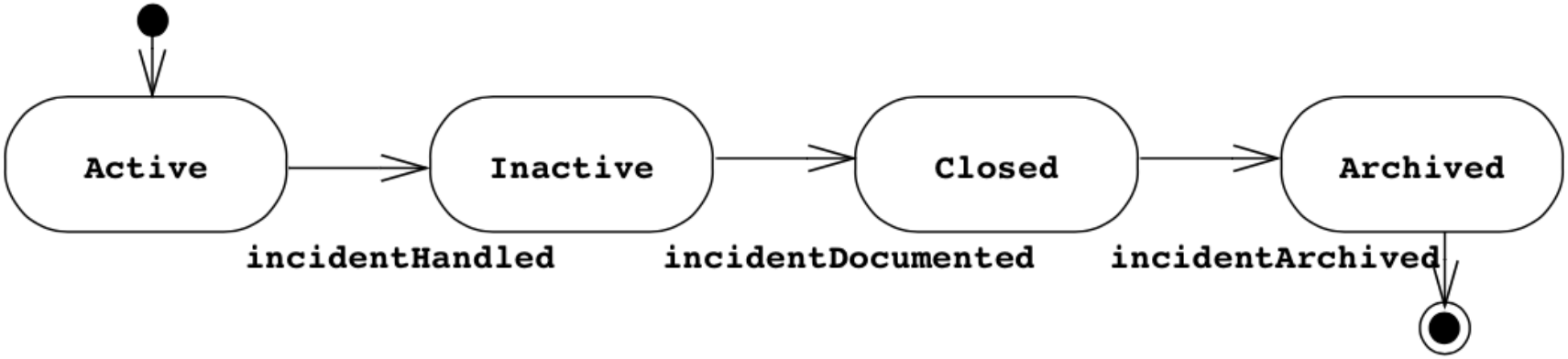
UML first pass: Statechart diagrams



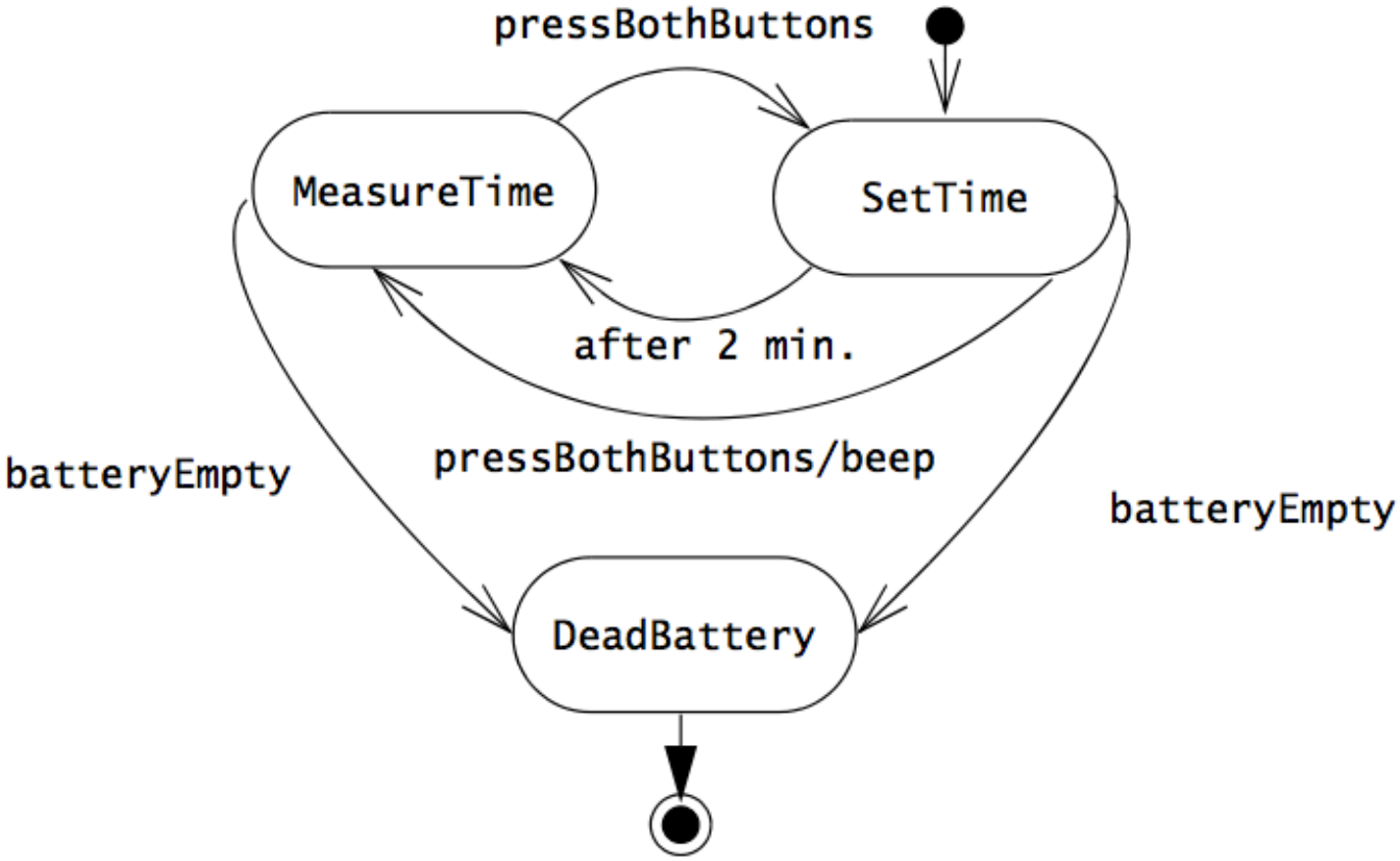
Represent behavior of *a single object* with interesting dynamic behavior.


