Chapter 2, Modeling with UML

Overview: modeling with UML

- What is modeling?
- What is UML?
- Use case diagrams
- Class diagrams
- Sequence diagrams
- Activity diagrams

What is modeling?

- Modeling consists of building an abstraction of reality.
- Abstractions are simplifications because:
  - **They ignore irrelevant details and**
  - **They only represent the relevant details.**
- Focus only on relevant parts of the problem
- What is *relevant* or *irrelevant* depends on the purpose of the model.
- ‘Throw away’ the details.

Why model software?

- Software is getting increasingly more complex
    - **Windows XP > 40 million lines of code**
    - **A single programmer cannot manage this amount of code in its entirety.**
- We need simpler representations for complex systems
    - **Modeling is a means for dealing with complexity**

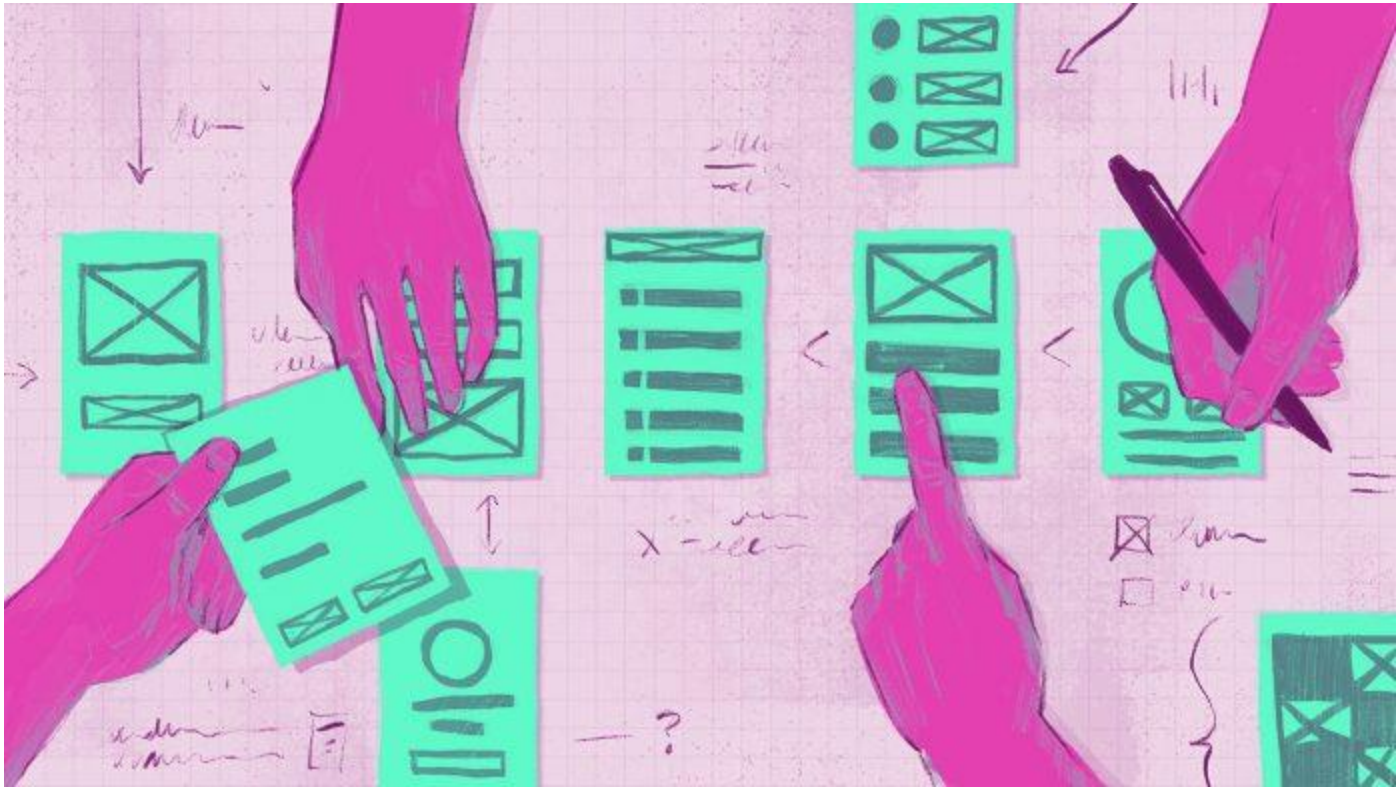
Why model software ?

Visualize software before its produced

Why model software ?

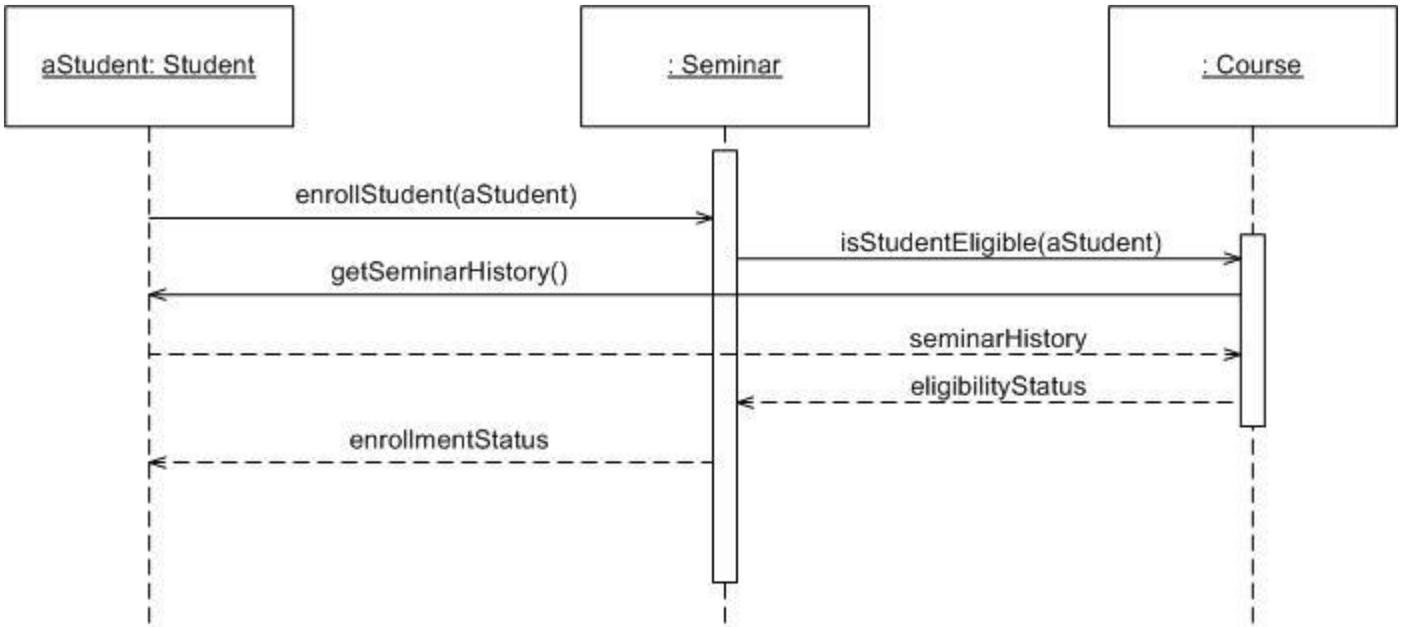
Code is not easily understandable by developers who did not write it

Why model software ?



Document the design decisions & Communicate Ideas

Why model software ?



Provide template for guiding the software production

Example: House

Floor plan

Interior Plan

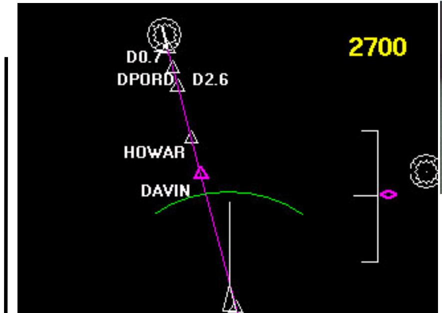
Wiring Plan

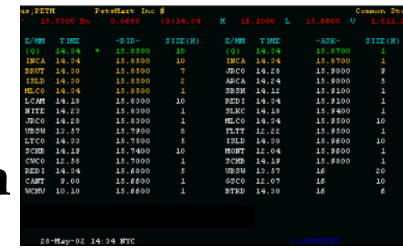
Application vs Solution Domain

- Application Domain (Analysis):
  - **The environment in which the system is operating**
- Solution Domain (Design, Implementation):
  - **The technologies used to build the system**
  - **Modeling space of all possible systems**
- Both domains contain abstractions that we can use for the construction of the system model.

Object-oriented Modeling

Application Domain





Solution Domain

System Model (Concepts) *(Analysis)*

UML Package

TrafficControl

Aircraft

TrafficController

Airport

FlightPlan

System Model (Concepts) *(Design)*

MapDisplay

Summary Display

FlightPlanDatabase

TrafficControl

What is UML?

- The UML is a language for
  - **visualizing**
  - **specifying**
  - **constructing**
  - **documenting**

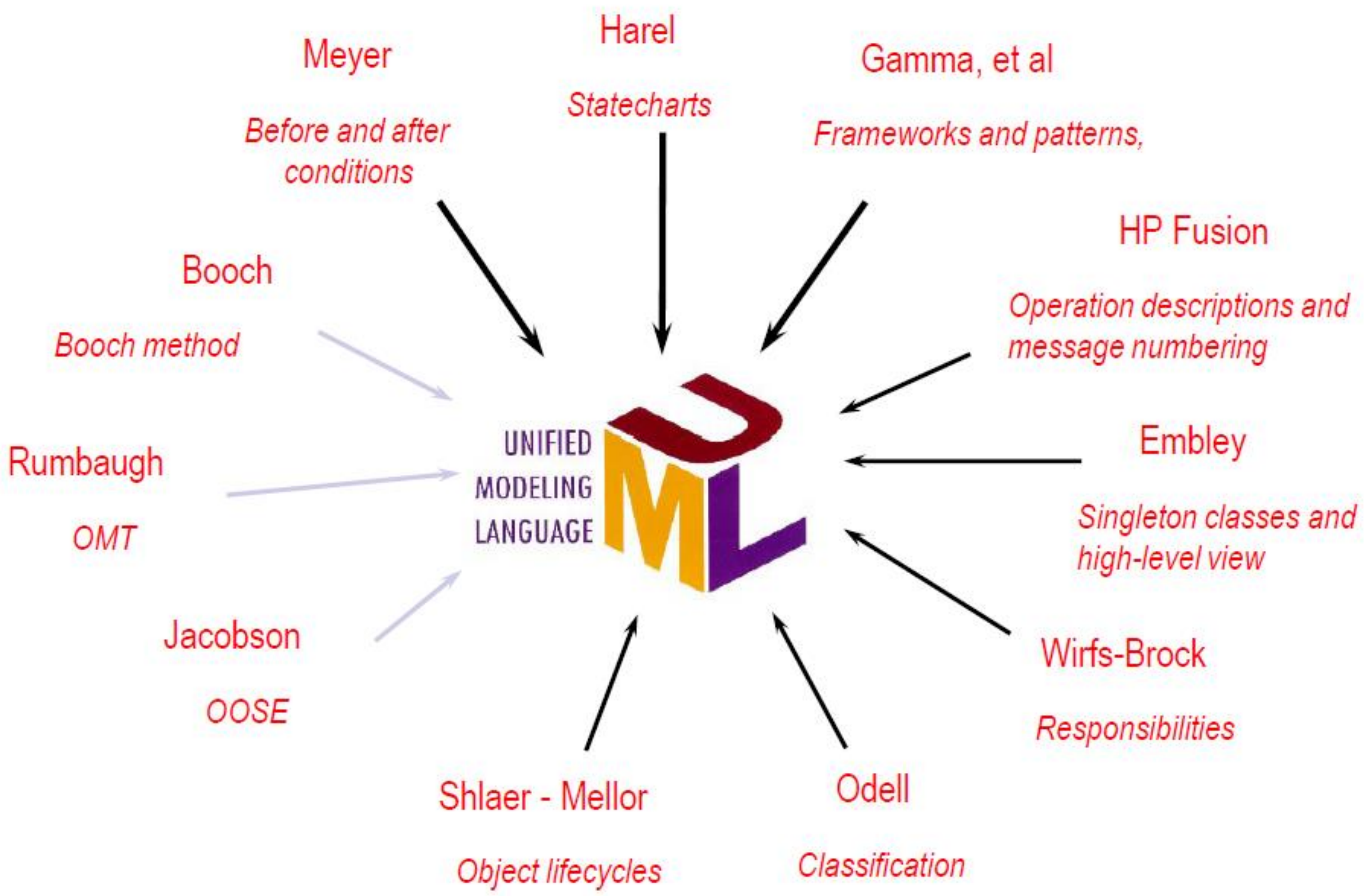
the artifacts of a software-intense system

« *The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components.*»

What is UML?

- UML (Unified Modeling Language)
  - **Nonproprietary standard for modeling software systems, OMG**
  - **Convergence of notations used in object-oriented methods**
    - **OMT (James Rumbaugh and collegues)**
    - **Booch (Grady Booch)**
    - **OOSE (Ivar Jacobson)**
- Current Version: UML 2.5
  - **Information at the OMG portal http://www.uml.org/**
- Commercial tools: Rational (IBM), Together (Borland), Visual Paradigm
- Open Source tools: ArgoUML, UMLet, Umbrello

Contributions to the UML



UML: First Pass

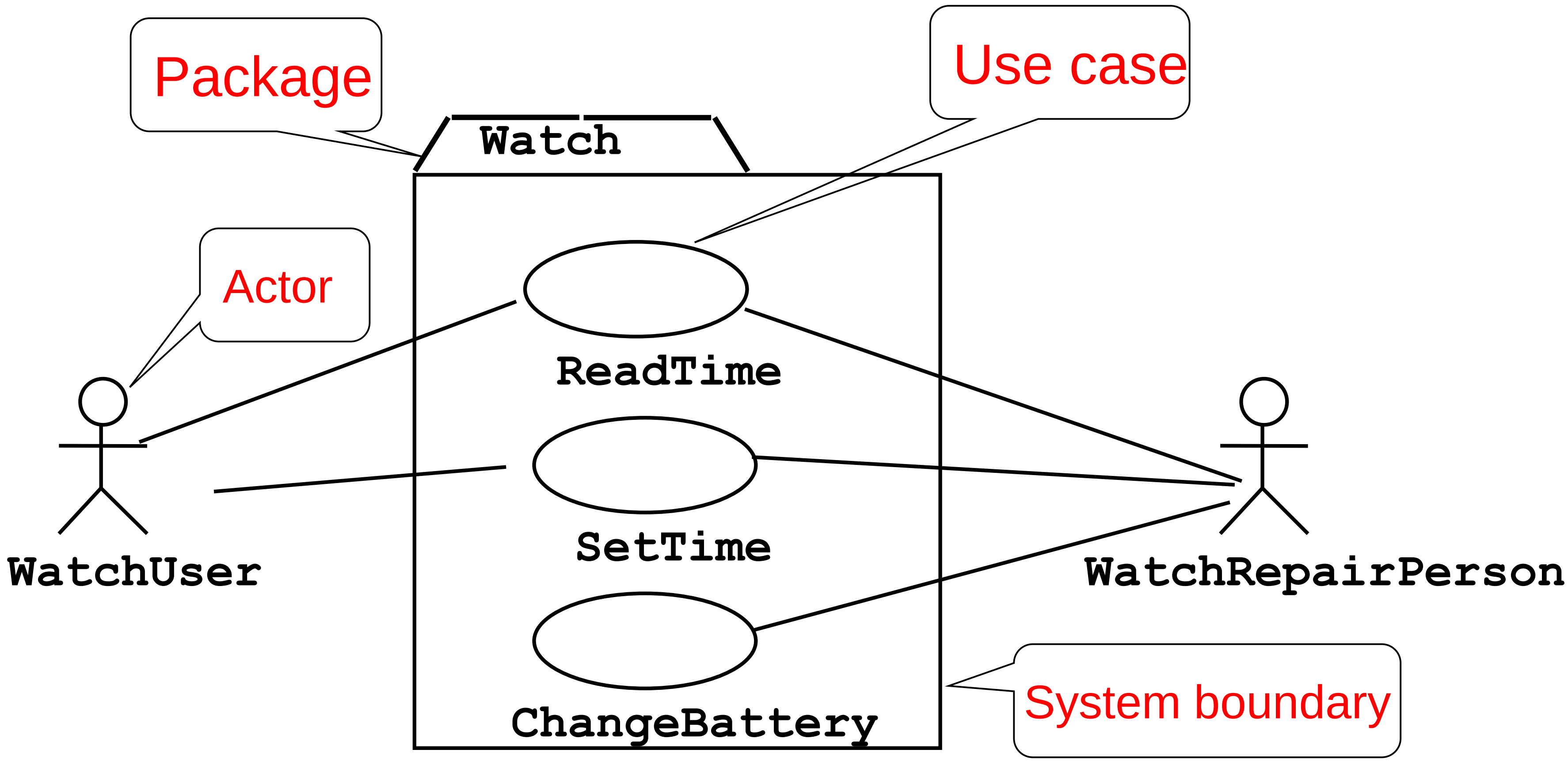
- You can model 80% of most problems by using about 20% UML
- We teach you those 20%

