Object Oriented Programming:
UML
Modeling with Classes

UML diagrams

Use case diagrams

- Describe user tasks and points of contact with the system

Class diagrams

- describe classes and their relationships

Sequence diagrams

- show the behaviour of systems as interactions between objects

State diagrams and activity diagrams

- show how systems behave internally

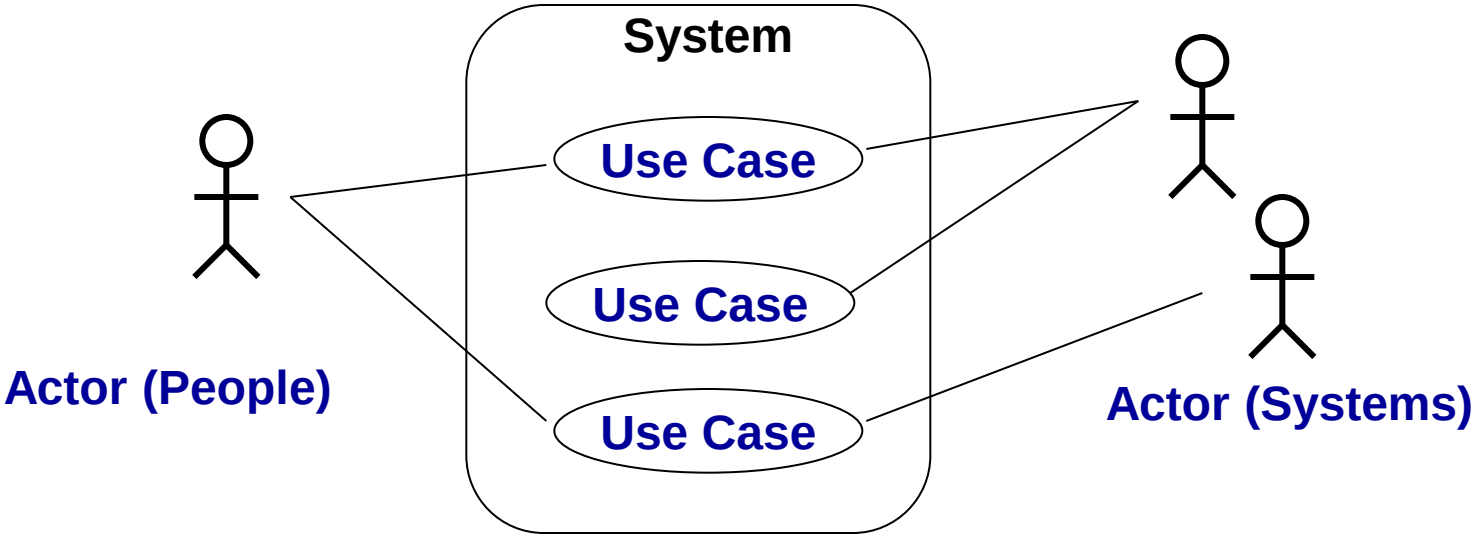
Component and deployment diagrams

- “big picture” of how the components of a system are related

Usecase diagram

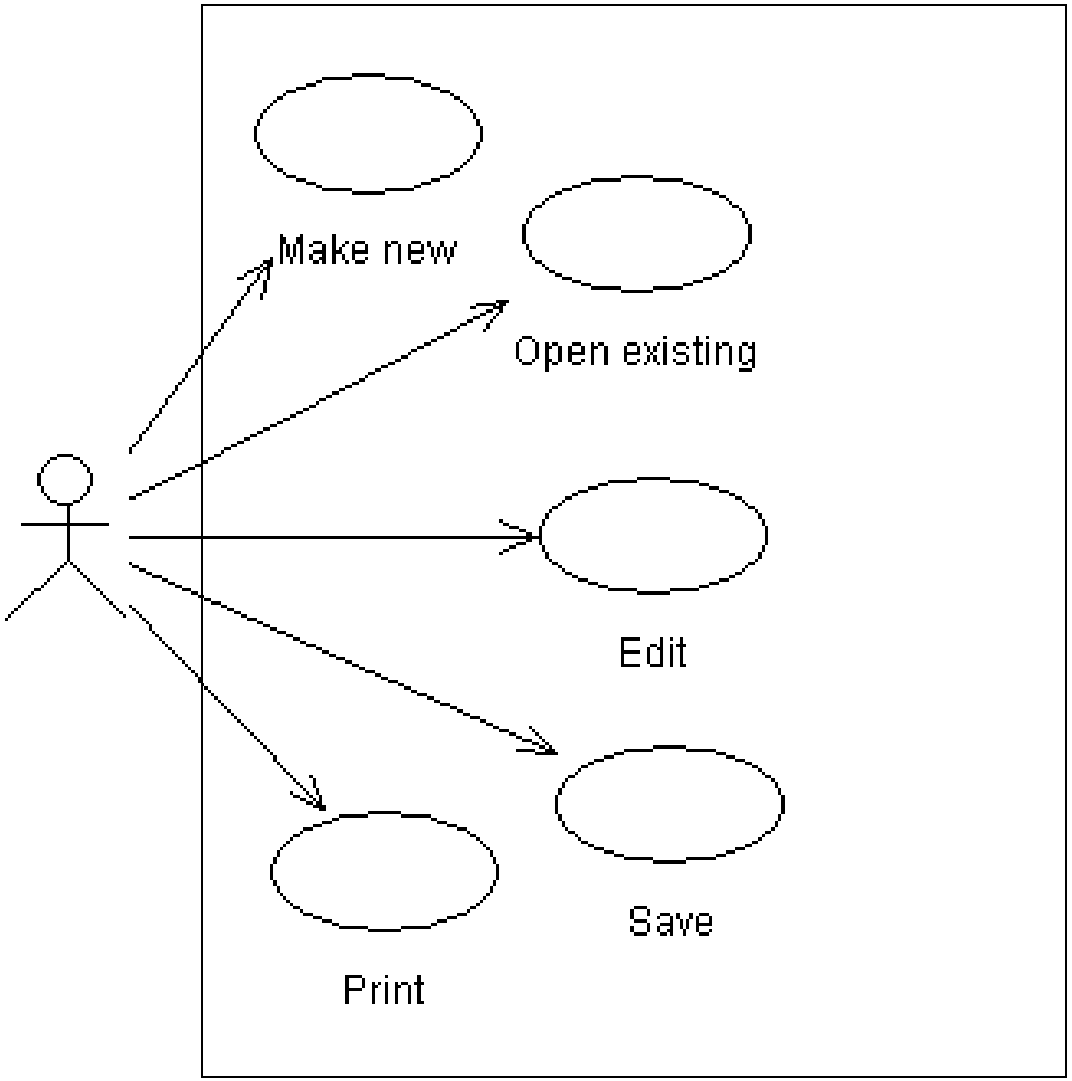
Use Cases

- Two types of Actors: Users and System administrators



Use case examples

(use cases for powerpoint.)