button1&2Pressed
BlinkHours
button1Pressed / increaseHours
button1&2Pressed
button1Pressed
BlinkMinutes
button2Pressed / increaseMinutes
button1&2Pressed
button1Pressed
StopBlinking
BlinkSeconds
button2Pressed / resetSeconds
batteryDischarged
button1Pressed
button1&2Pressed
powered by Astah

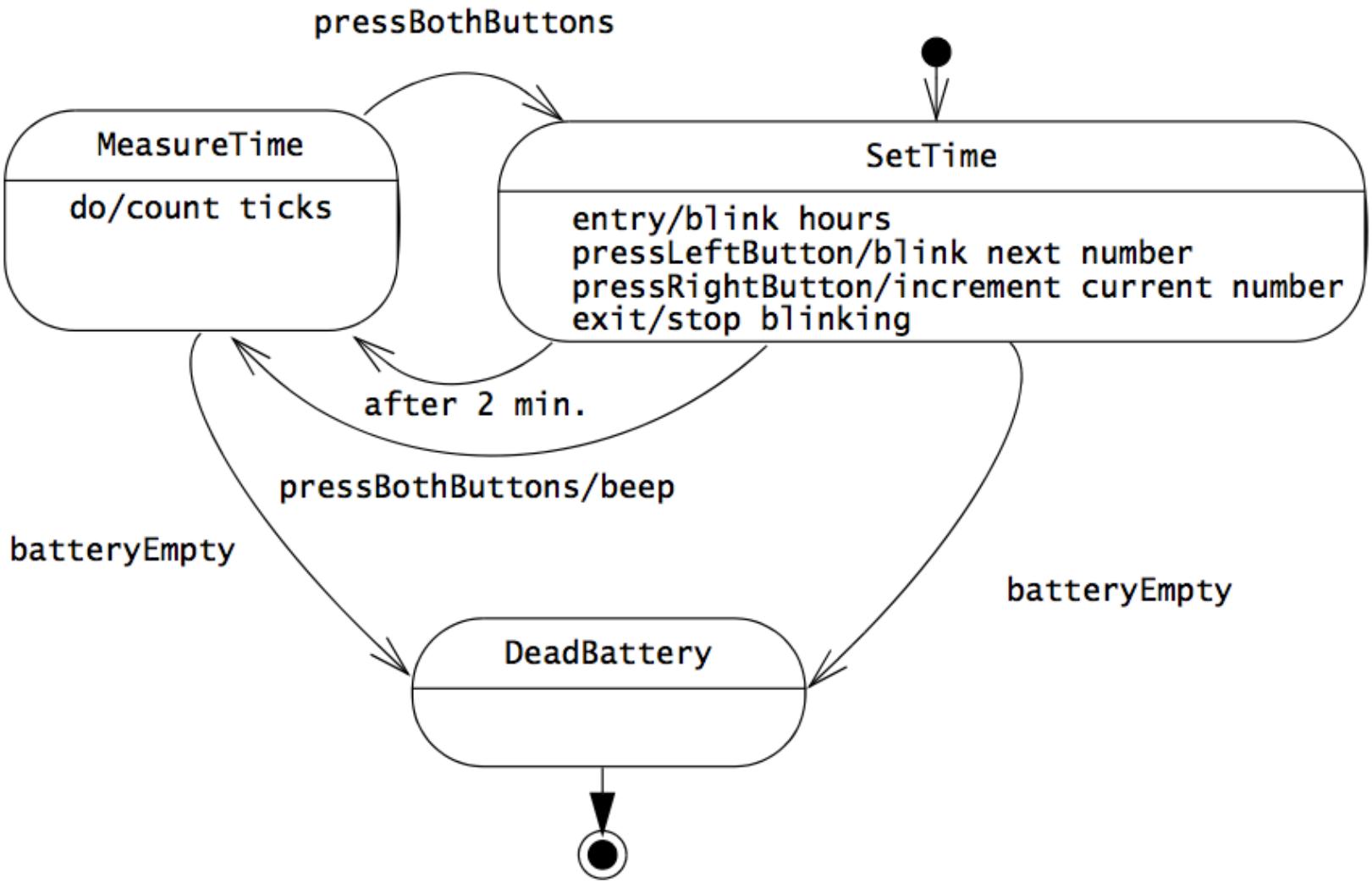
Statechart for the Incident class



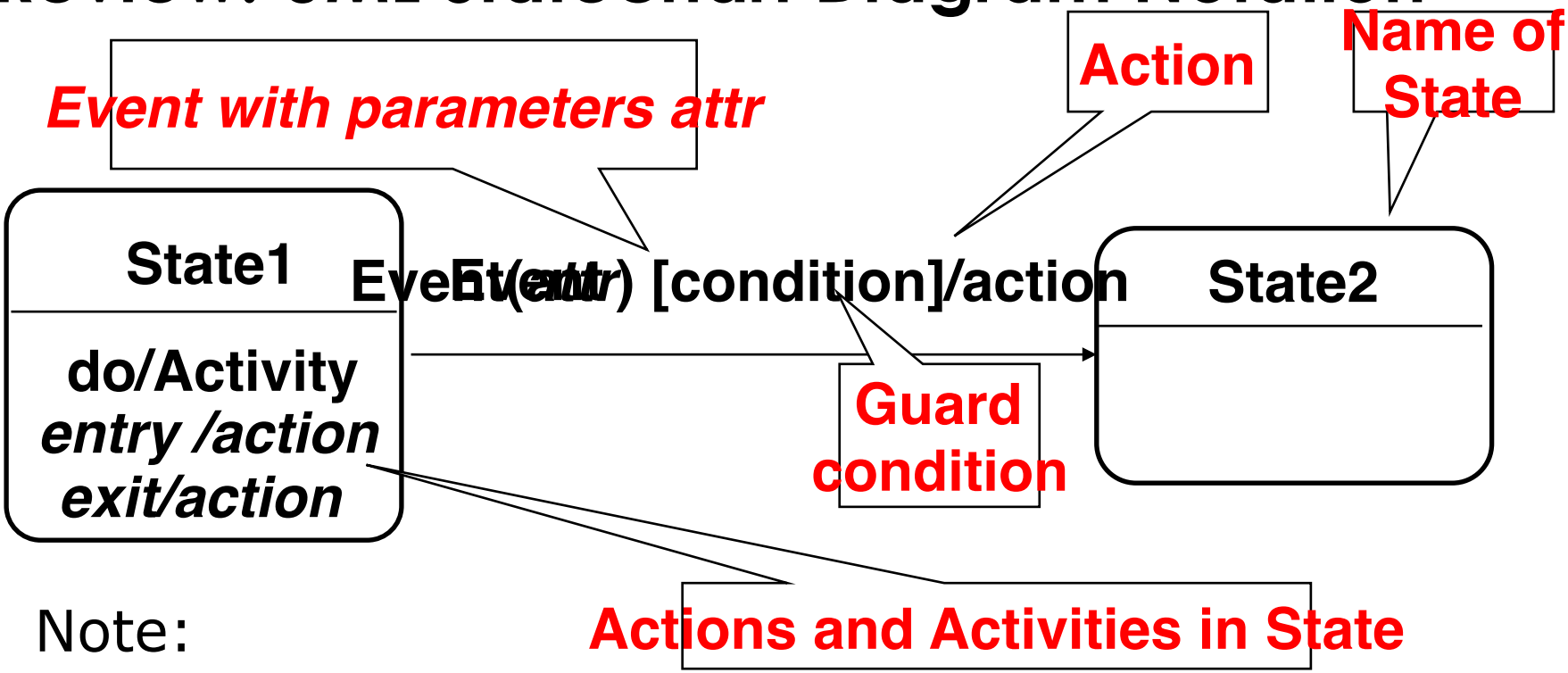
State machine diagram for 2Bwatch



Internal transitions in 2BWatch statechart