- 80-20 rule: Pareto principle (http://en.wikipedia.org/wiki/Pareto_principle)
  - **80% of your profits come from 20% of your customers**
  - **80% of your complaints come from 20% of your customers**
  - **80% of your profits come from 20% of the time you spend**
  - **80% of your sales come from 20% of your products**

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**
- State diagrams
  - **Describe the dynamic behavior of an individual object**

UML first pass: Use case diagrams

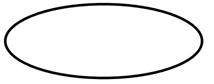


Use case diagrams represent the functionality of the system from user's point of view

UML Core Conventions

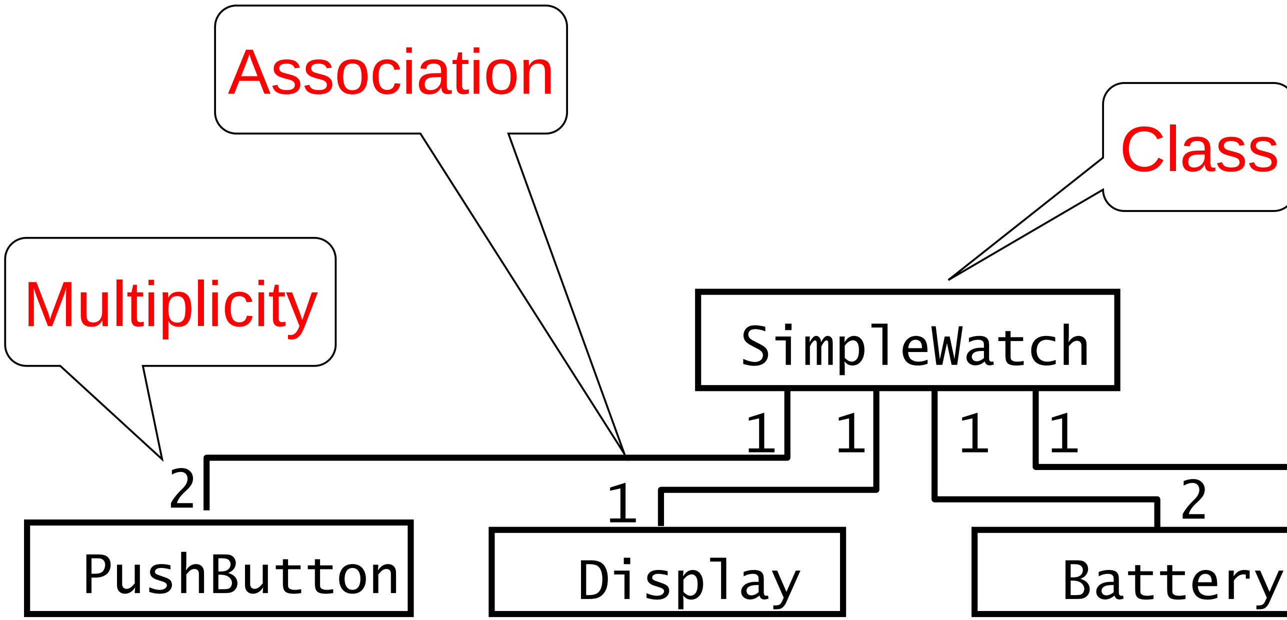
- All UML Diagrams denote graphs of nodes and edges
  - **Nodes are entities and drawn as rectangles or ovals**
  - **Rectangles denote classes or instances**

  - **Ovals denote functions**



- Names of Classes are not underlined
  - `SimpleWatch`
  - `Firefighter`
- Names of Instances are underlined
  - <u>`myWatch:SimpleWatch`</u>
  - <u>`Joe:Firefighter`</u>
- An edge between two nodes denotes a relationship between the corresponding entities

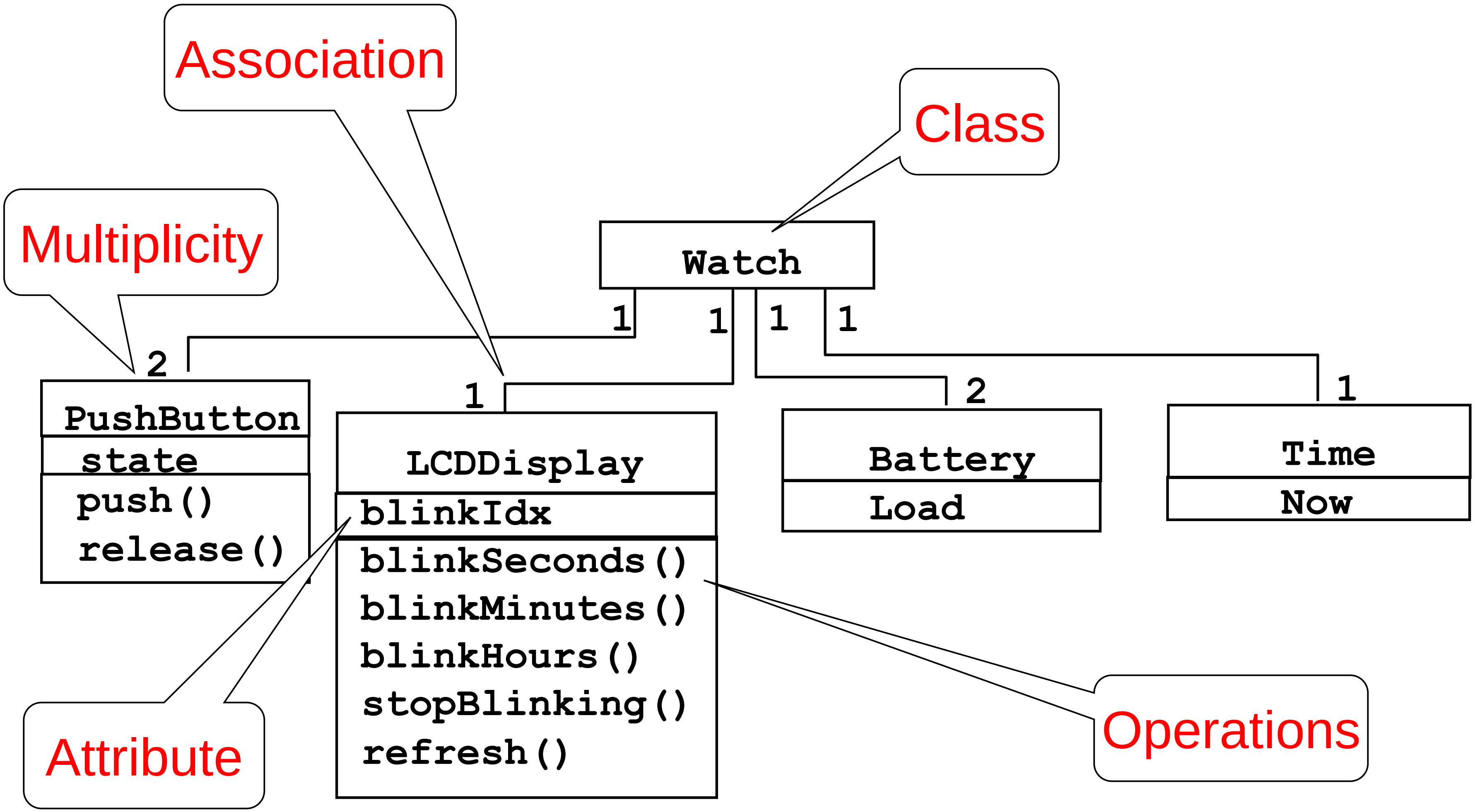
UML first pass: Class diagrams



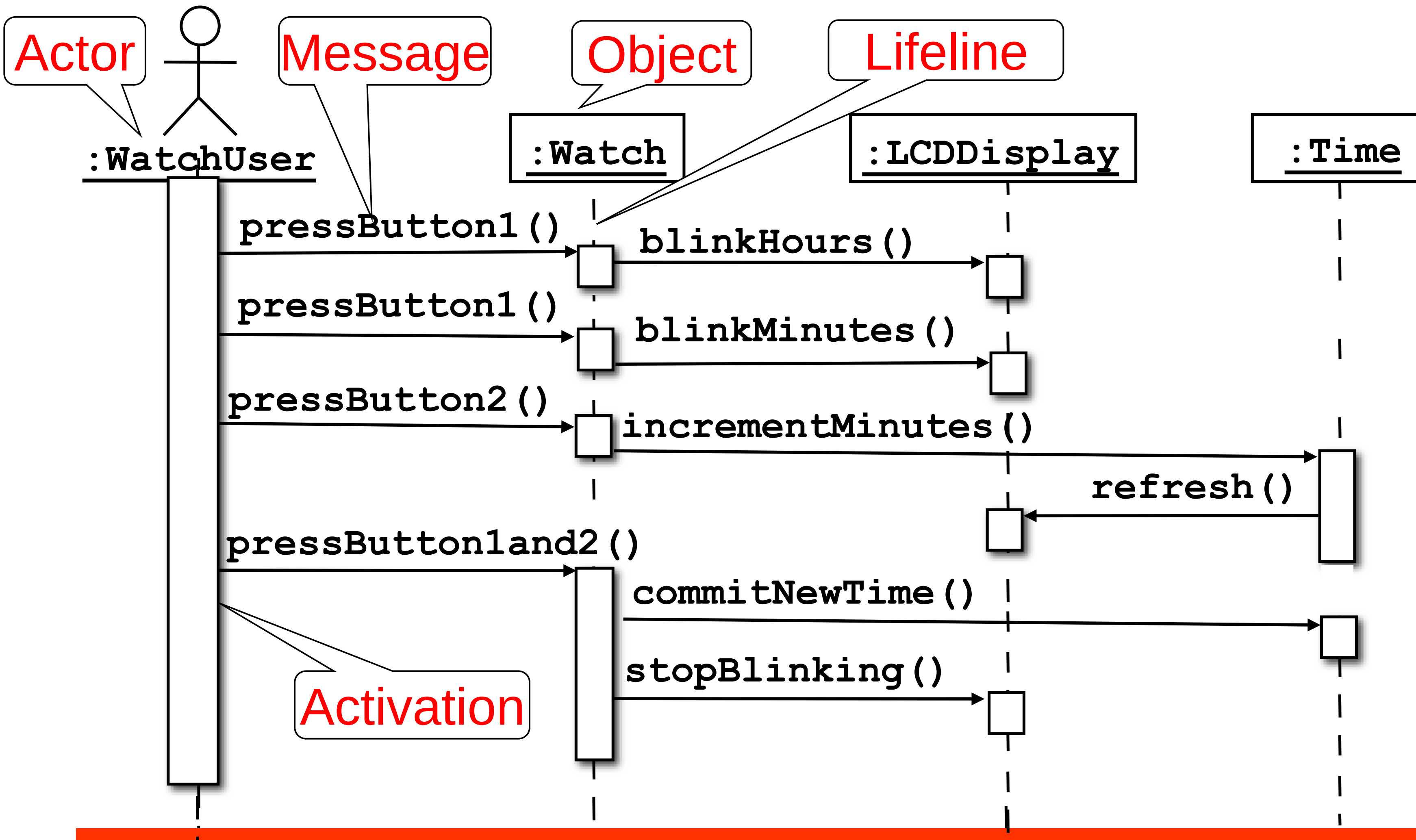
Class diagrams represent the structure of the system

UML first pass: Class diagrams

Class diagrams represent the structure of the system

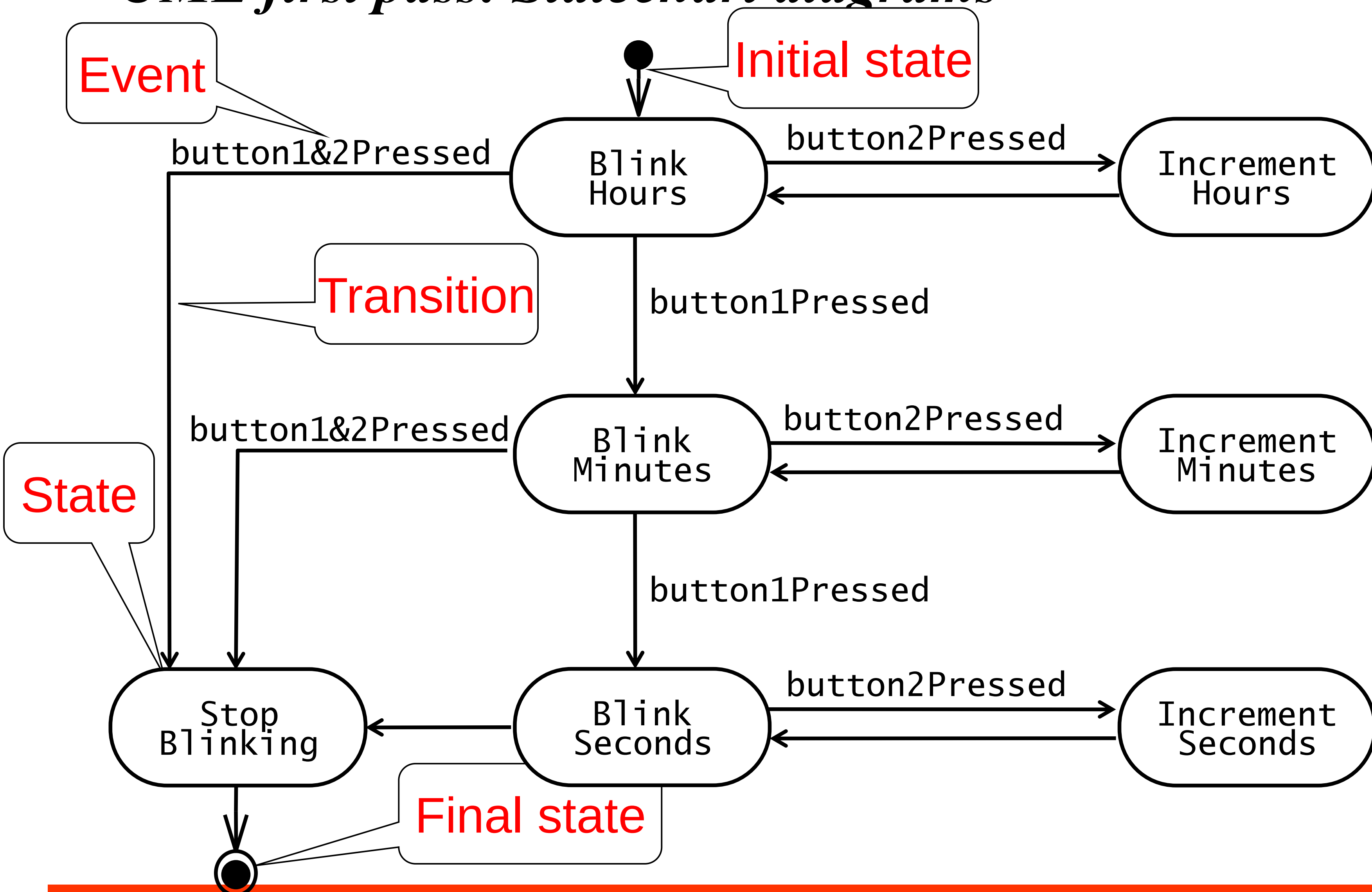


UML first pass: Sequence diagram



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

UML first pass: Statechart diagrams



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

Other UML Notations

UML provides many other notations, for example

- Deployment diagrams for modeling configurations
  - **Useful for testing and for release management**
- We introduce these and other notations as we go along in the lectures
  - **OCL: A language for constraining UML models**