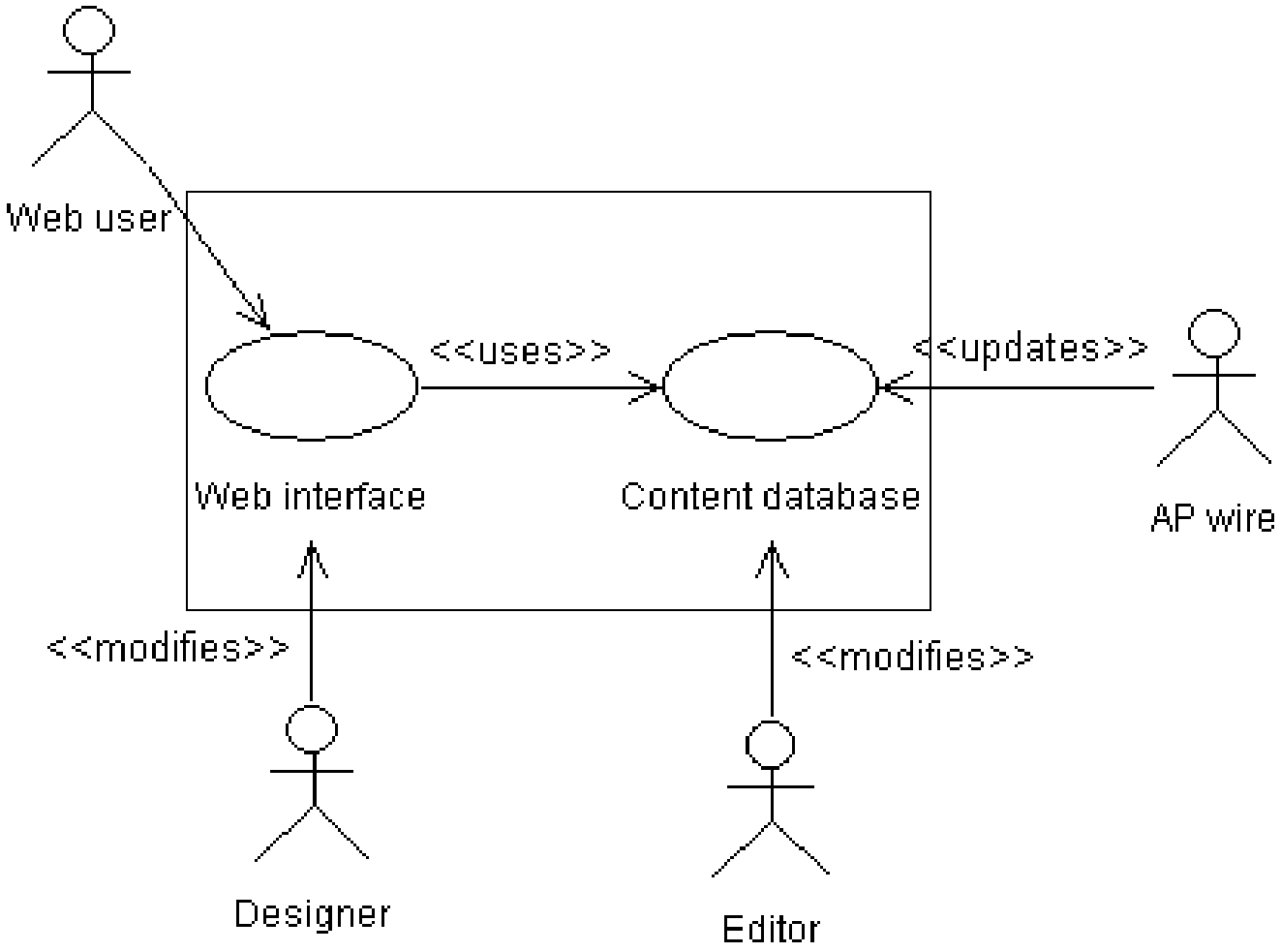


About the last example...

- Gives a view of powerpoint.
- focusses your attention to the key features

Use case examples

(Relationships in a news web site.)



About the last example...

- The last is more complicated and realistic use case diagram. It captures several key use cases for the system.
- Note the multiple actors. In particular, 'AP wire' is an actor, with an important interaction with the system, but is not a person (or even a computer system, necessarily).
- The notes between << >> marks are *stereotypes:* make the diagram more informative.

Usecase diagram

Give a Usecase diagram for an ATM machine:

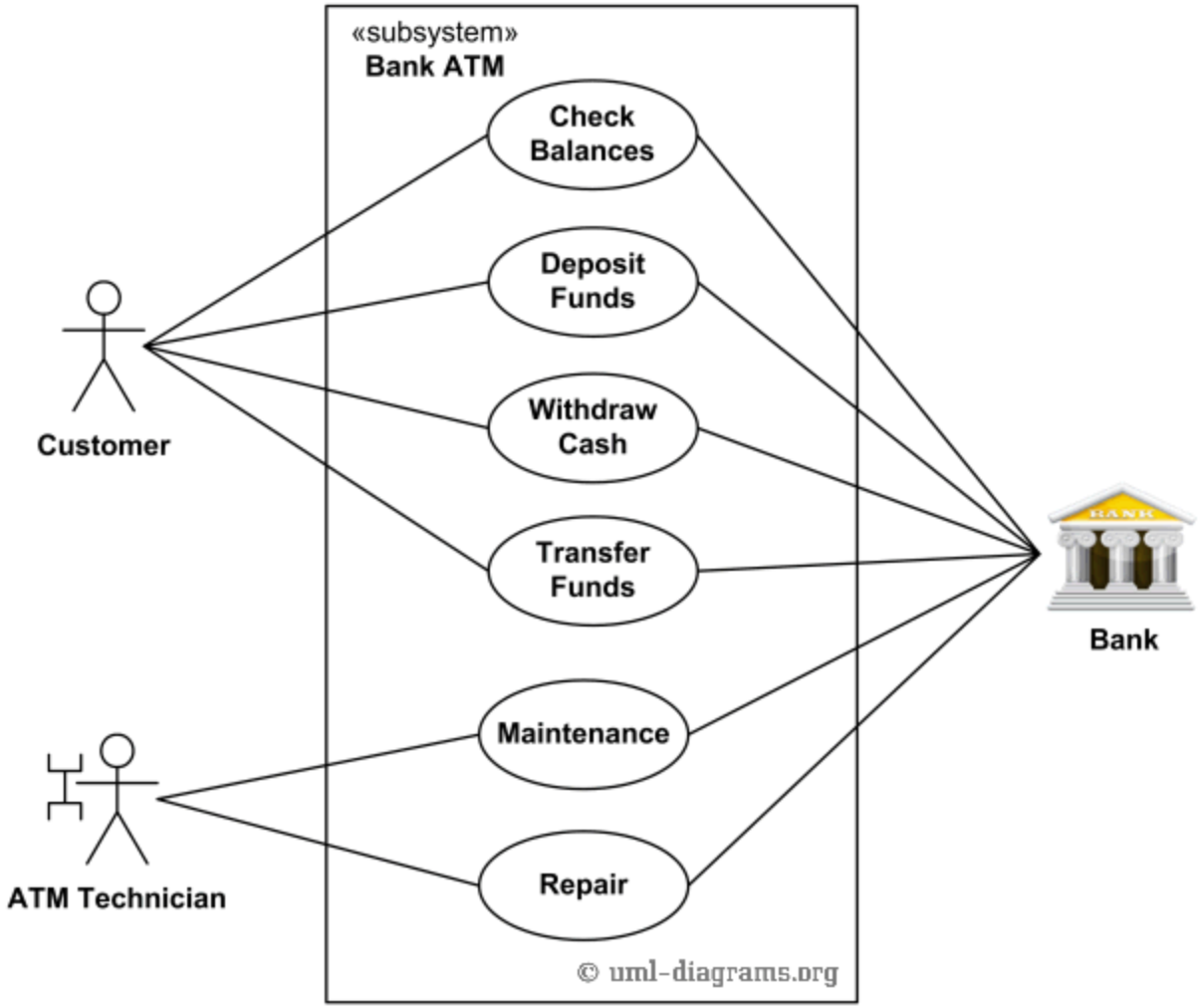
An automated teller machine (**ATM**) provides
bank customers with access to financial transactions.

Customer uses bank ATM to *Check Balances* of
his/her bank accounts, *Deposit Funds*, *Withdraw Cash* and/or
Transfer Funds

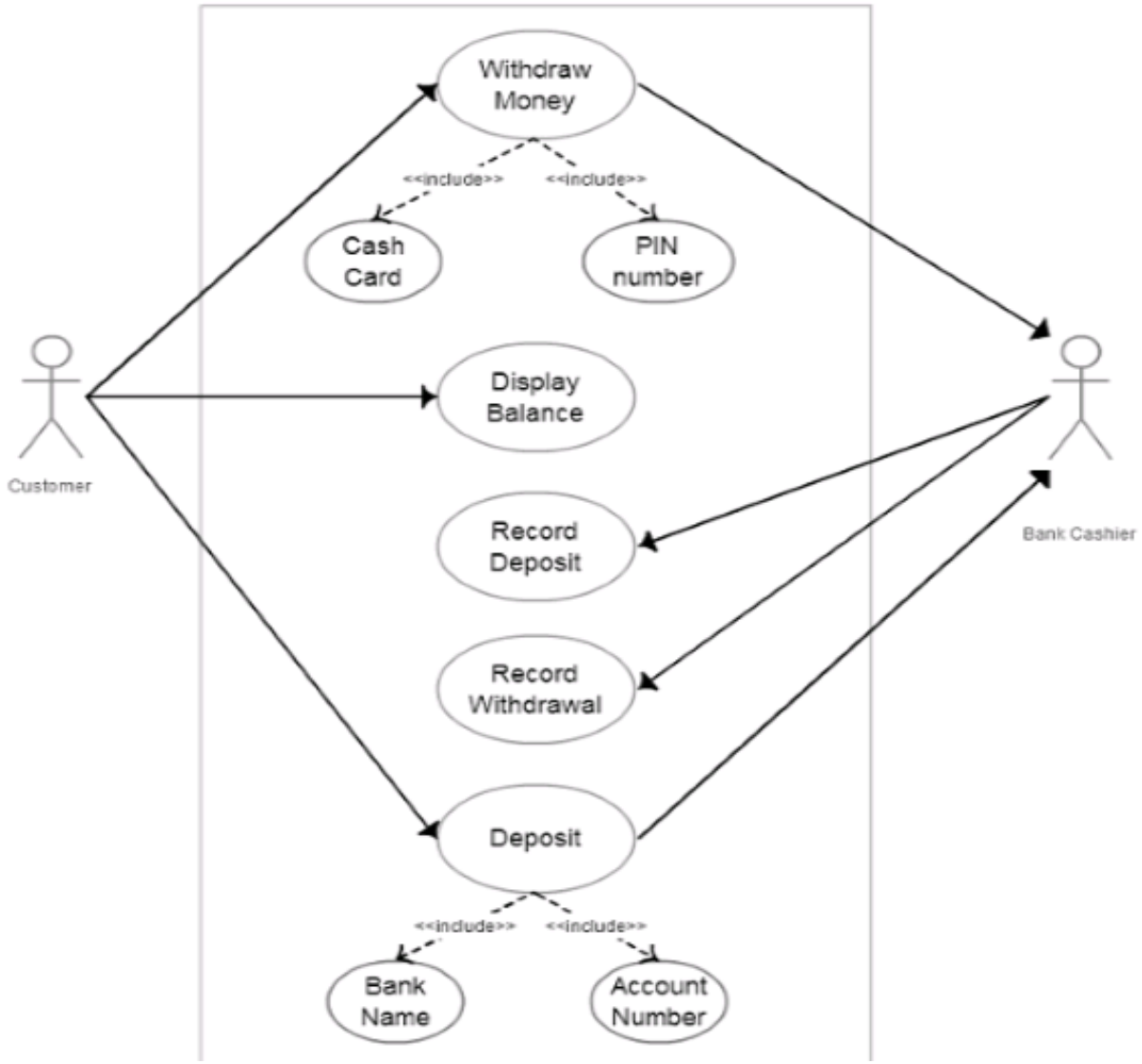
ATM Technician provides *Maintenance* and *Repairs*.

Bank actor: customer transactions or to the ATM servicing.