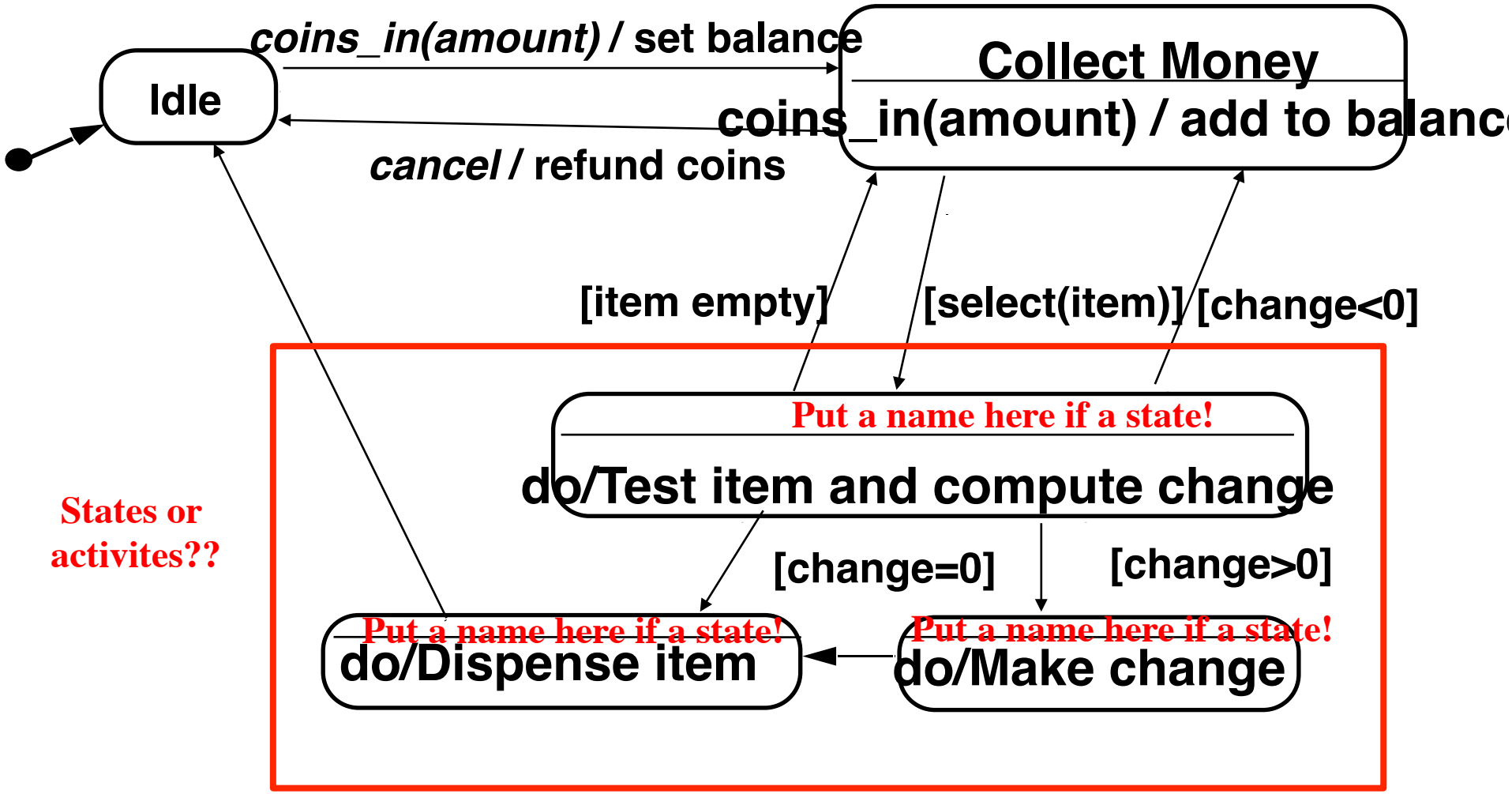


Review: UML Statechart Diagram Notation

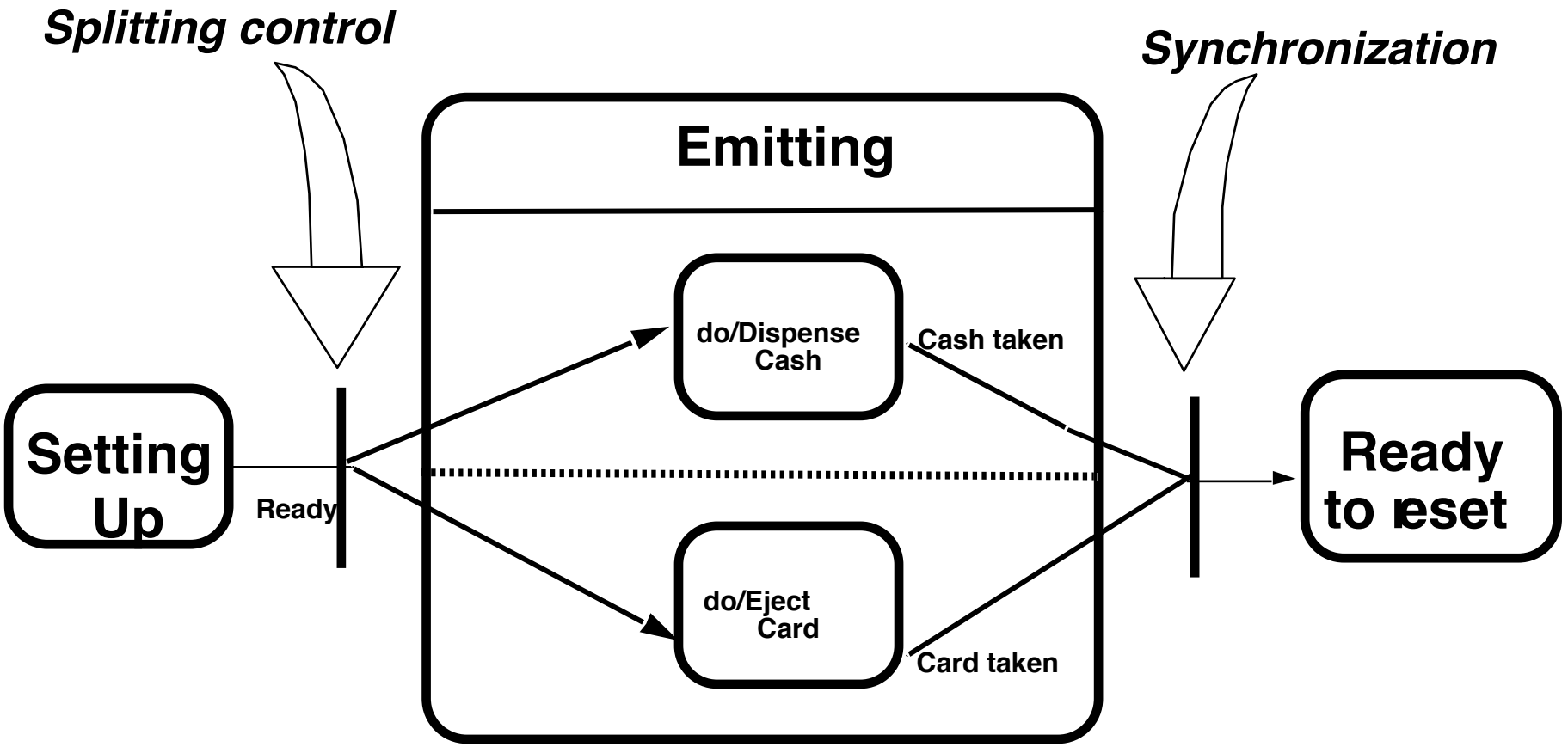


- Note:
  - *Events are italics*
  - Conditions are enclosed with brackets: []
  - Actions are prefixed with a slash /