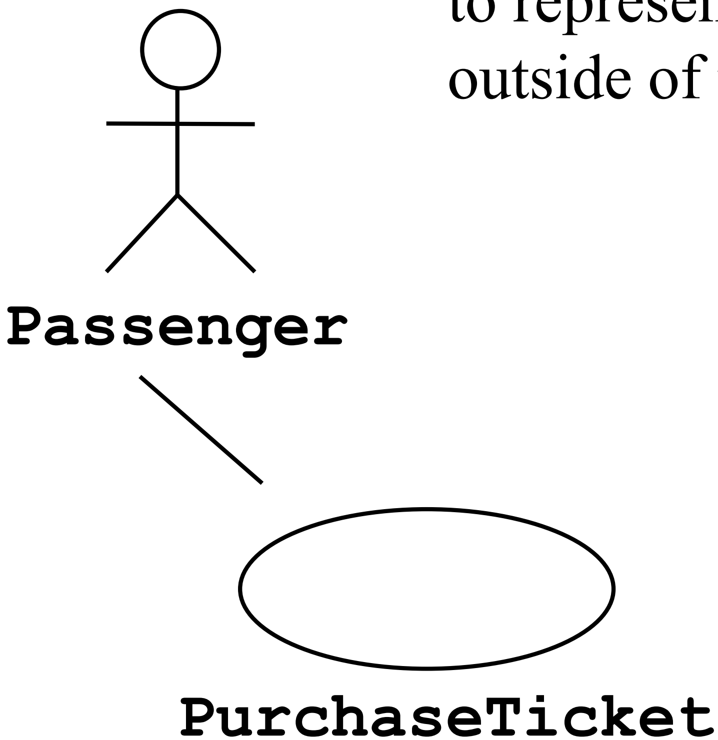
UML Basic Notation Summary

- UML provides a wide variety of notations for modeling many aspects of software systems
- Today we concentrated on a few notations:
  - **Functional model: Use case diagram**
  - **Object model: Class diagram**
  - **Dynamic model: Sequence diagrams, statechart**

What should be done first? Coding or Modeling?

- It all 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.**

UML Use Case Diagrams



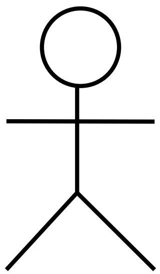
Used during requirements elicitation and analysis to represent external behavior ("visible from the outside of the system")

An ***actor*** represents a role; 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



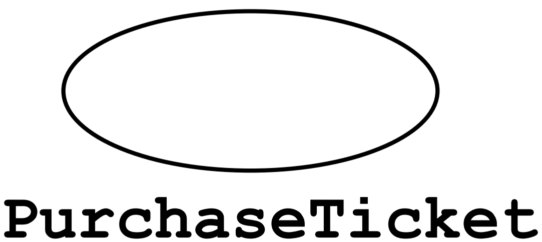
Passenger

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

Optional Description

Name

Use Case



- A ***class of functionality*** provided by the system
- 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 / quality requirements.**

Textual Use Case Description Example

1. Name: `Purchase ticket`

2. Participating actor: `Passenger`

3. Entry condition:

- `Passenger` stands in front of ticket distributor
- `Passenger` has sufficient money to purchase ticket

4. Exit condition:

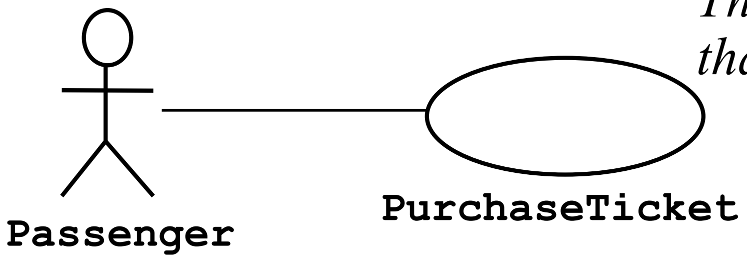
- `Passenger` has ticket

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 /quality requirements:

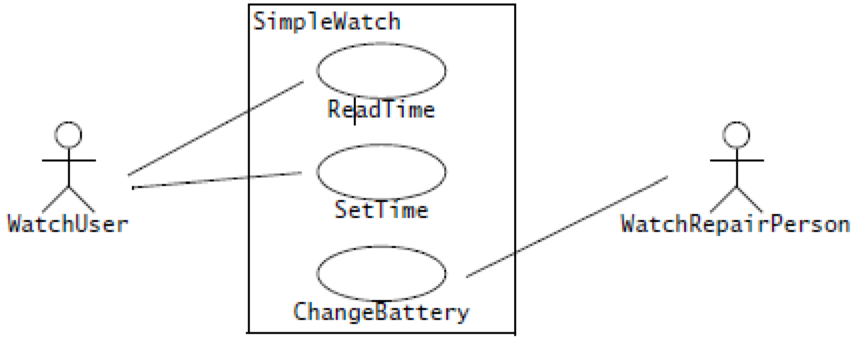
The transactions should take no longer than a minute



Use Case Diagrams

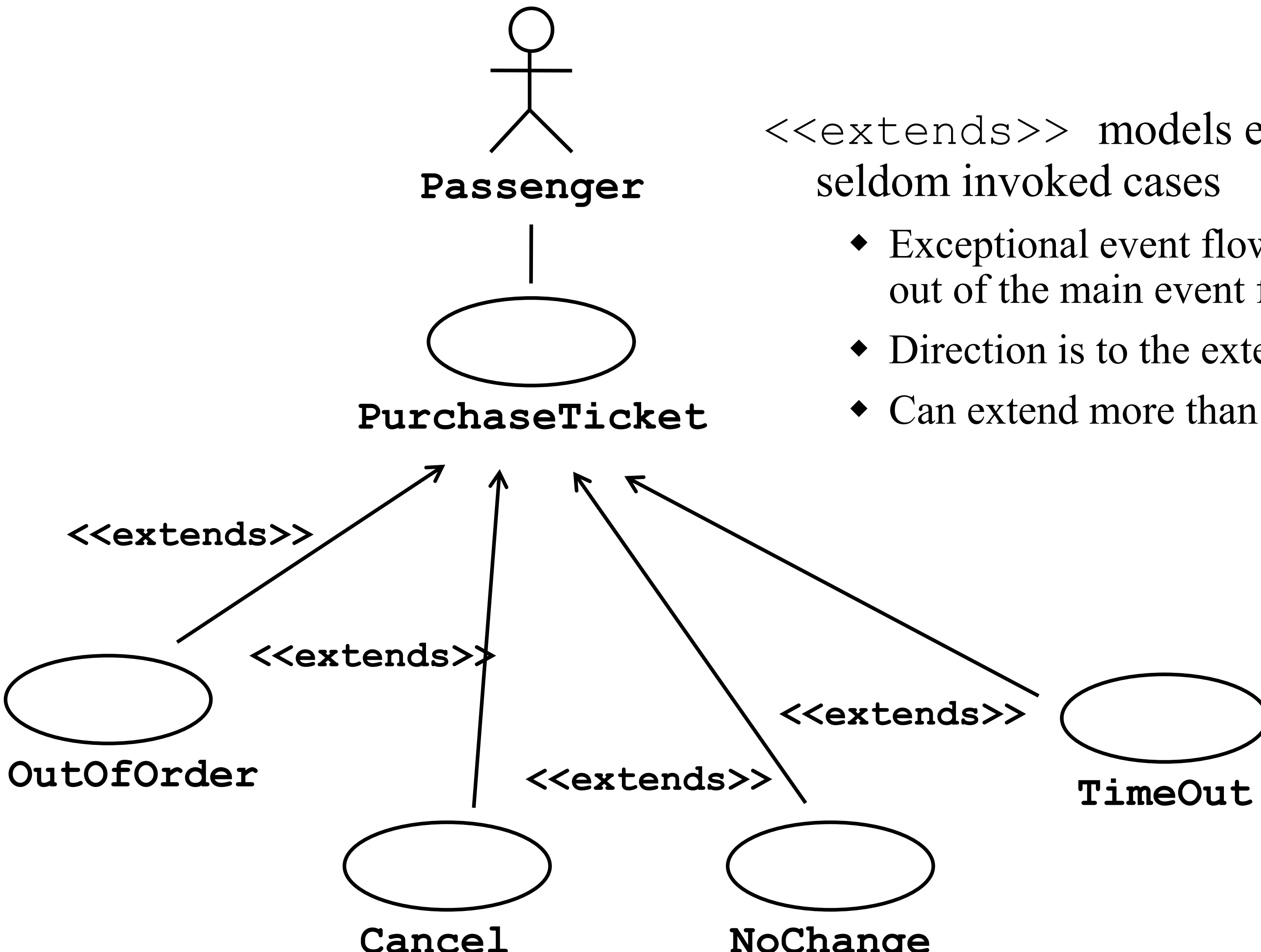
- Use cases are written in natural language. This enables developers to use them for communicating with the client and the users, who generally do not have an extensive knowledge of software engineering notations.
- Use case diagrams can include four types of relationships
  - **Communication**
  - **Inclusion**
  - **Extension**
  - **and inheritance.**

Use case - Communication



- **Communication relationships are depicted by a solid line between the actor and use case symbol**

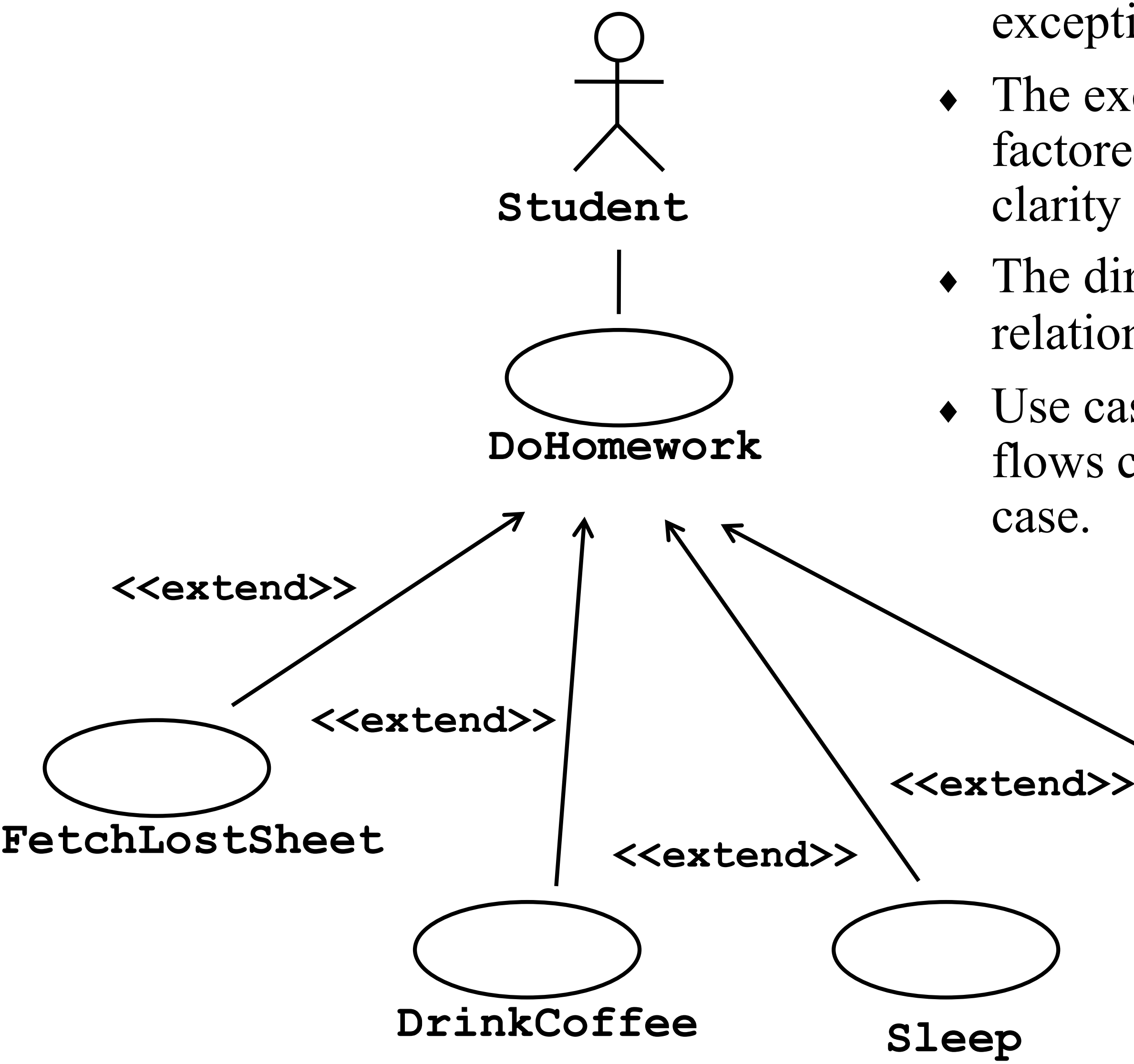
Uses Cases can be related: <<extends>>



`<<extends>>` models exceptional or seldom invoked cases

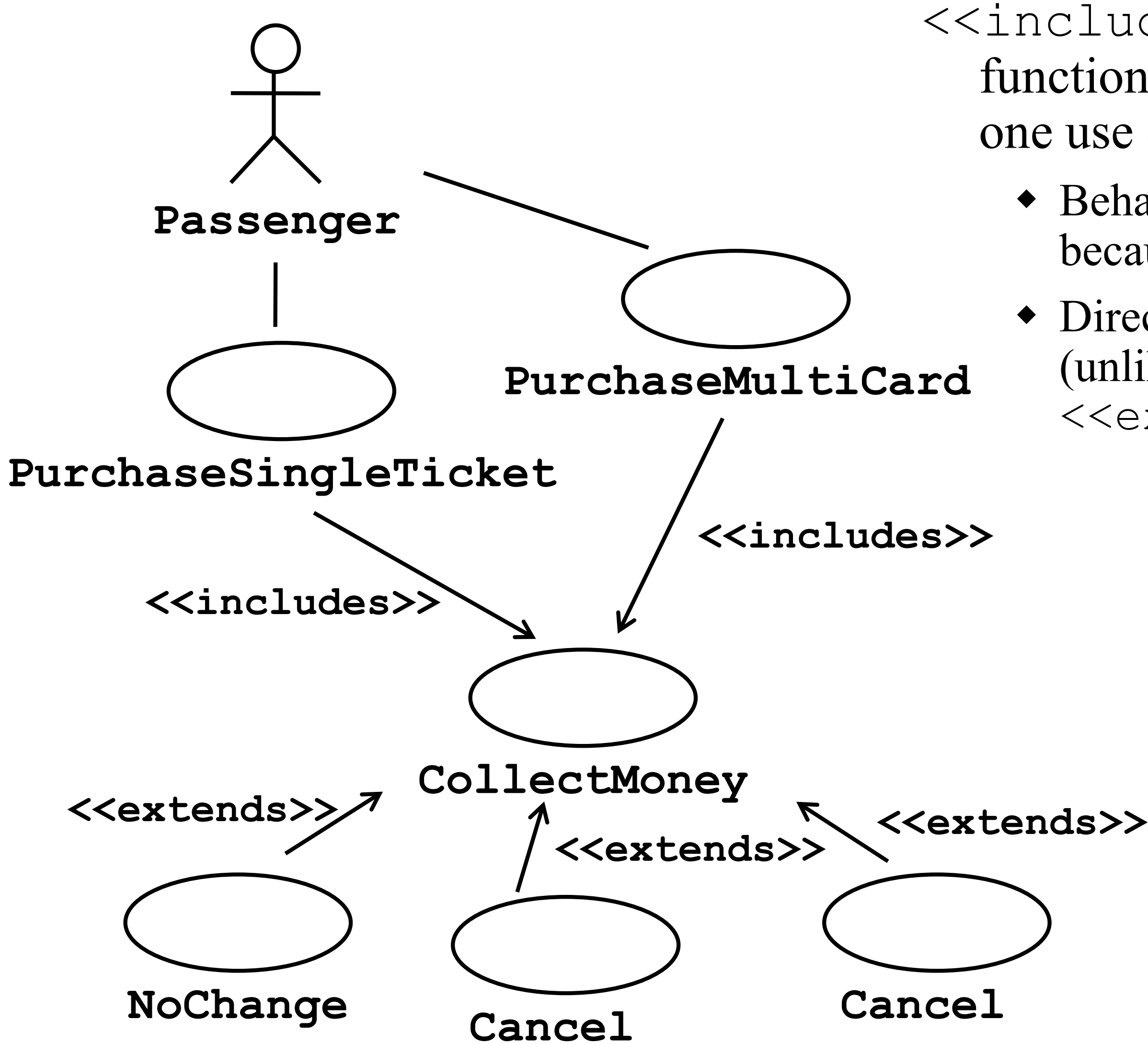
- Exceptional event flows are factored out of the main event flow for clarity
- Direction is to the extended use case
- Can extend more than one use case.