Usecase diagram for an ATM machine



Usecase diagram for an ATM machine

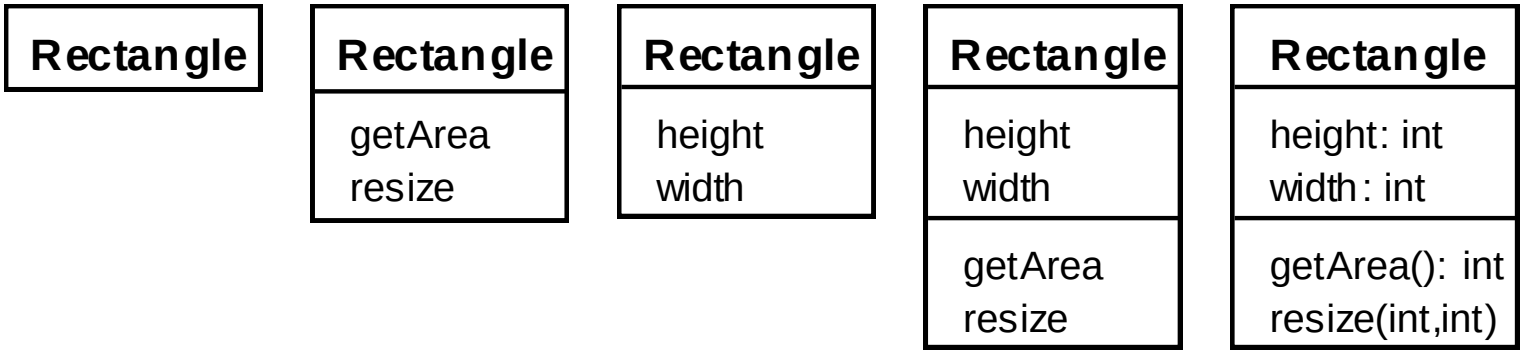


Essentials of UML Class Diagrams

- *The main symbols shown on class diagrams are:*
  - *Classes*
    - represent the types of data themselves
  - *Attributes*
    - are simple data found in classes and their instances
  - *Operations*
    - the functions performed by classes and their instances
  - *Associations*
    - represent linkages between instances of classes
  - *Generalizations*
    - group classes into inheritance hierarchies

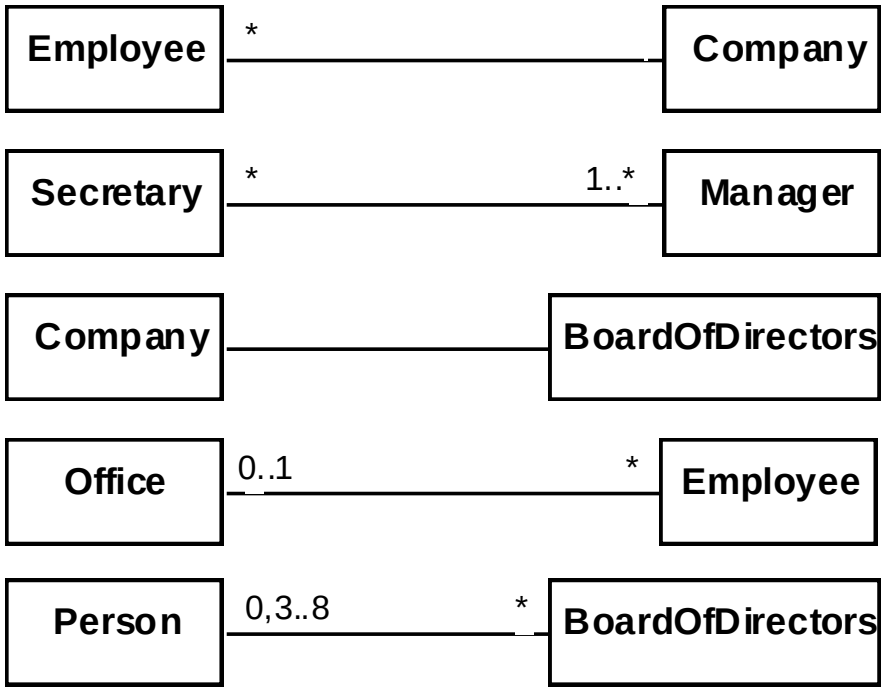
Classes

- A class is simply represented as a box with the name of the class inside
  - The diagram may also show the attributes and operations
  - The *UML signature* of an operation is: operationName(parameterName: parameterType …): returnType