Example of a StateChart Diagram



Example of Concurrency within an Object



UML Activity Diagram

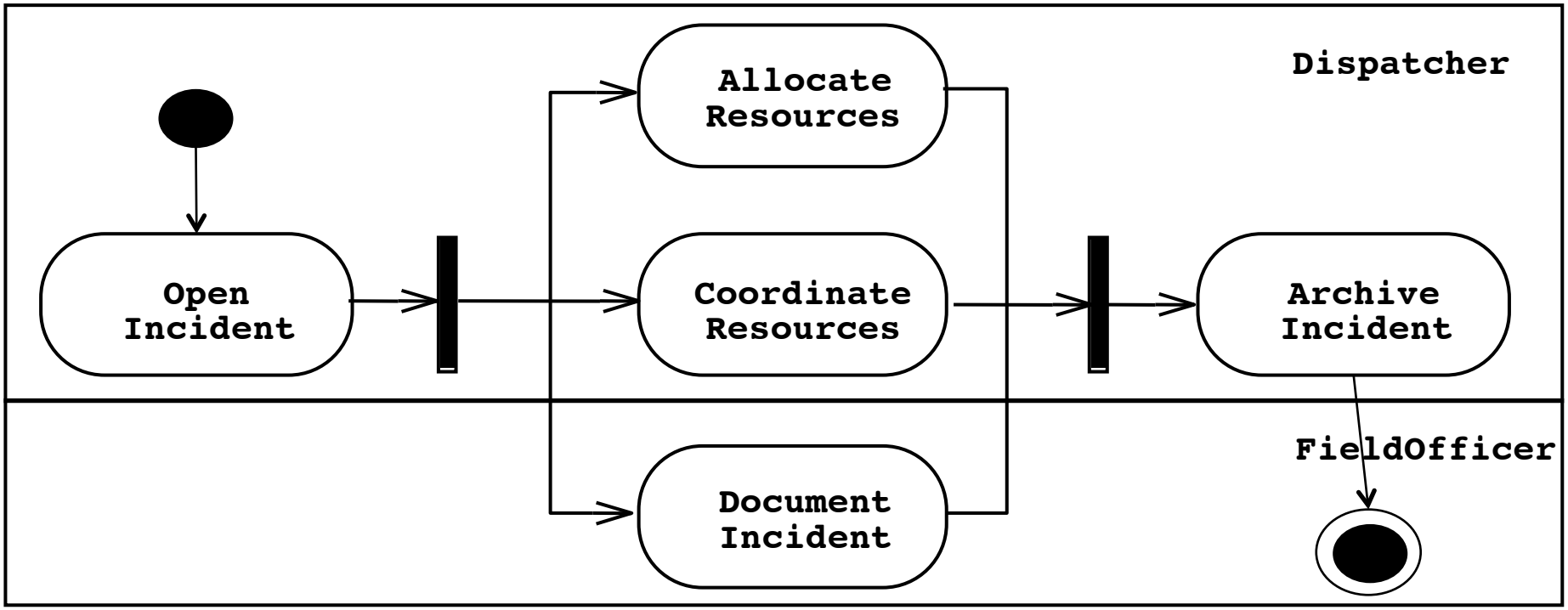
UML Activity Diagrams

An activity diagram consists of nodes and edges

- **Nodes describe activities and objects**
  - Control nodes
  - Executable nodes
    - Most prominent: **Action**
  - Object nodes
    - E.g. a document
- **Edge** is a directed connection between nodes
  - There are two types of edges
    - Control flow edges
    - Object flow edges

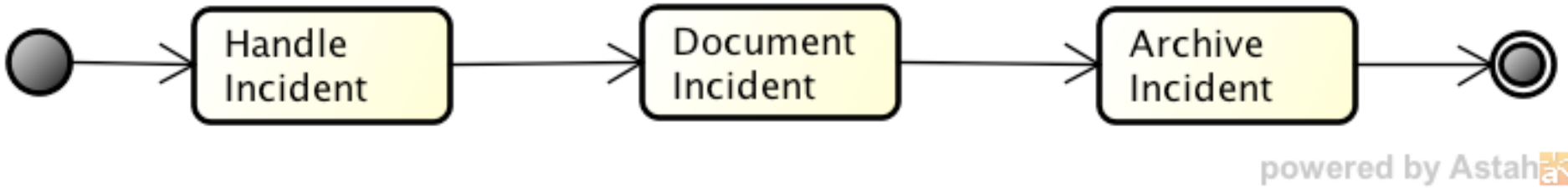
Activity Diagrams: Grouping of Activities

- Activities may be grouped into swimlanes to denote the object or subsystem that implements the activities.