The <<extend>> Relationship



- `<<extend>>` 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 `<<extend>>` relationship is to the extended use case
- Use cases representing exceptional flows can extend more than one use case.

Uses Cases can be related: <<includes>>



`<<includes>>` represents common functionality needed in more than one use case

- Behavior factored out for reuse, not because it is an exception
- Direction is to the using use case (unlike the direction of the `<<extends>>` relationship).

Uses Cases : inheritance

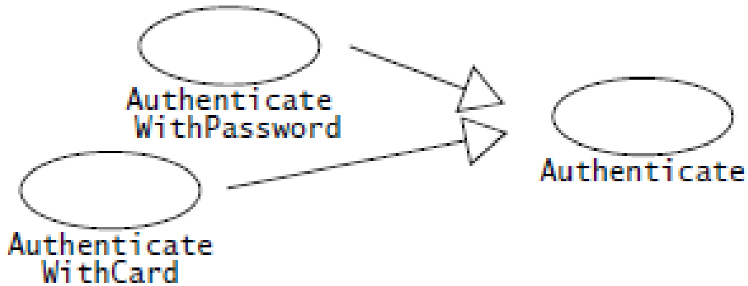
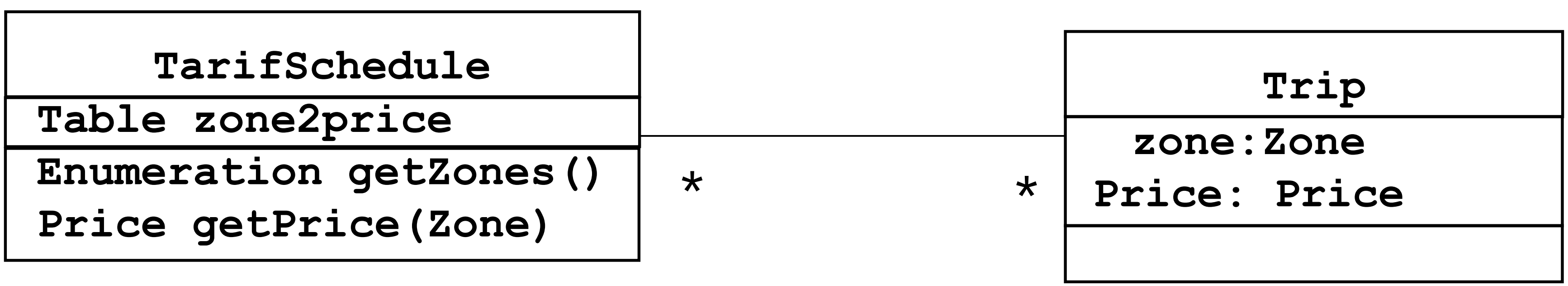


Figure 2-19 An example of an inheritance relationship (UML use case diagram). The `Authenticate` use case is a high-level use case describing, in general terms, the process of authentication. `AuthenticateWithPassword` and `AuthenticateWithCard` are two specializations of `Authenticate`.

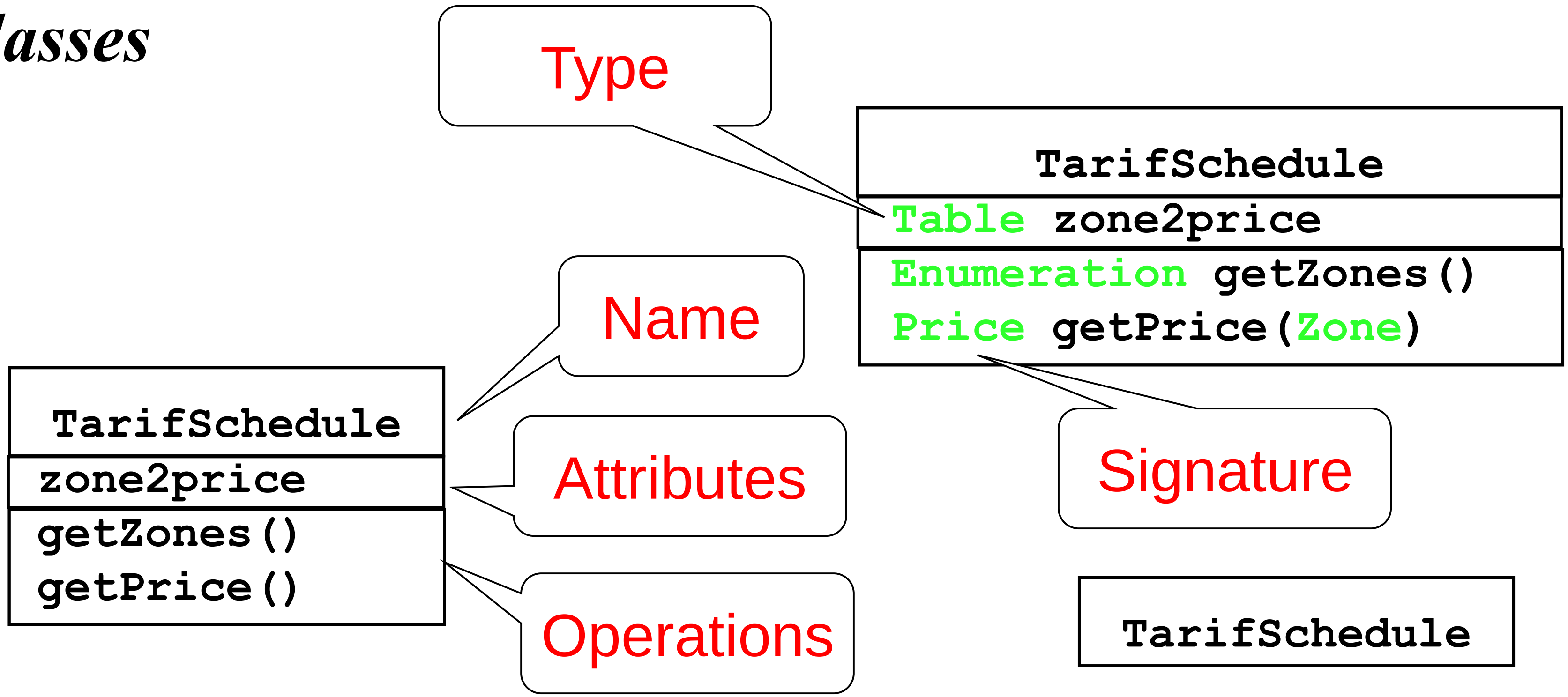
- One use case can specialize another more general one by adding more detail

Class Diagrams

- Represent the structure of the system
- Used
  - **during requirements analysis to model application domain concepts**
  - **during object design to specify the detailed behavior and attributes of classes.**

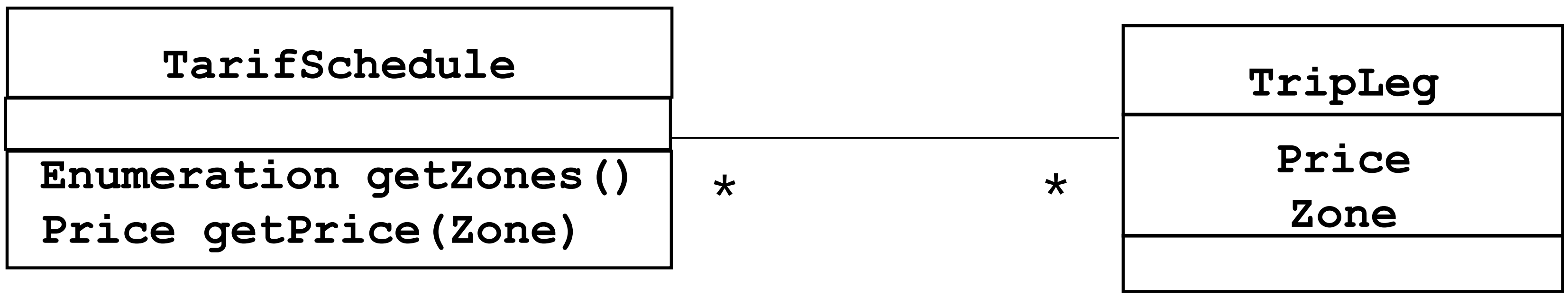


Classes



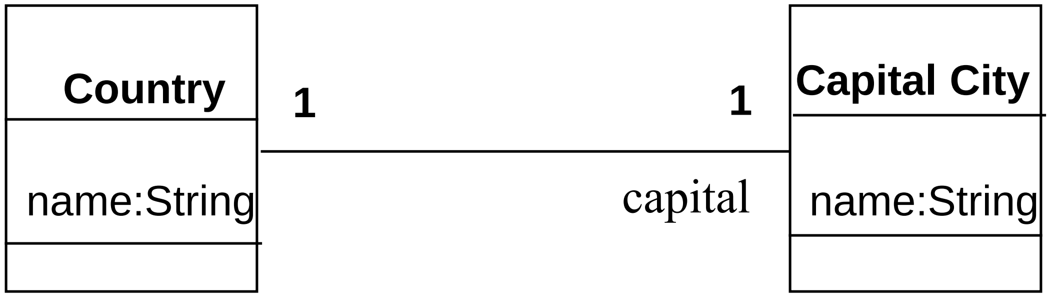
- A ***class*** represents a concept
- A class encapsulates state ***(attributes)*** and behavior ***(operations)***
  - Each attribute has a ***type***
  - Each operation has a ***signature***
- Class name is the only mandatory information

Associations

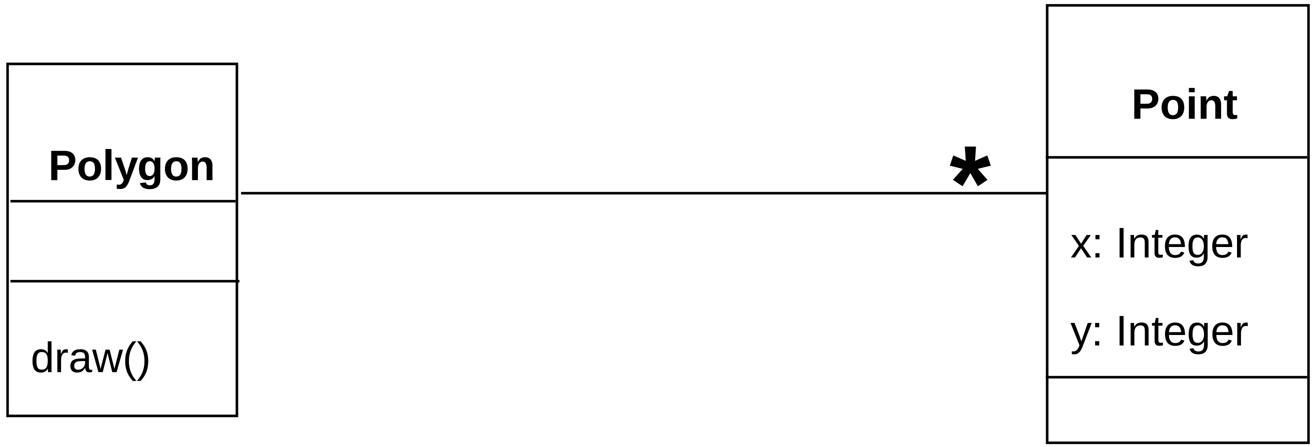


- Denote relationships between classes
- Multiplicity of an association end denotes how many objects the instance of a class can legitimately reference

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



1-to-1 association

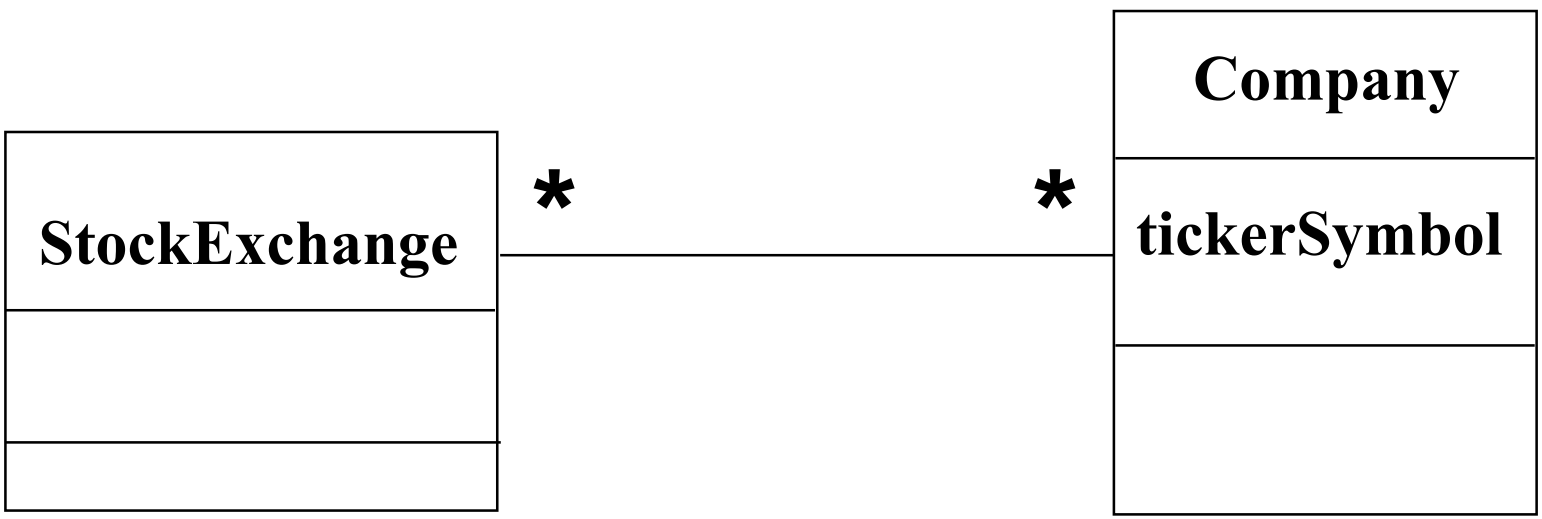


1-to-many association

Many-to-Many Associations

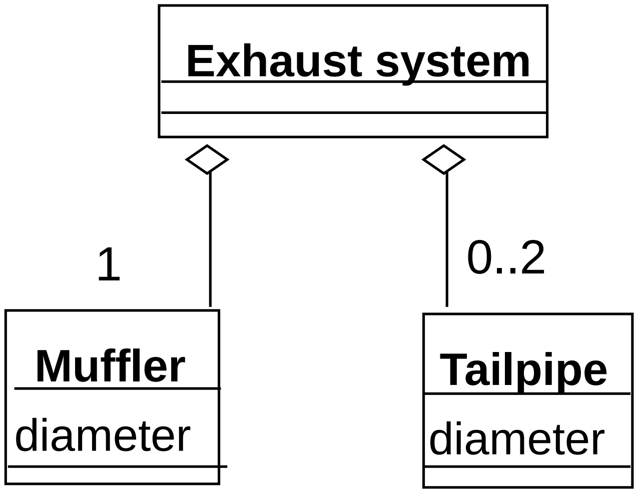
Problem Statement: *A stock exchange lists many companies. Each company is uniquely identified by a ticker symbol*