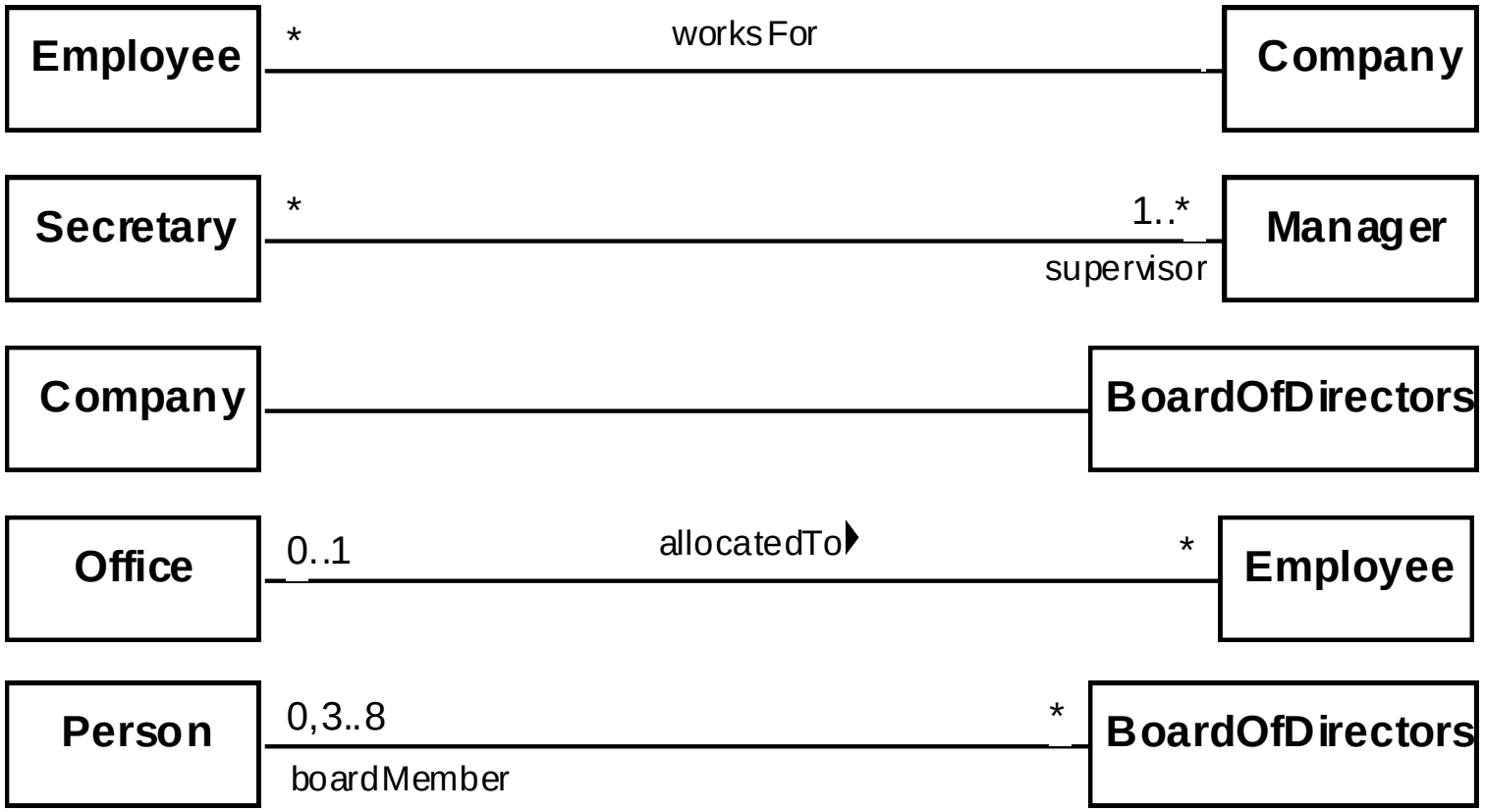
Associations and Multiplicity

- An *association* is a line that relates two classes
- Symbols indicating *multiplicity* are shown at each end



Labelling associations

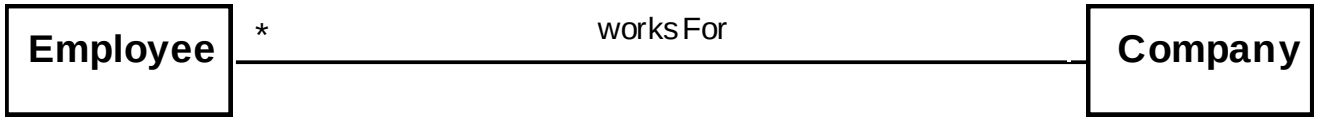
An association can be *labeled*, to clarify its nature



Interpreting associations

Many-to-one

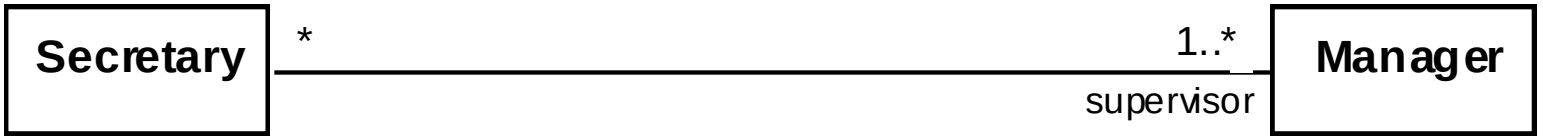
- A company has many employees,
- An employee can only work for one company.
  - No moonlighting!
- A company can have zero employees
  - E.g. a ‘shell’ company
- Every employee must work for some company



Interpreting associations

Many-to-many

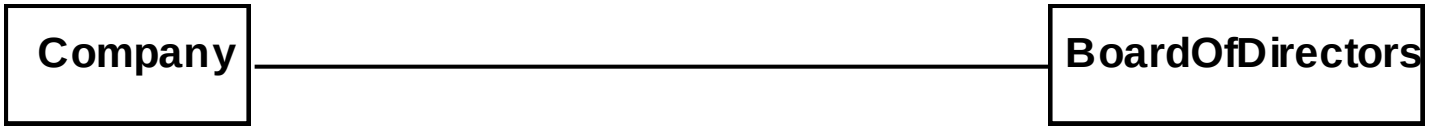
- A secretary can work for many managers
- A manager can have many secretaries
- Secretaries can work in pools
- Managers can have a group of secretaries
- Some managers might have zero secretaries.
- Is it possible for a secretary to have, perhaps temporarily, zero managers?



Interpreting associations

One-to-one

- For each company, there is exactly one board of directors
- A board is the board of only one company
- A company must always have a board
- A board must always be of some company



Another example

A booking is always for exactly one passenger

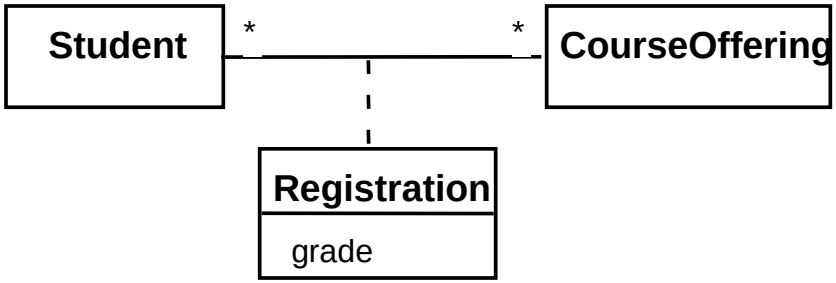
- no booking with zero passengers
- a booking could *never* involve more than one passenger.

A Passenger can have any number of Bookings

- a passenger could have no bookings at all
- a passenger could have more than one booking

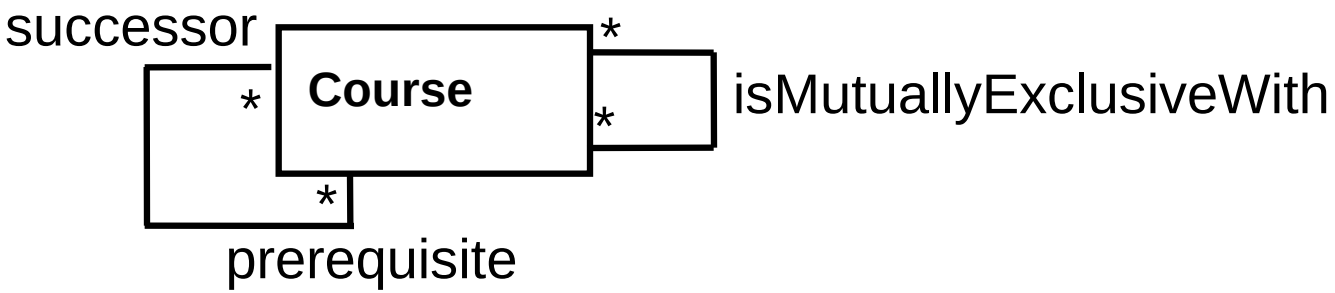
Association classes

Sometimes, an attribute shared by two associated classes cannot be placed in either one. E.g., the following are equivalent:



Reflexive associations

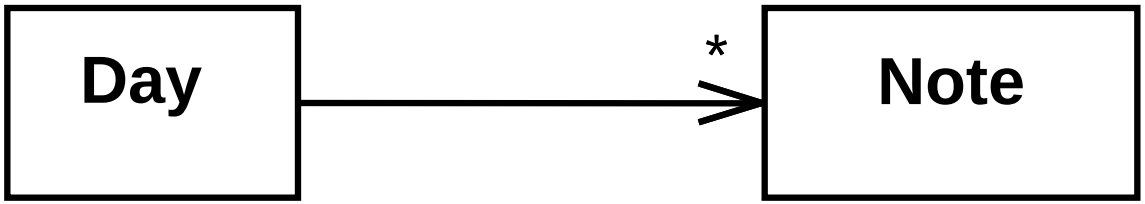
An association can connect a clas



Directionality in associations

Associations are by default *bi-directional*

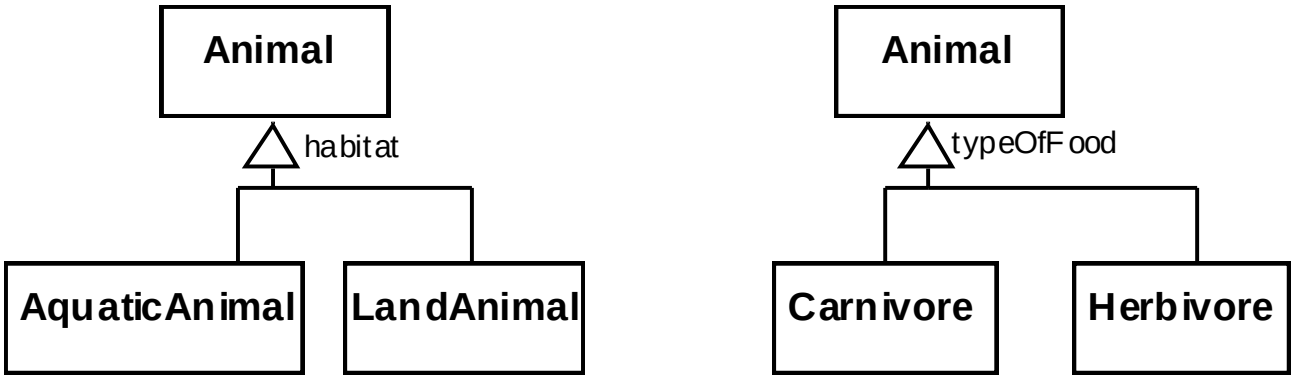
The direction can be limited by adding an arrow:



Generalization

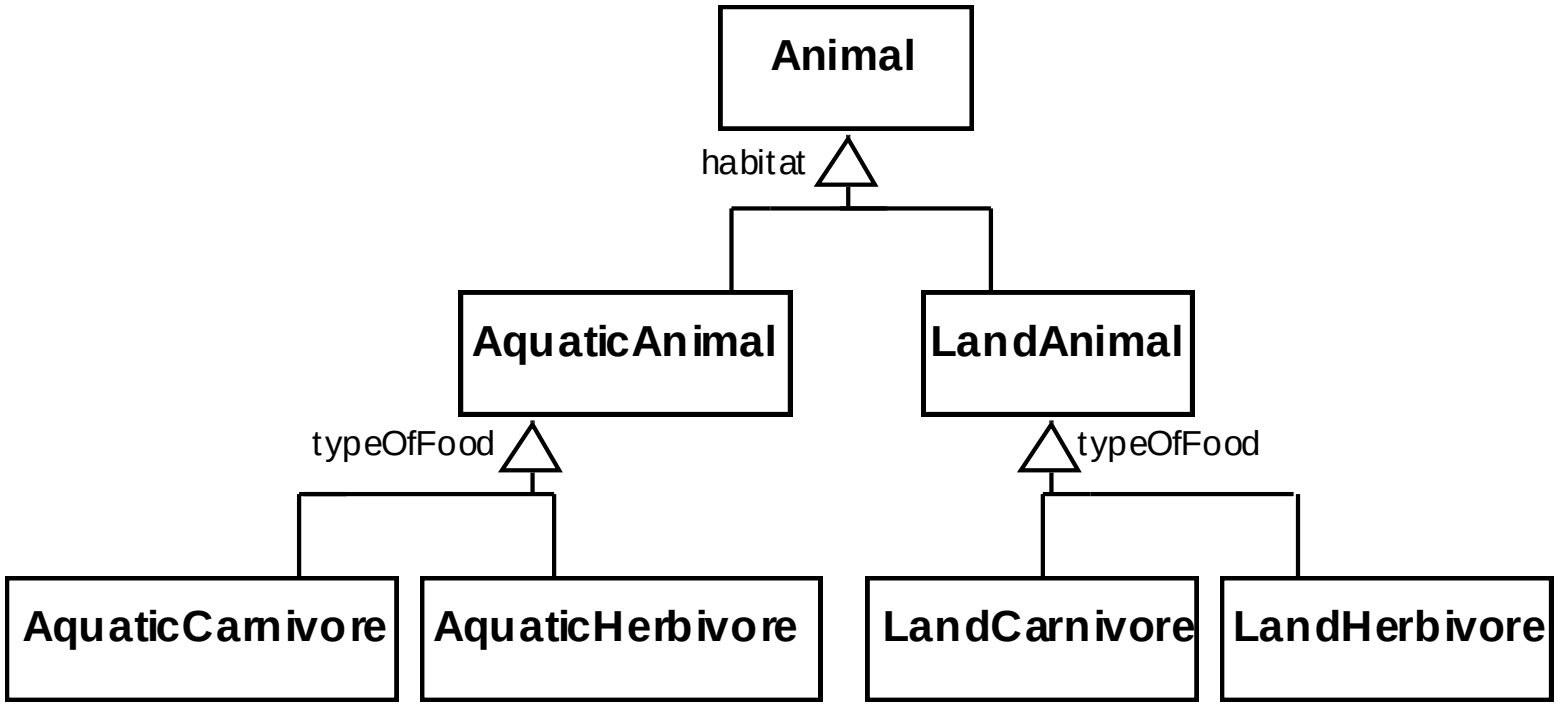
- Specializing a superclass into two or more subclasses