State Chart Diagrams vs Activity Diagrams

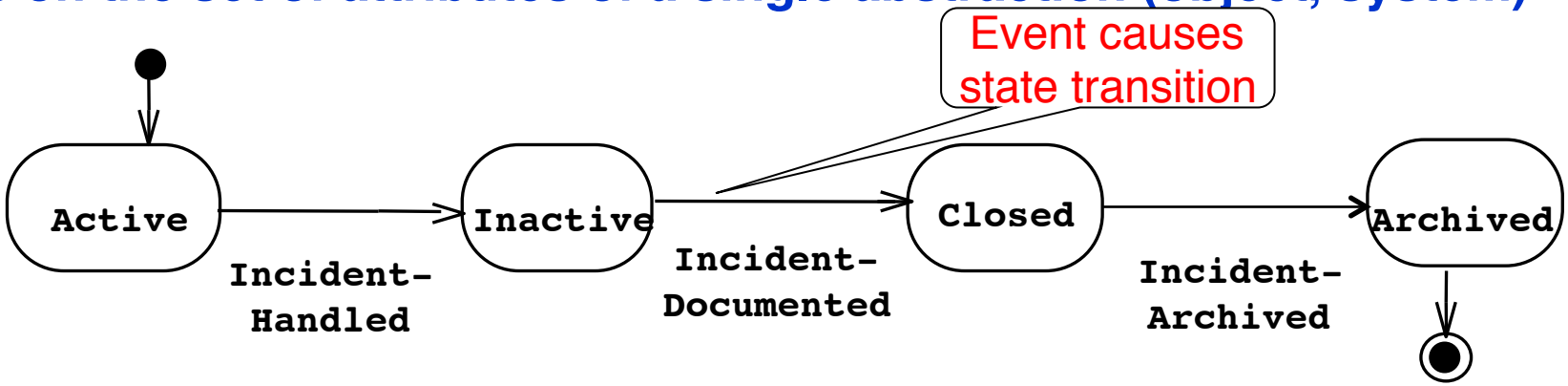
- An activity diagram that contains only activities can be seen as a special case of a state chart diagram
- Such an activity diagram is useful to describe the overall workflow of a system



Statechart Diagram vs Activity Diagram

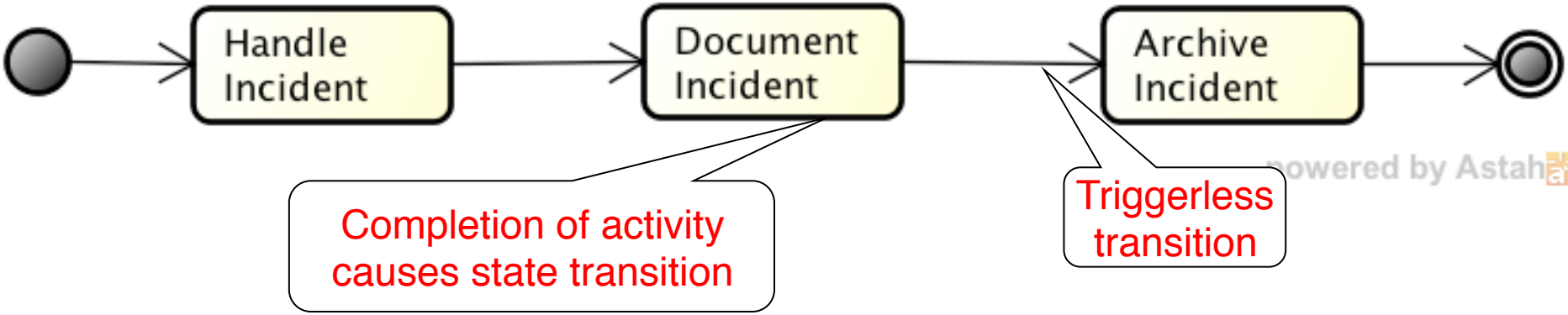
Statechart Diagram for Incident

Focus on the set of attributes of a single abstraction (object, system)