From Problem Statement to Object Model:

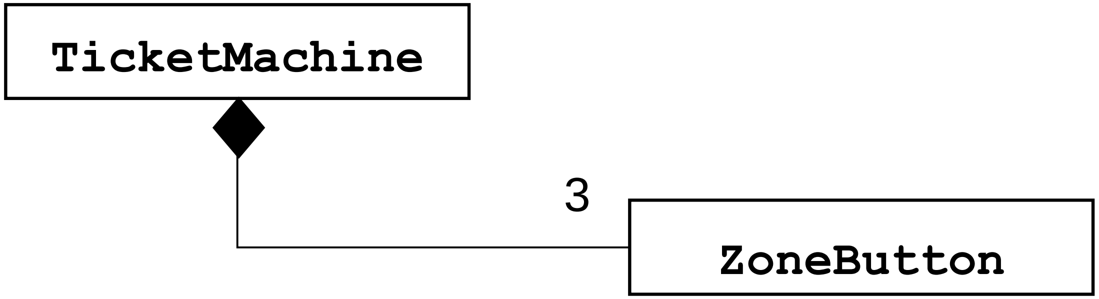


Aggregation vs. Composition

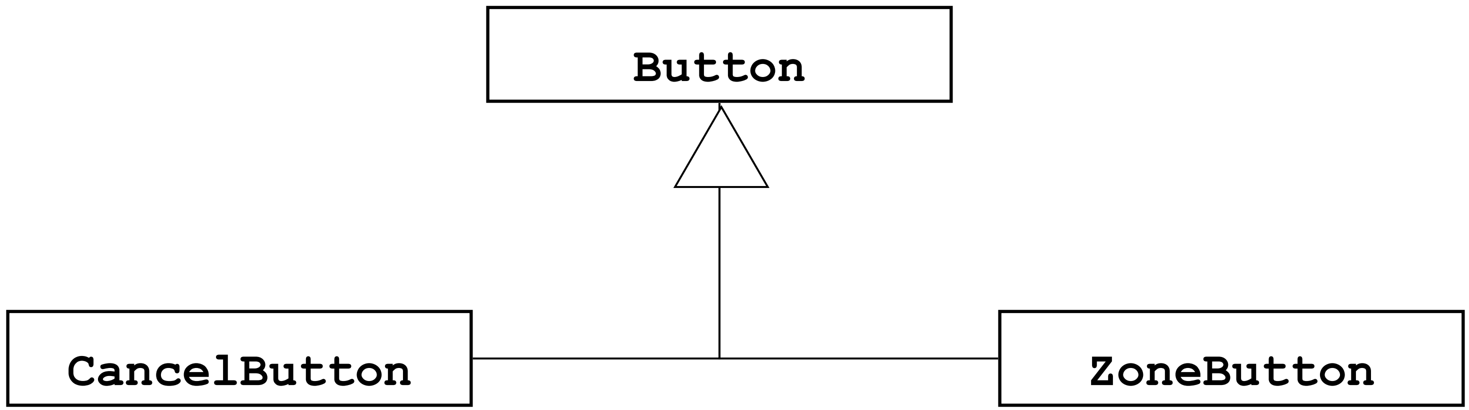
- ***Aggregation (diamond)***: special case of association denoting a "**consists-of**" hierarchy
- The *aggregate* is the parent class, the components are the children classes



- ***Composition (solid diamond)***: strong form of aggregation.
- The *life time of the component instances* is controlled by the aggregate. That is, the parts don't exist on their own ("the whole controls/destroys the parts").



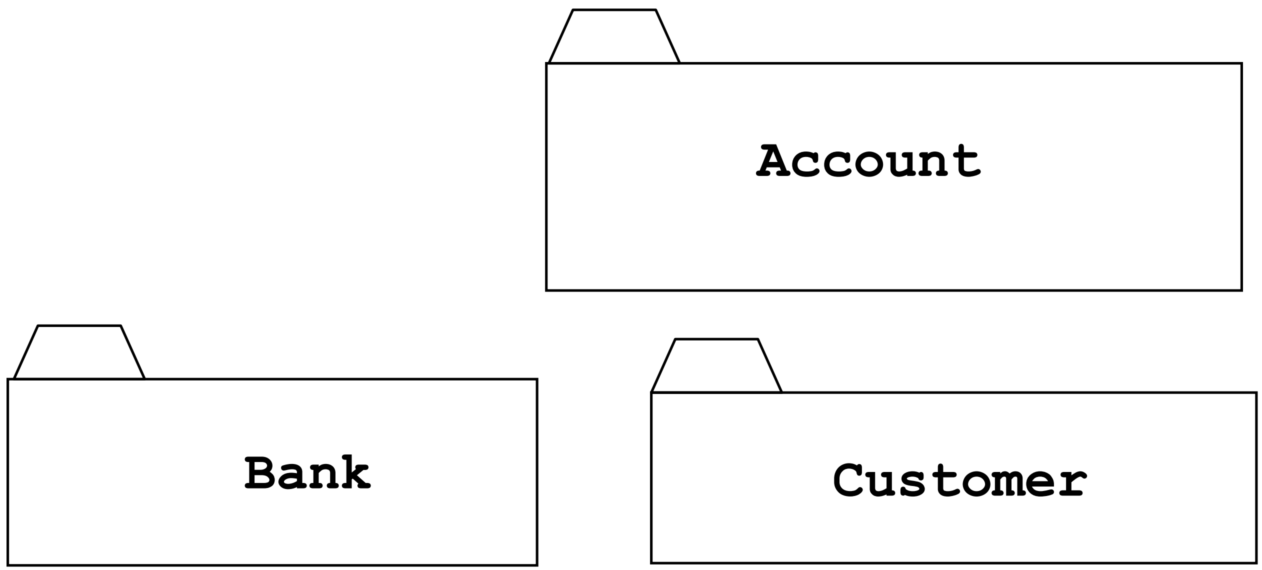
Inheritance



- Another special case of an association denoting a "**kind-of**" hierarchy
- Simplifies analysis model by introducing a taxonomy
- Children classes inherit attributes and operations of parent class

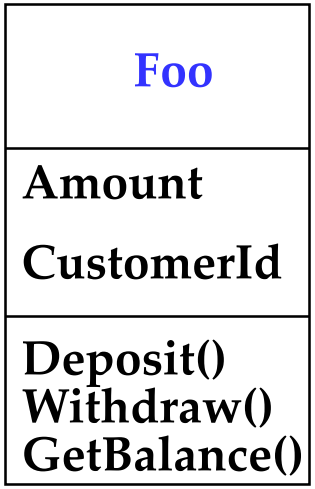
Packages

- Organize UML models to increase their readability
- 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

~~“Dada”~~

Amount

CustomerId

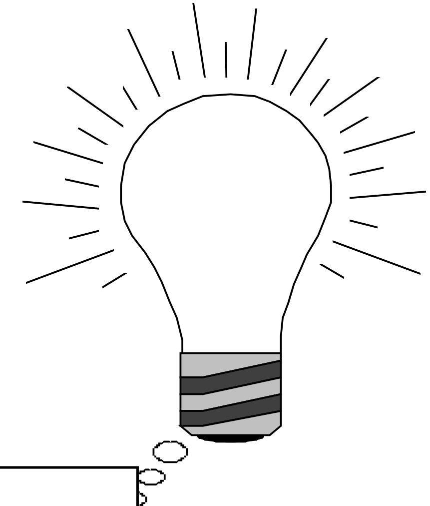
Deposit()
Withdraw()
GetBalance()

~~Foo~~

Amount

CustomerId

Deposit()
Withdraw()
GetBalance()



Account

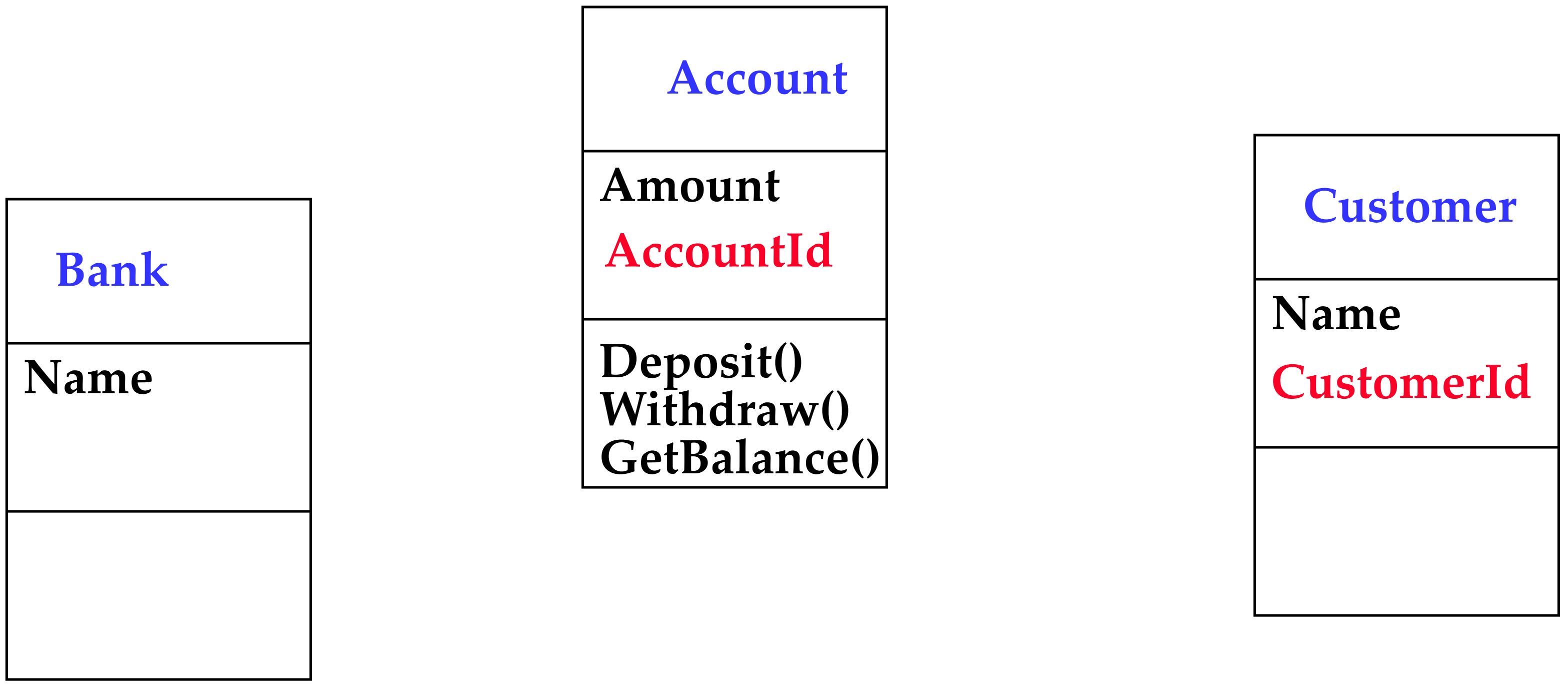
Amount

CustomerId

Deposit()
Withdraw()
GetBalance()

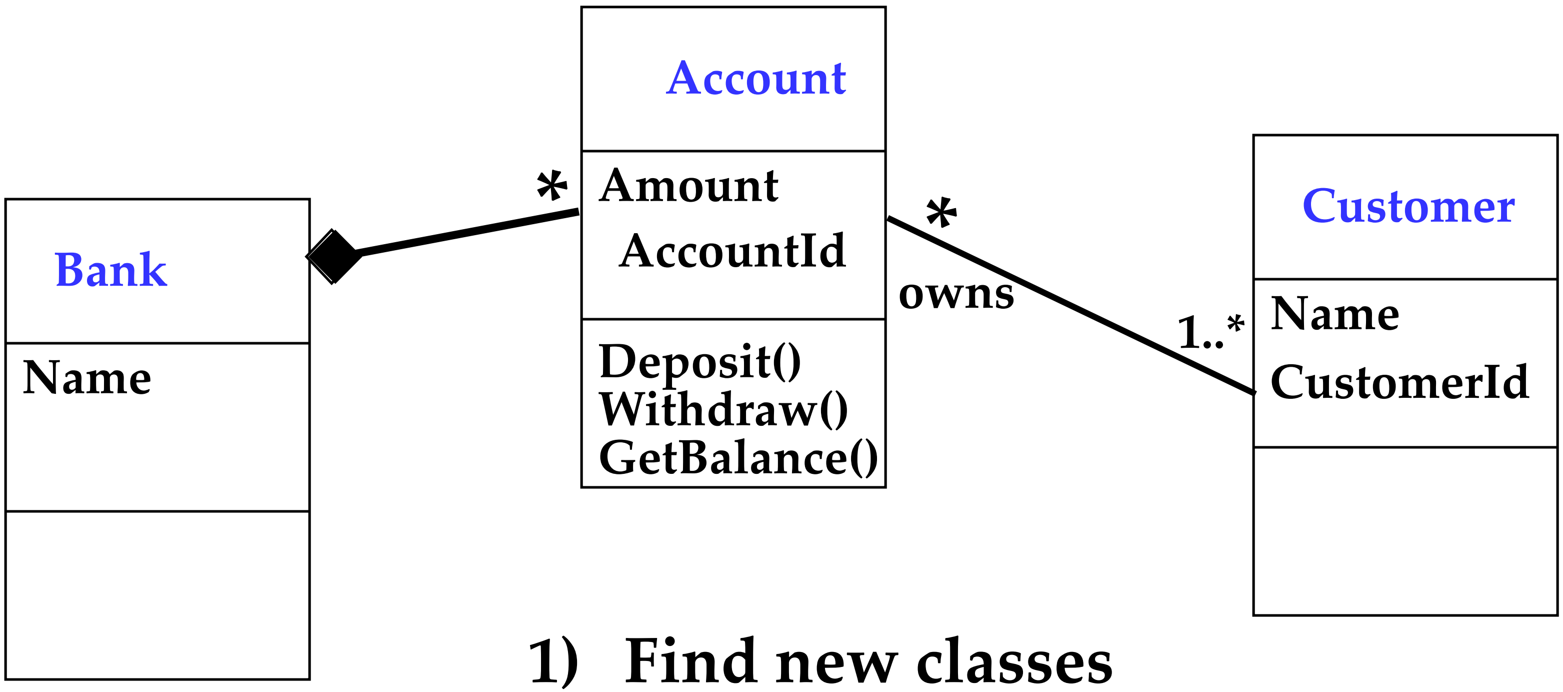
Is Foo the right name?