  The *label* that describes the criterion for specialization



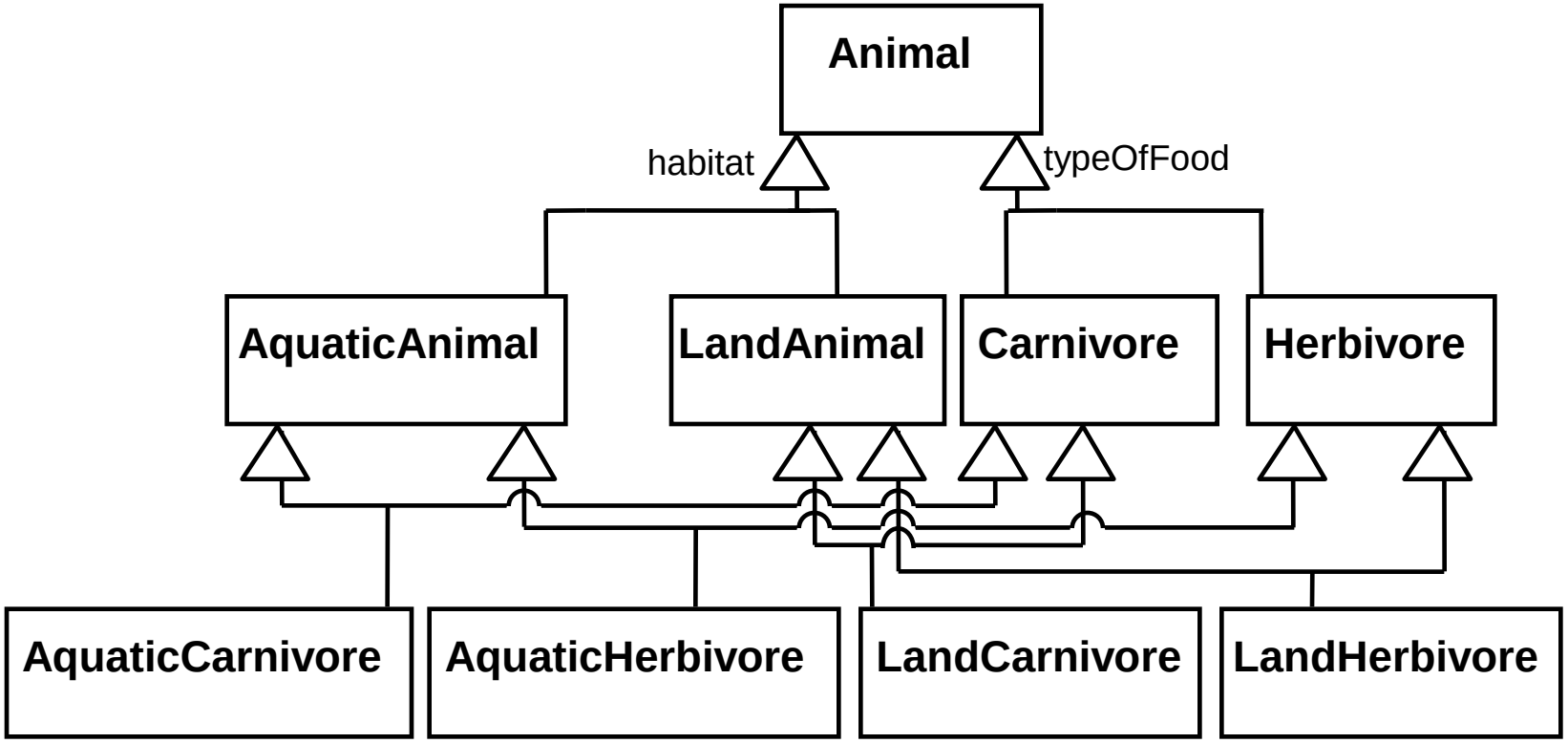
Handling multiple discriminators

Creating higher-level generalization



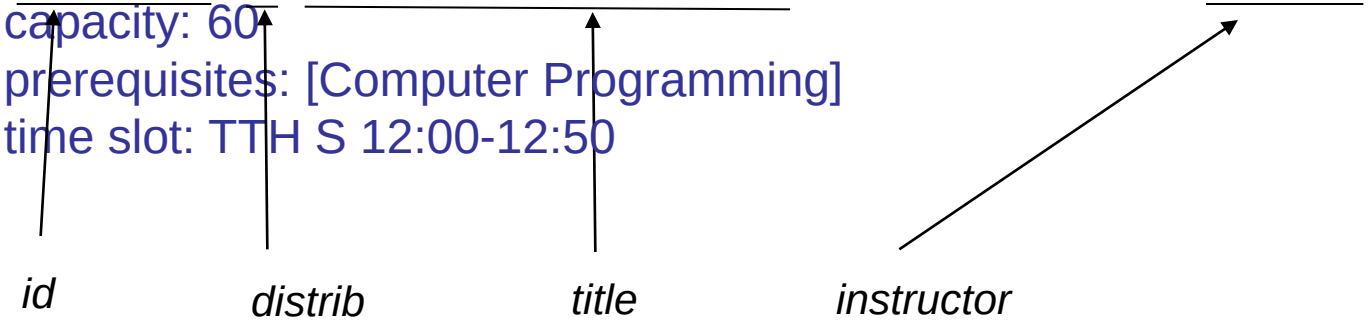
Handling multiple discriminators

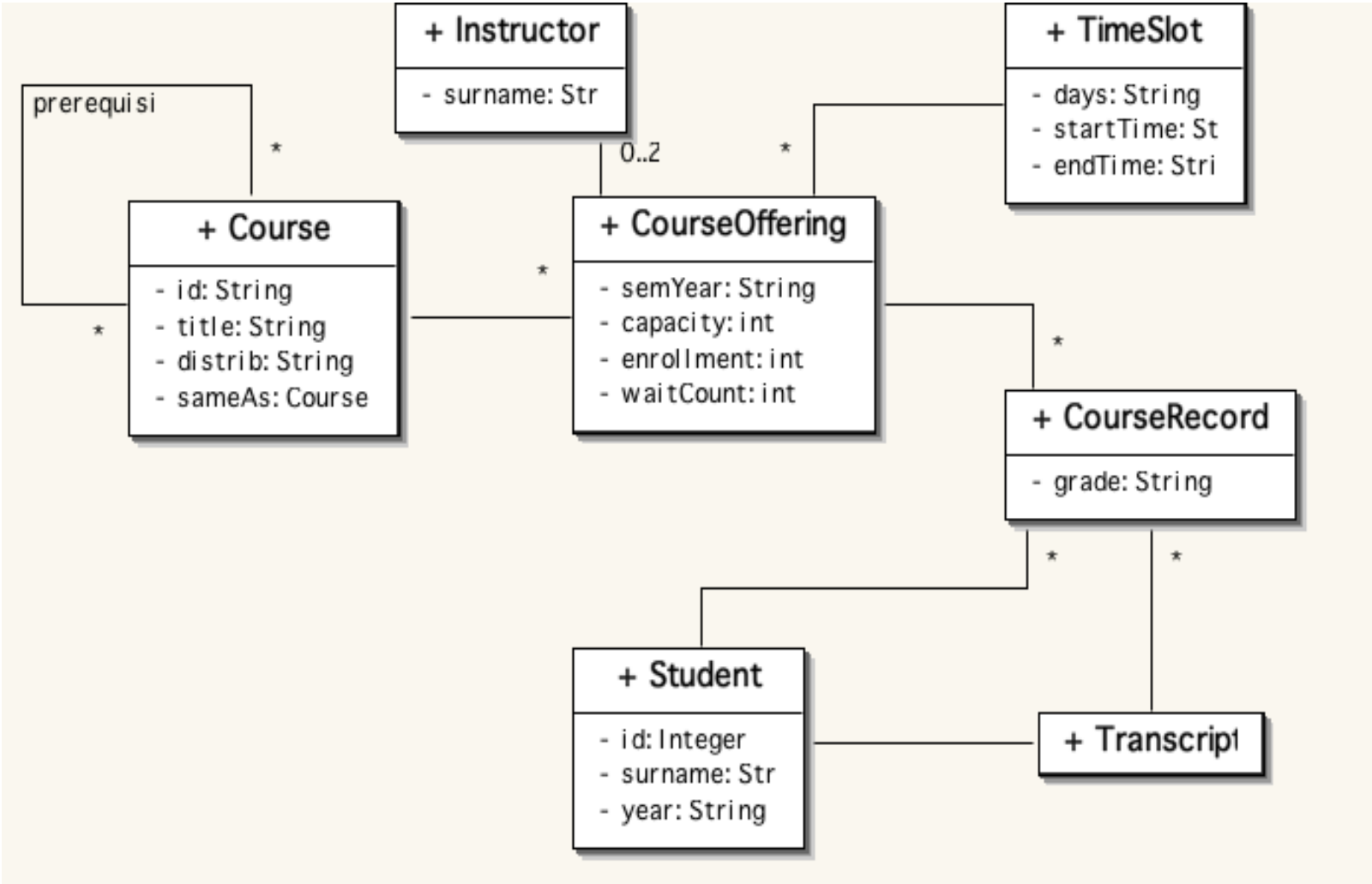
Using multiple inheritance



Course offerings encoded

CS G523 Software for Embedded Systems TP class room Neena Goveas
capacity: 60
prerequisites: [Computer Programming]
time slot: TTH S 12:00-12:50




+ Instructor
- surname: Str
+ TimeSlot
- days: String
- startTime: St
- endTime: Stri
prerequisi
*
0..2
*
+ Course
- id: String
- title: String
- distrib: String
- sameAs: Course
+ CourseOffering
- semYear: String
- capacity: int
- enrollment: int
- waitCount: int
*
*
*
+ CourseRecord
- grade: String
*
*
+ Student
- id: Integer
- surname: Str
- year: String
+ Transcript

Associations versus generalizations in instance diagrams

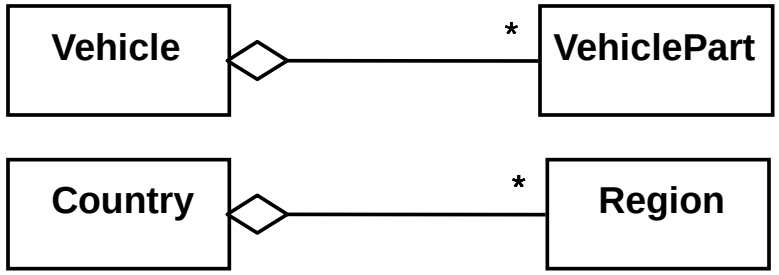