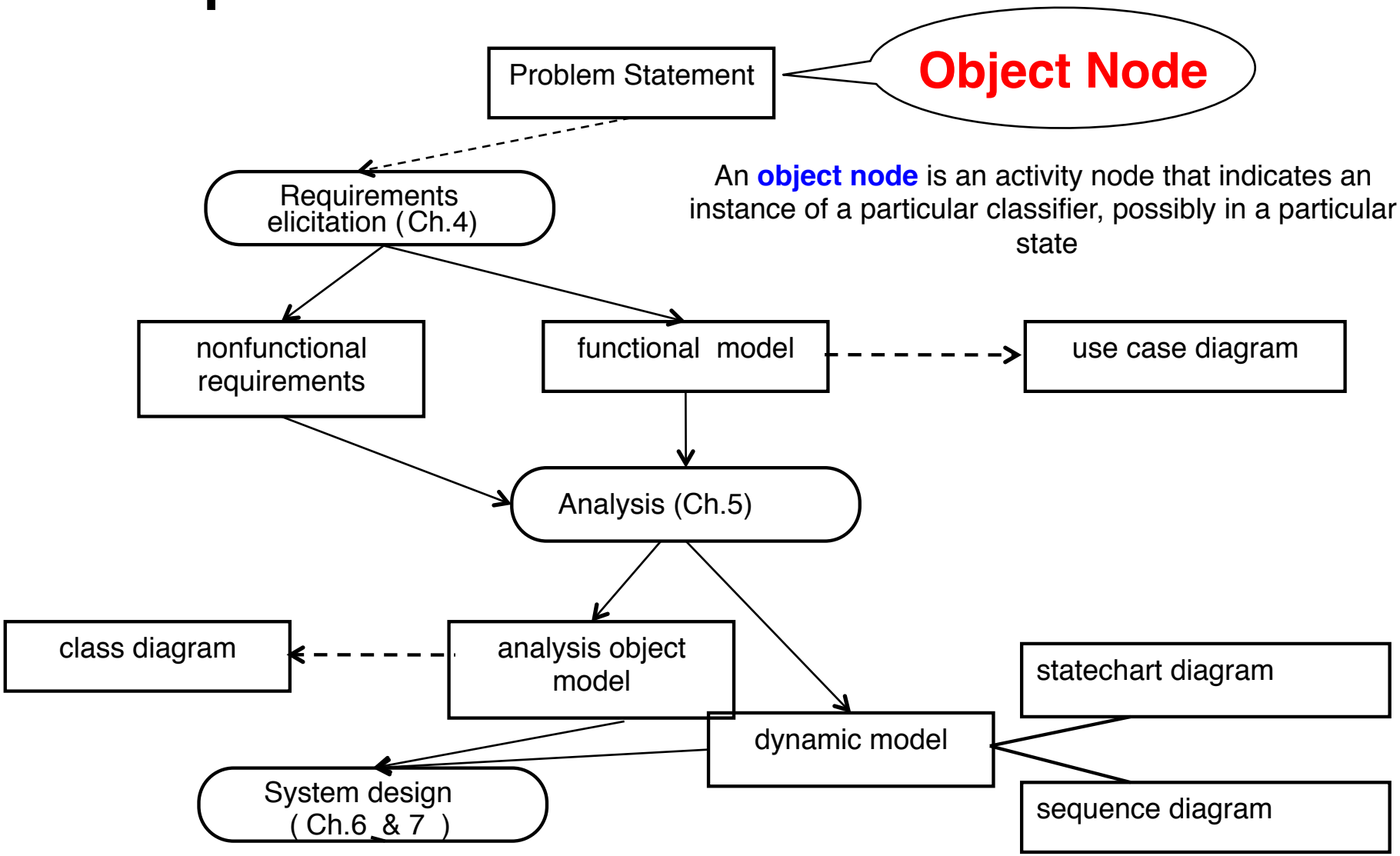


Activity Diagram for Incident

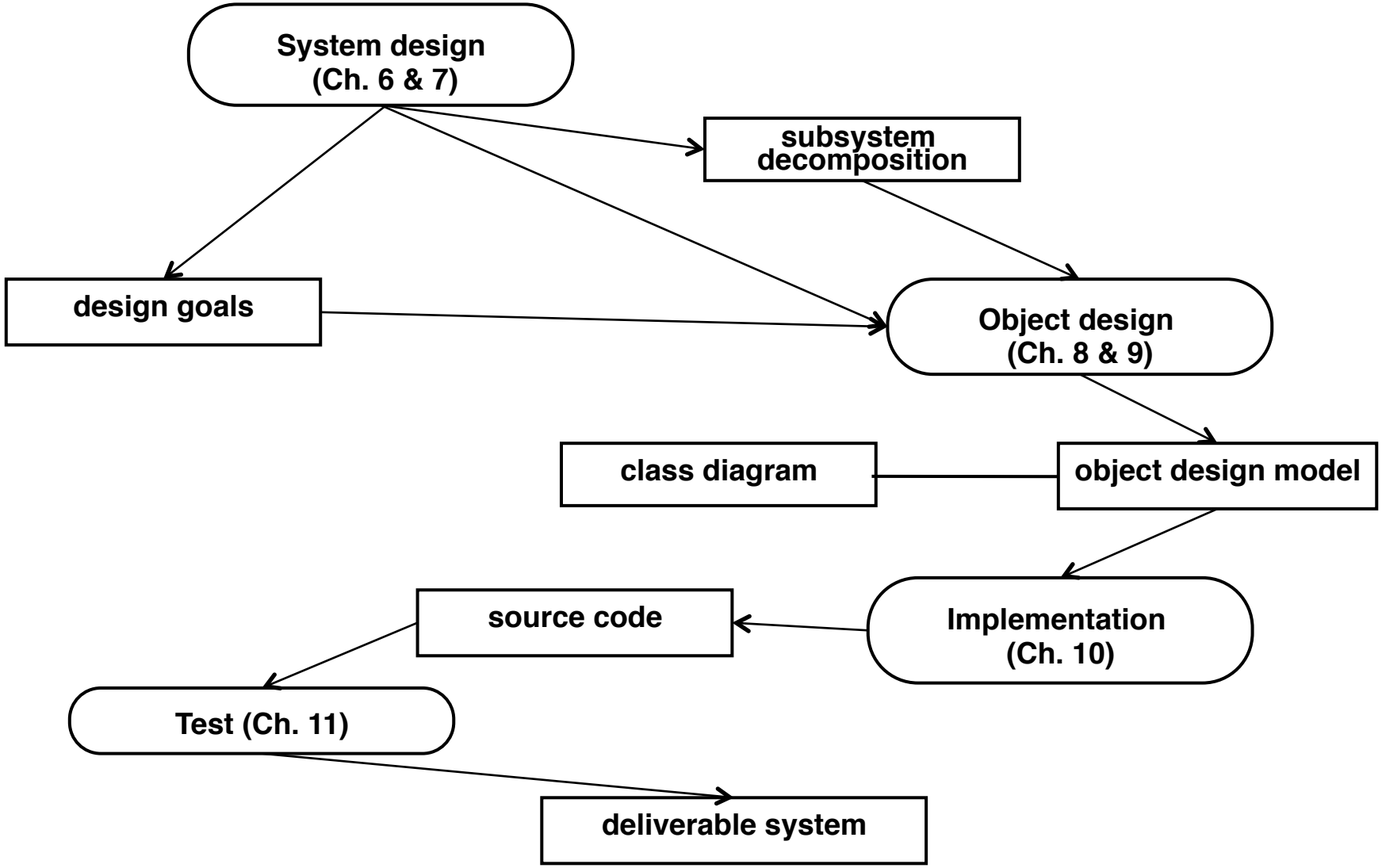
(Focus on dataflow in a system)