Object Modeling in Practice: More classes



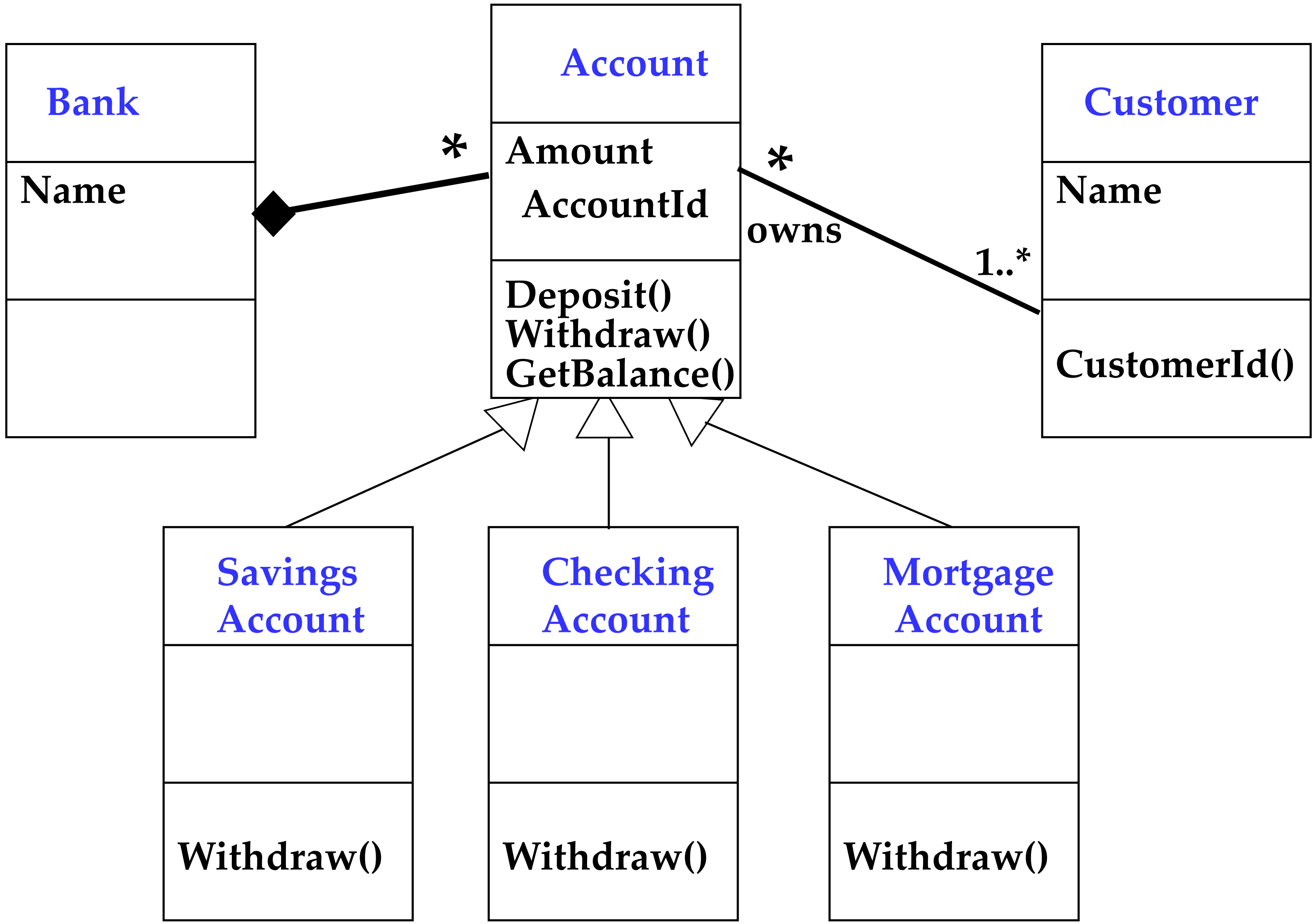
1) Find new classes
2) Review names, attributes and methods

Object Modeling in Practice: Associations

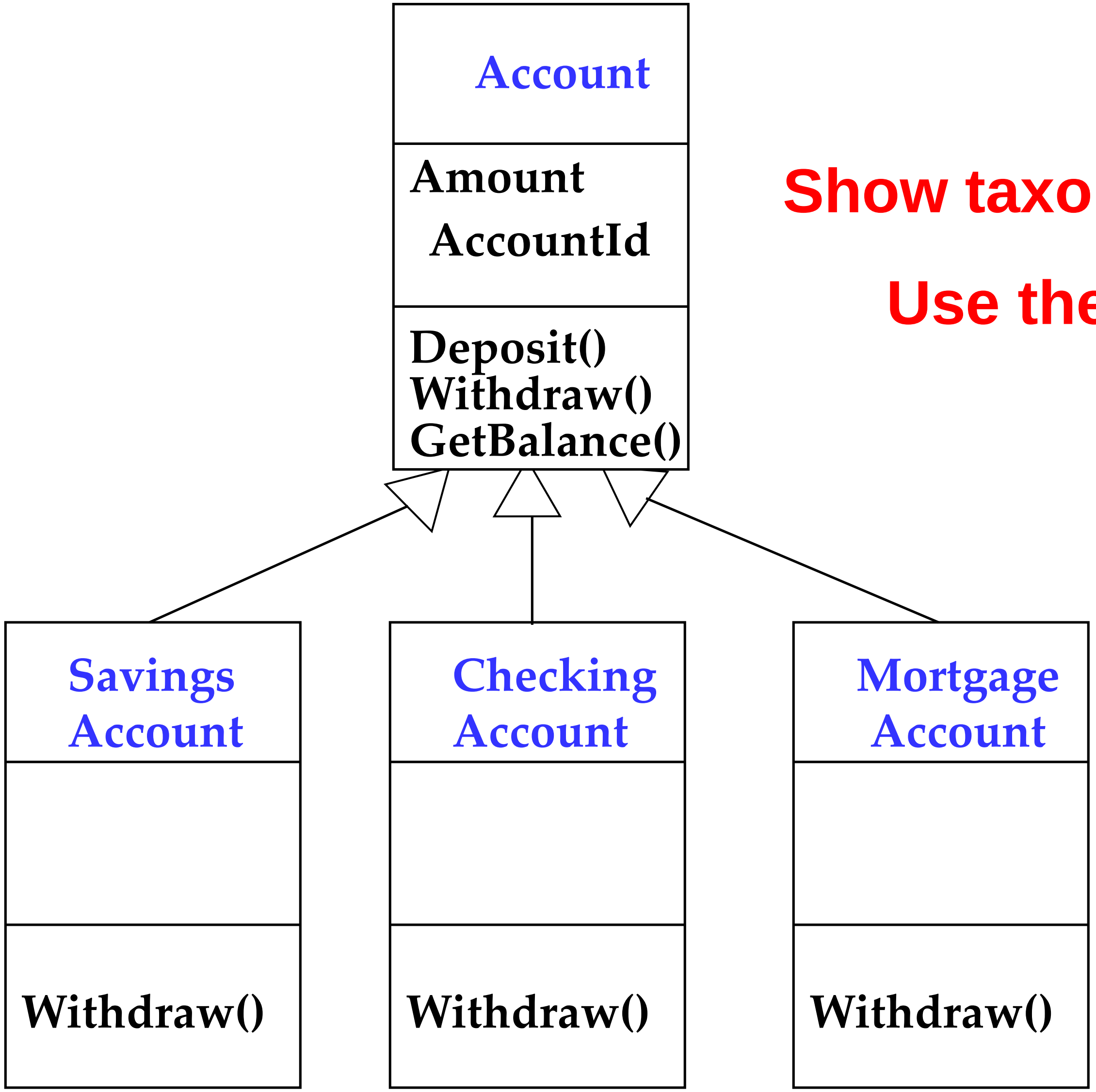


1) Find new classes
2) Review names, attributes and methods
3) Find Associations between Classes
4) Label generic associations
5) Determine multiplicity of associations
6) Review associations

Practice Object Modeling: Find Taxonomies



Practice Object Modeling: Simplify, Organize

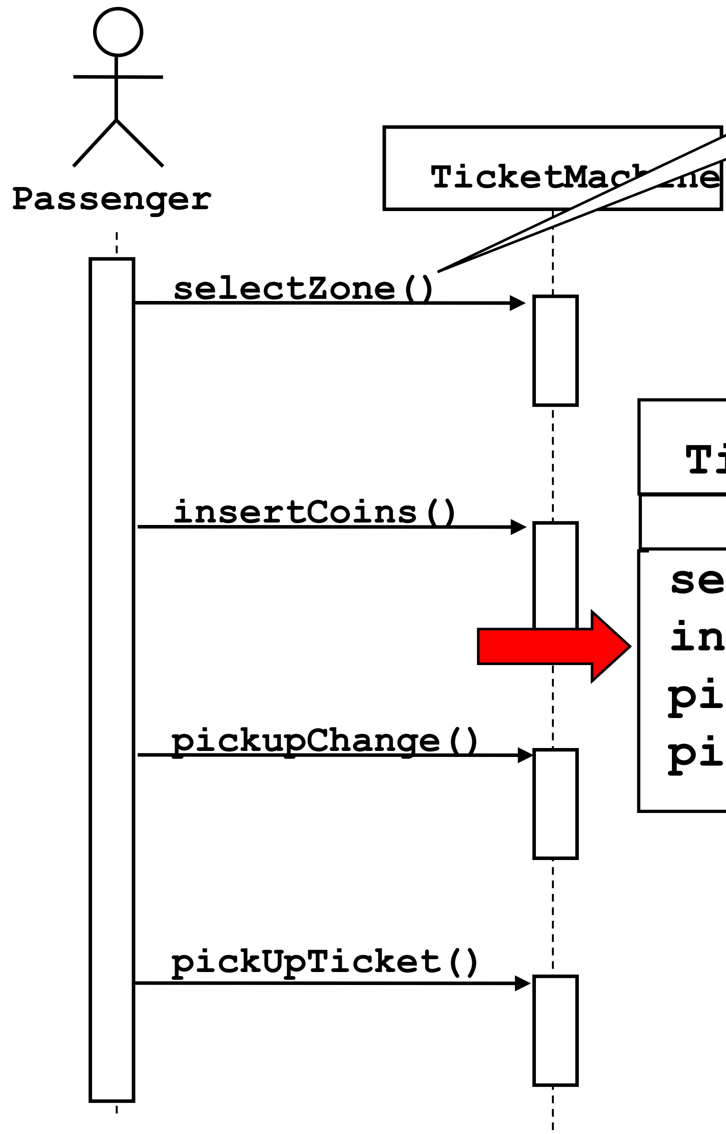


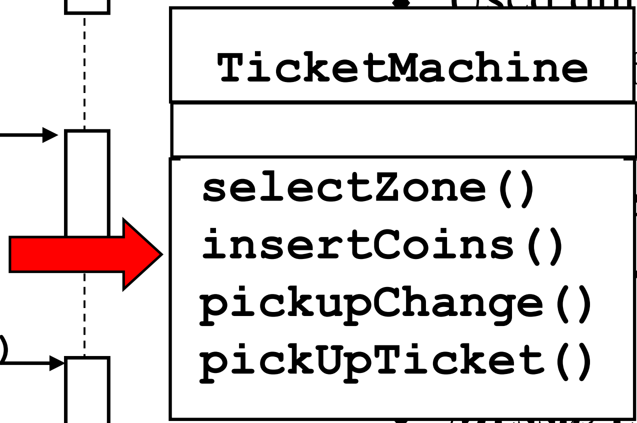
Show taxonomies separately

Use the 7+-2 heuristic

Sequence Diagrams

Focus on control flow

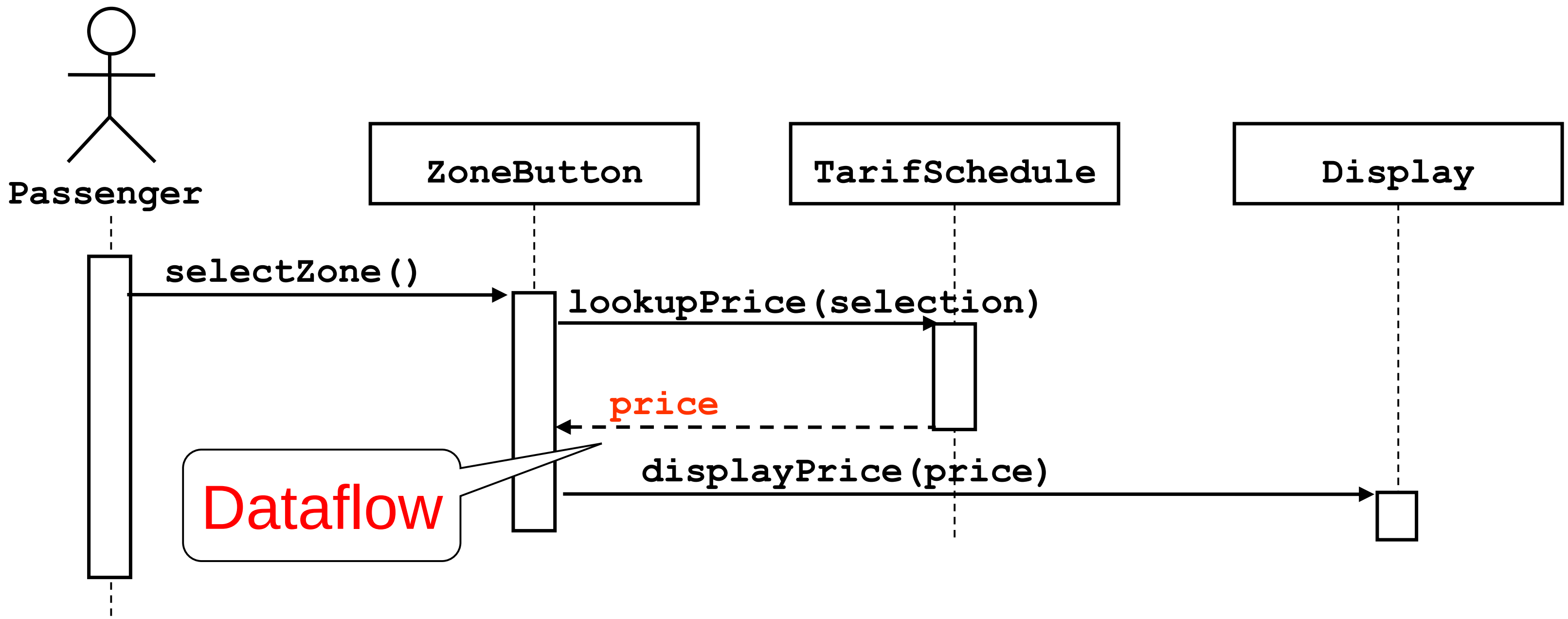




- Used during analysis
  - **to refine use case descriptions**
  - **to find additional objects ("participating objects")**
- Used during system design
  - **...fine su...system interfaces**
- ...*s* are repr...
  ...es. *Actors* b...
- ... are repres...
- *Messages* are represented by arrows

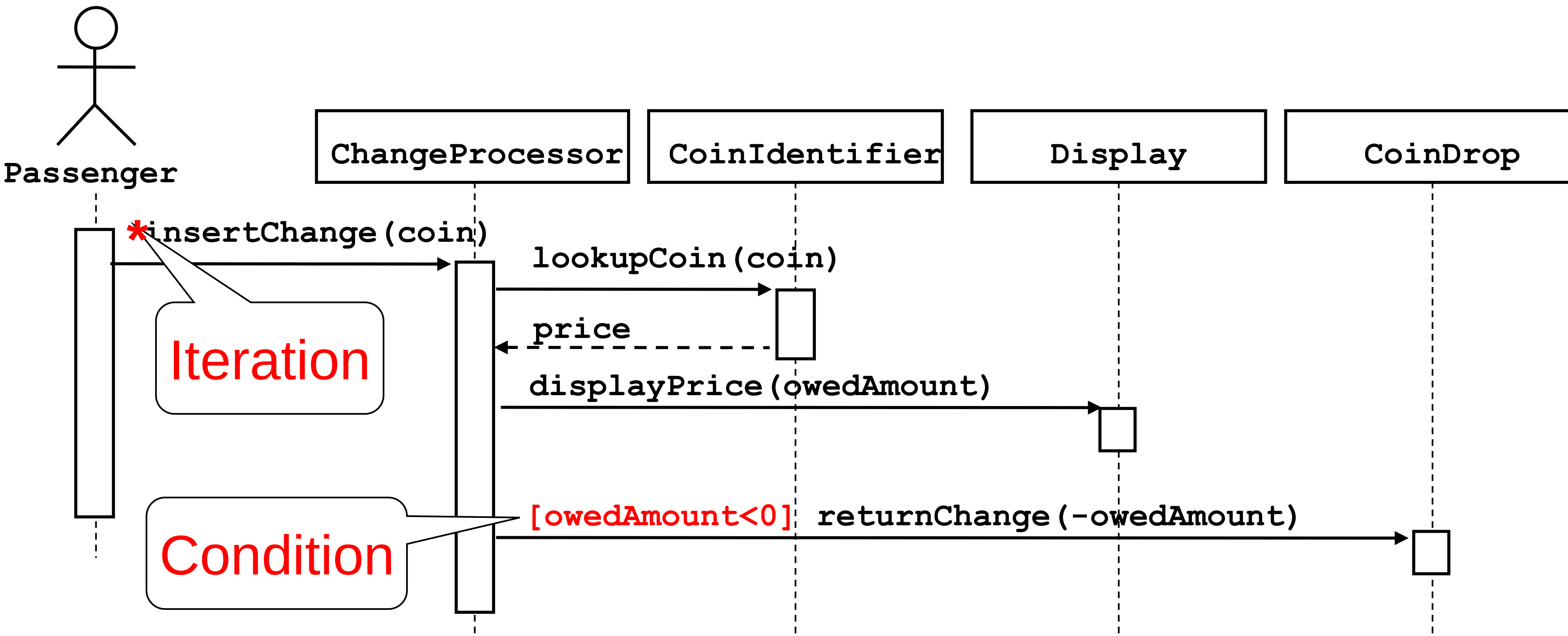
Messages are Operations on participating Object

Sequence Diagrams can also model the Flow of Data



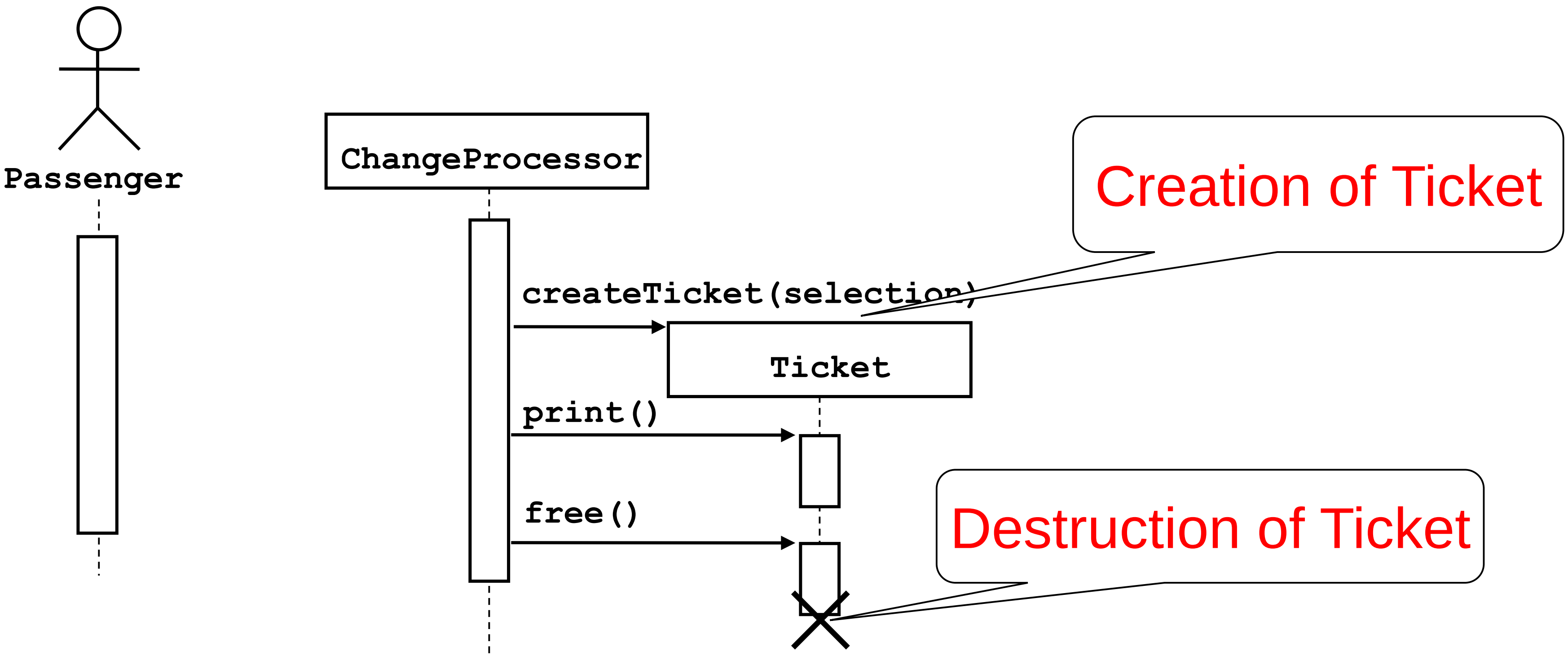
- Source of an arrow indicates activation which sent the message
- Horizontal dashed arrows indicate data flow, for example return results from a message

Sequence Diagrams: Iteration & Condition



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

Creation and Destruction



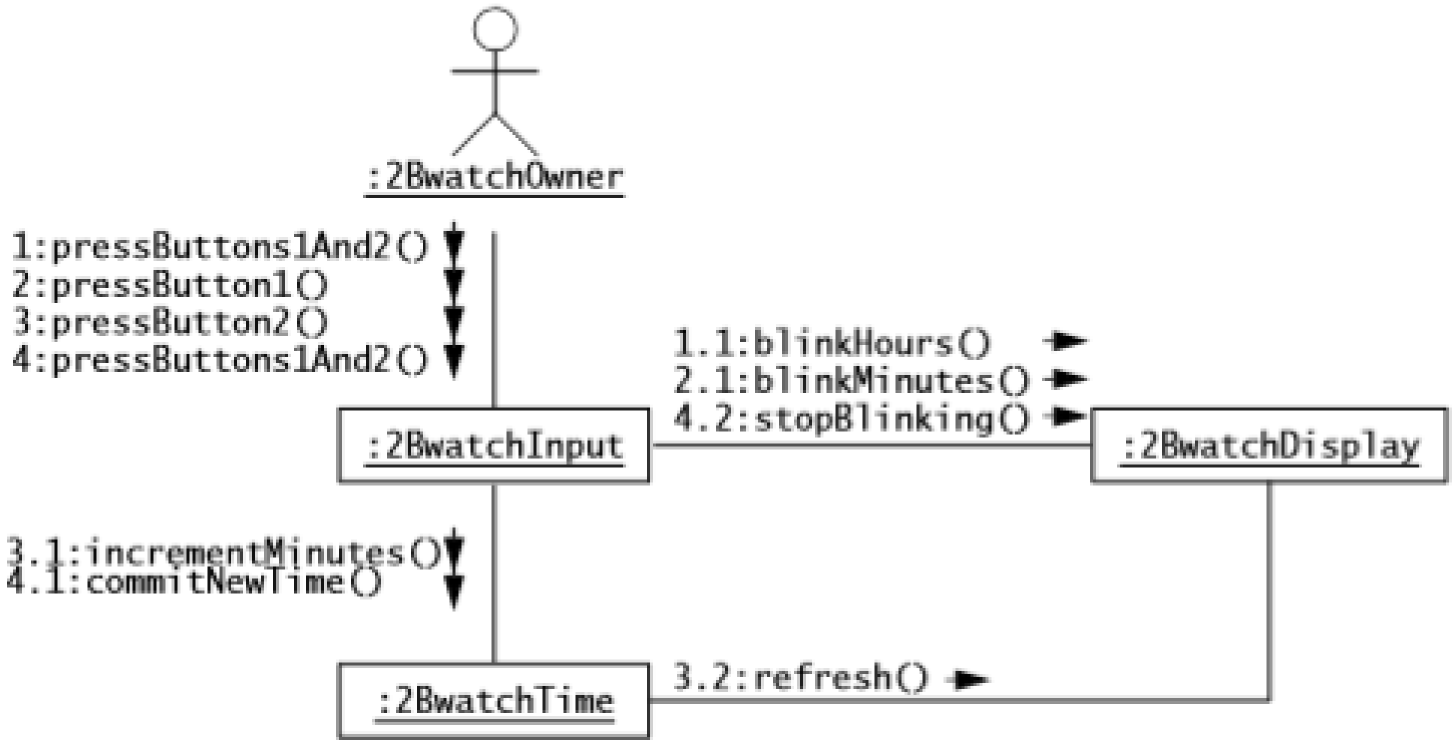
- Creation denoted by a message arrow pointing to object
- Destruction denoted by an X mark at the end of the destruction activation
  - **In garbage collection environments, destruction can be used to denote the end of the useful life of an object.**

Sequence Diagram Properties

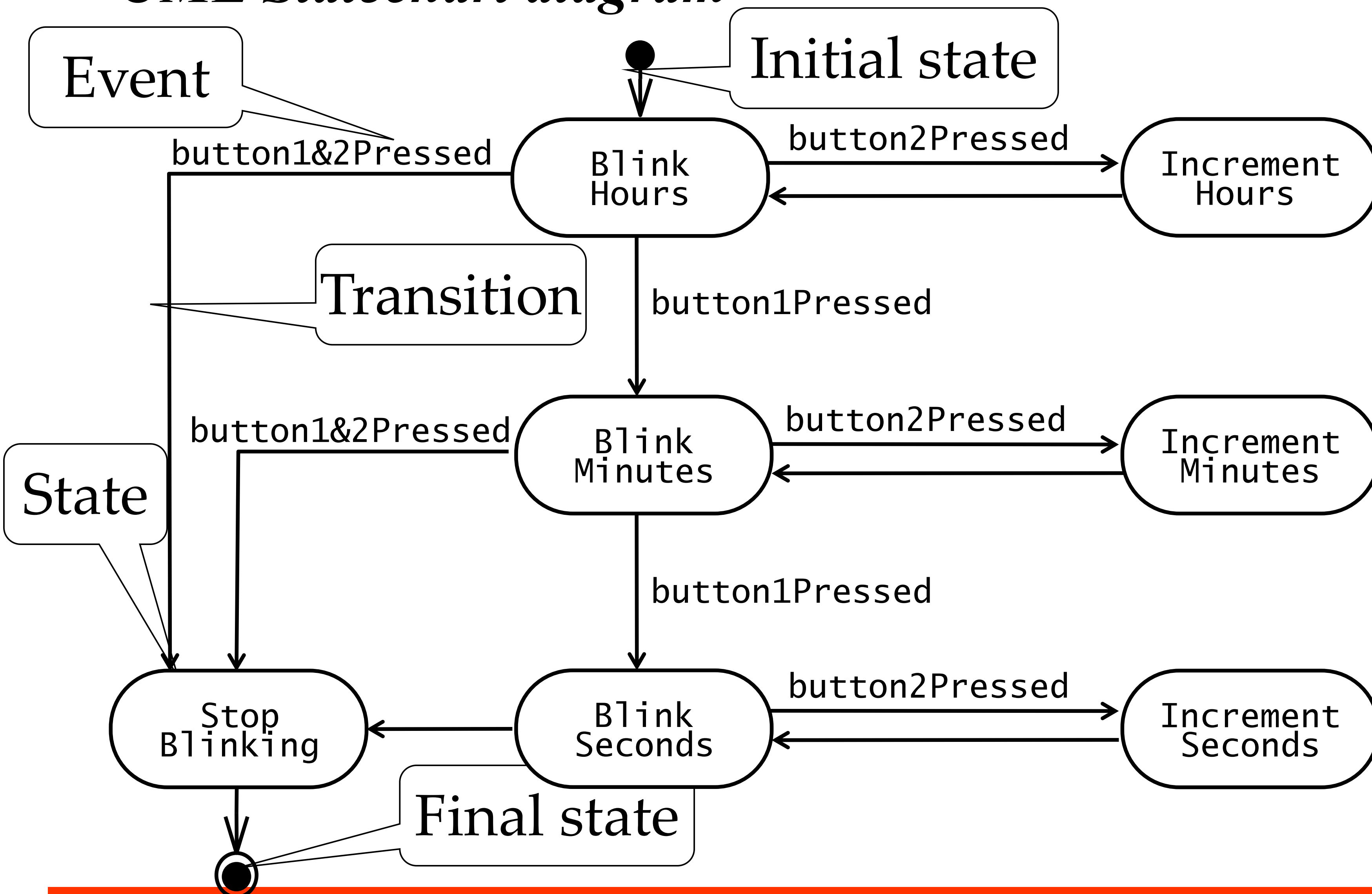
- ***Behavior in terms of interactions***
- ***How objects interact to get the job done***
- Useful to identify or find missing objects
- Time consuming to build, but worth the investment
- Complement the class diagrams (which represent static structure)

Communication Diagram

♦ Depicts the same information as sequence diagrams

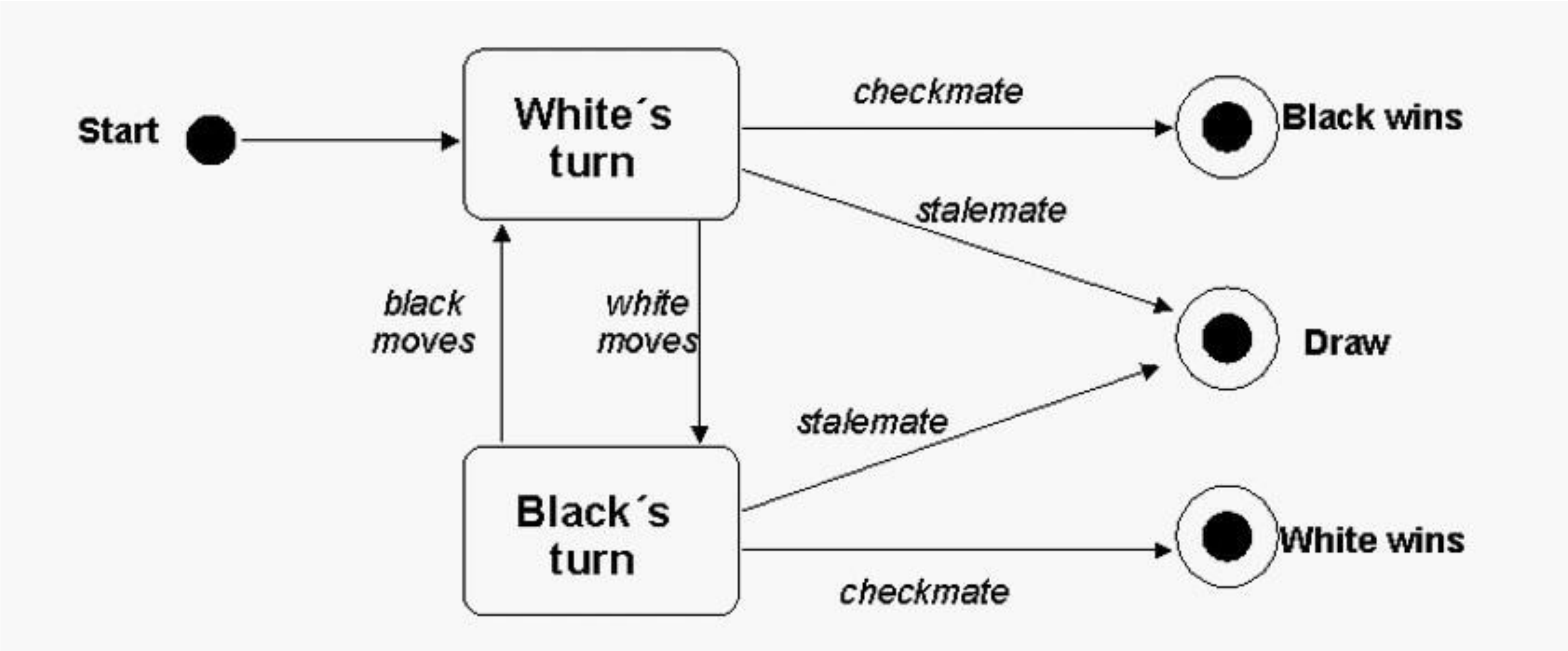


UML Statechart diagram



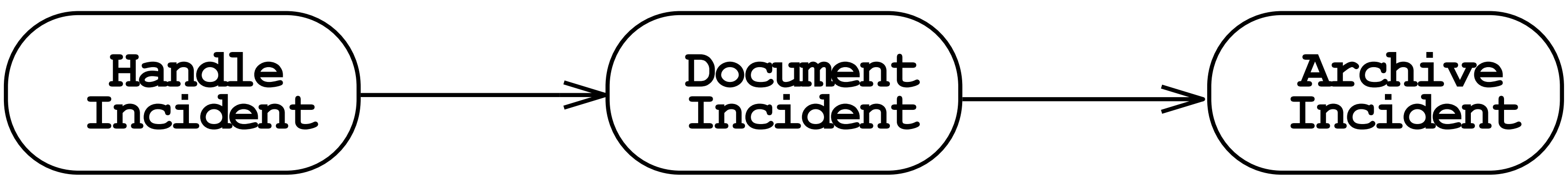
Represents behavior of *a single object* with interesting dynamic behavior.