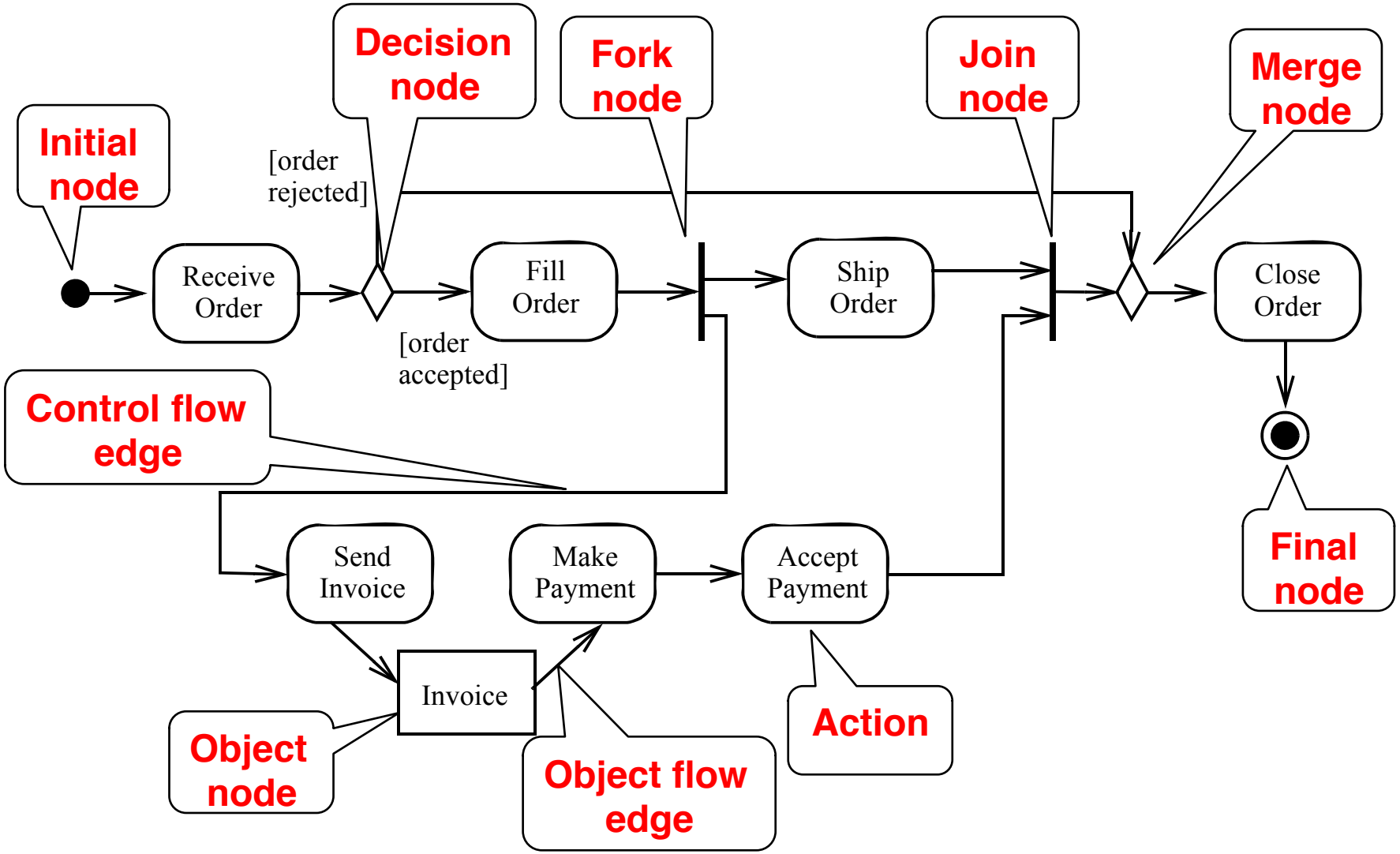
Example: Structure of the Text Book



Example: Structure of the Text Book (2)



Summary: Activity Diagram Example



What should be done first? Coding or Modeling?

- It depends….
- Forward Engineering
  - Creation of code from a model
  - Start with modeling
  - Greenfield projects
- Reverse Engineering
  - Creation of a model from existing code
  - Interface or reengineering projects
- Roundtrip Engineering
  - Move constantly between forward and reverse engineering
  - Reengineering projects
  - Useful when requirements, technology and schedule are changing frequently.

Additional References

- Martin Fowler
  - UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd ed., Addison-Wesley, 2003
- Grady Booch,James Rumbaugh,Ivar Jacobson
  - The Unified Modeling Language User Guide, Addison Wesley, 2nd edition, 2005
- Open Source UML tools
  - **Astah Community**: http://astah.net/editions/community
  - http://java-source.net/open-source/uml-modeling

UML Summary

- UML provides a wide variety of notations for representing many aspects of software development
  - Powerful, but complex
- UML is a programming language
  - Can be misused to generate unreadable models
  - Can be misunderstood when using too many exotic features
- We concentrated on a few notations:
  - Functional model: Use case diagram
  - Object model: class diagram
  - Dynamic model: sequence diagrams, statechart and activity diagrams.