UML state diagram - Chess

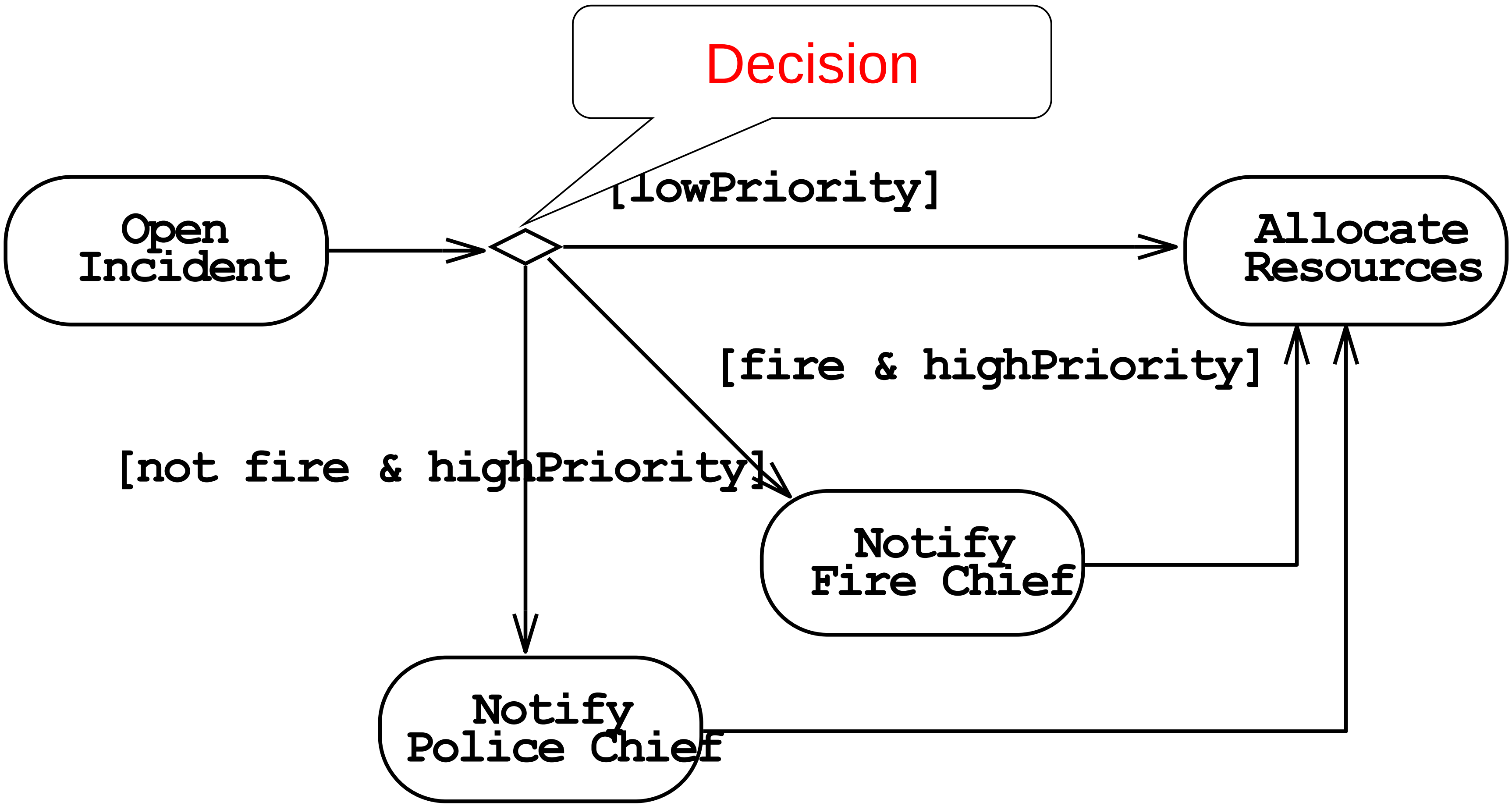


Activity Diagrams

- Useful to depict the workflow in a system (analogous to flowcharts)
- Activities are of ***the system***, not the user!
- Example from: FRIEND (First Responder Interactive Emergency Navigational Database)

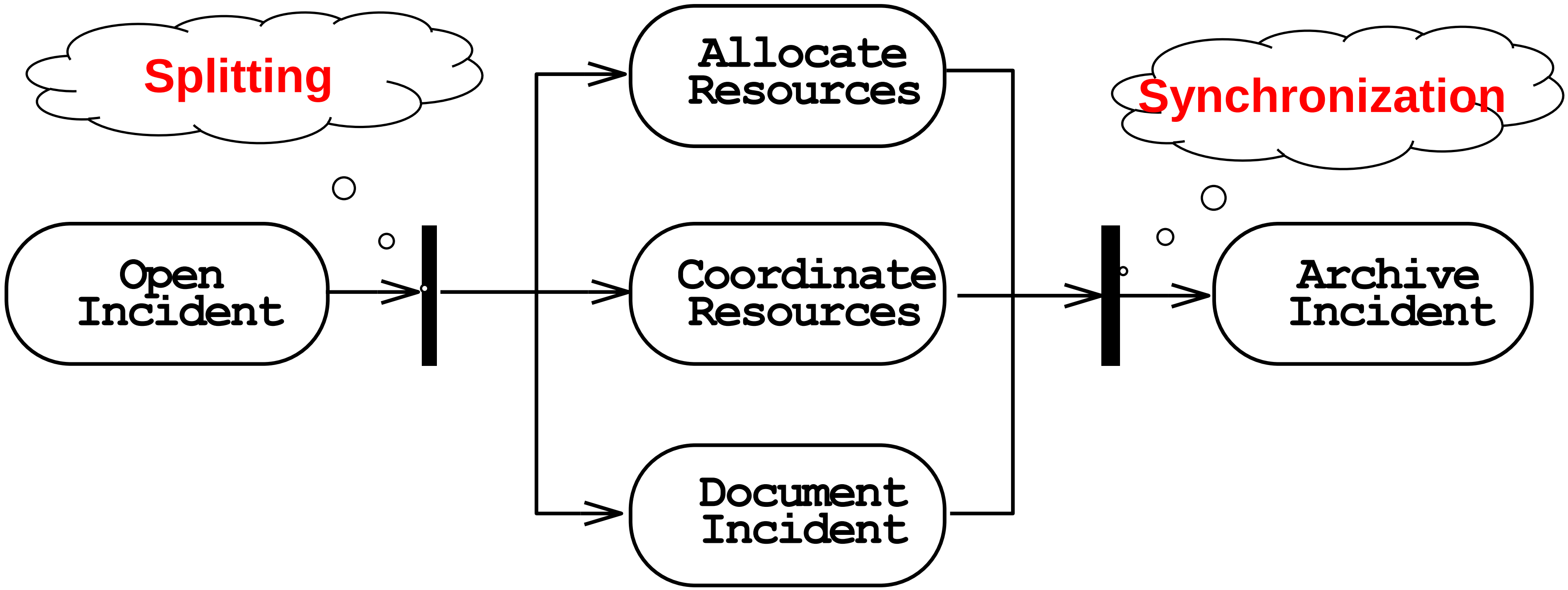


Activity Diagrams allow to model Decisions



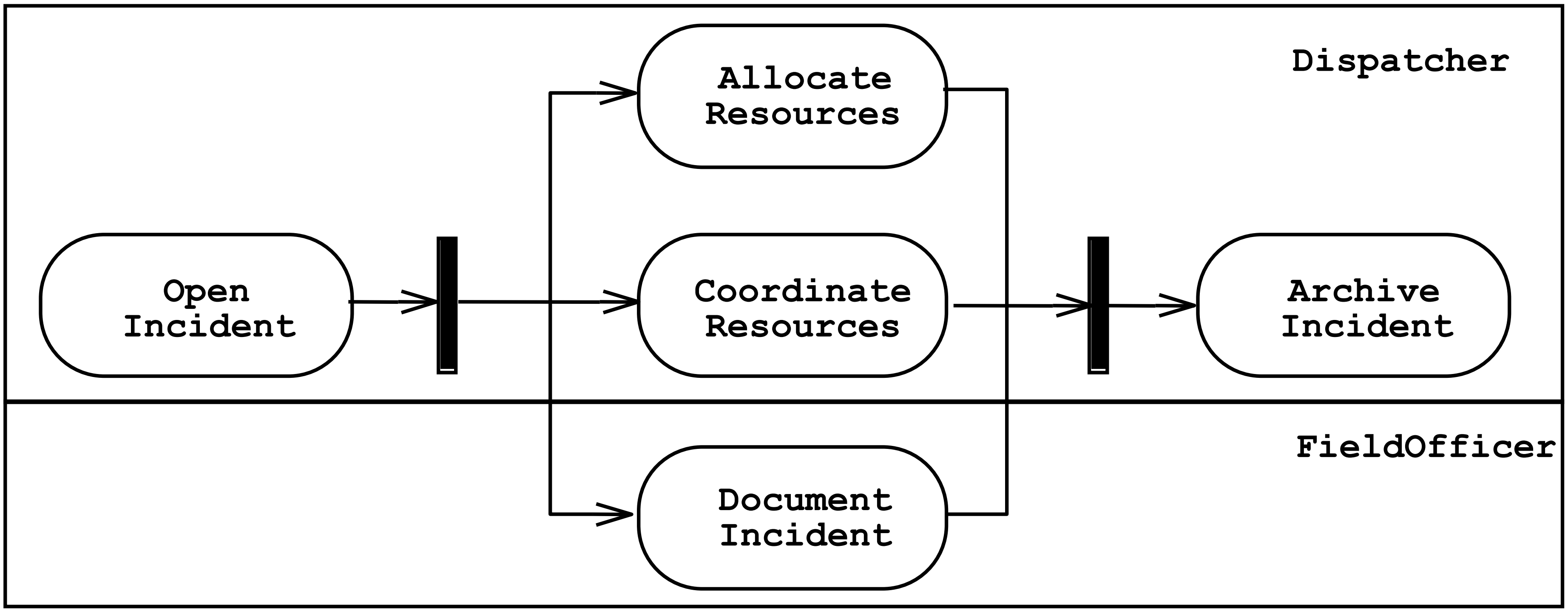
Activity Diagrams can model Concurrency

- Synchronization of multiple ***independent*** activities
- Splitting the flow of control into multiple threads



Activity Diagrams: grouping

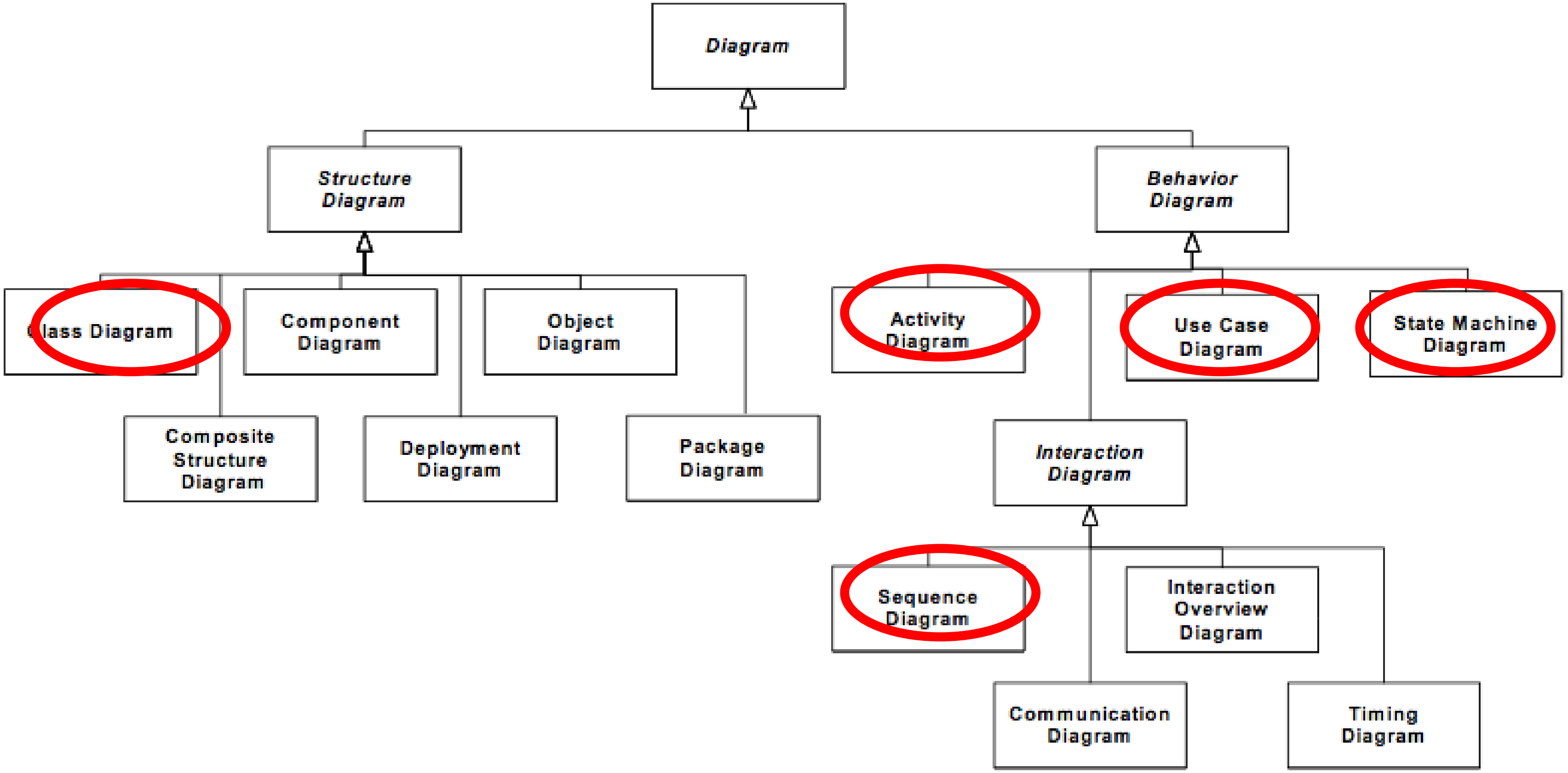
- May be grouped into ***swimlanes*** to denote object or subsystem that implements the activities.



UML Summary

- Provides a wide variety of notations for representing many ***aspects*** of software development
  - **Powerful, but complex**
- UML as a programming language
  - **Can be misused to generate unreadable models**
  - **Can be misunderstood when using too many exotic features**
- So far:
  - **Functional model: Use case diagram**
  - **Object model: Class diagram**
  - **Dynamic model: Sequence, State, and Activity diagrams**

Another view on UML Diagrams



Midterm Exam

- Nov 5, 17:40

OO Analysis w/ UML