- Associations describe relationships between *instances* at *run time*
  - An instance diagram is generated from a class diagram
  - It shows an instance of *both* classes joined by each association

- Generalizations describe *static* relationships between *classes*
  - They do not appear in instance diagrams at all
  - An instance of a class is also an instance of all its superclass(es)

Aggregation

Aggregations are associations that represent ‘part-whole’ relationships.

- The ‘whole’ side is often called the *assembly* or the *aggregate*
- This symbol is a shorthand notation association named isPartOf

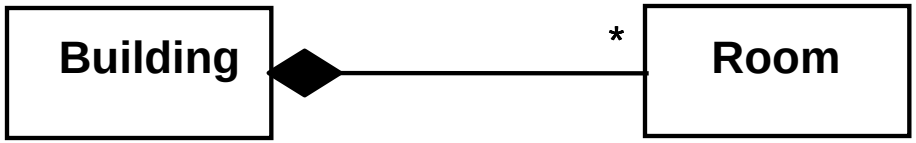


When to use an aggregation

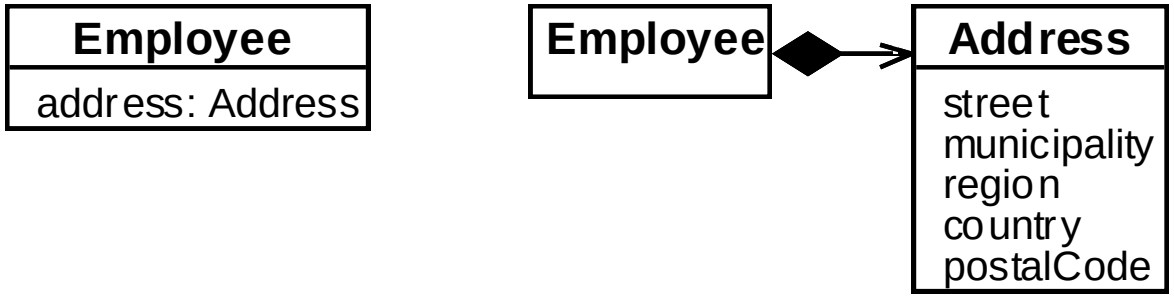
- An association is an aggregation if:
  - You can state that
    - the parts ‘are part of’ the aggregate
    - or the aggregate ‘is composed of’ the parts
  - When something owns or controls the aggregate, then they also own or control the parts

Composition

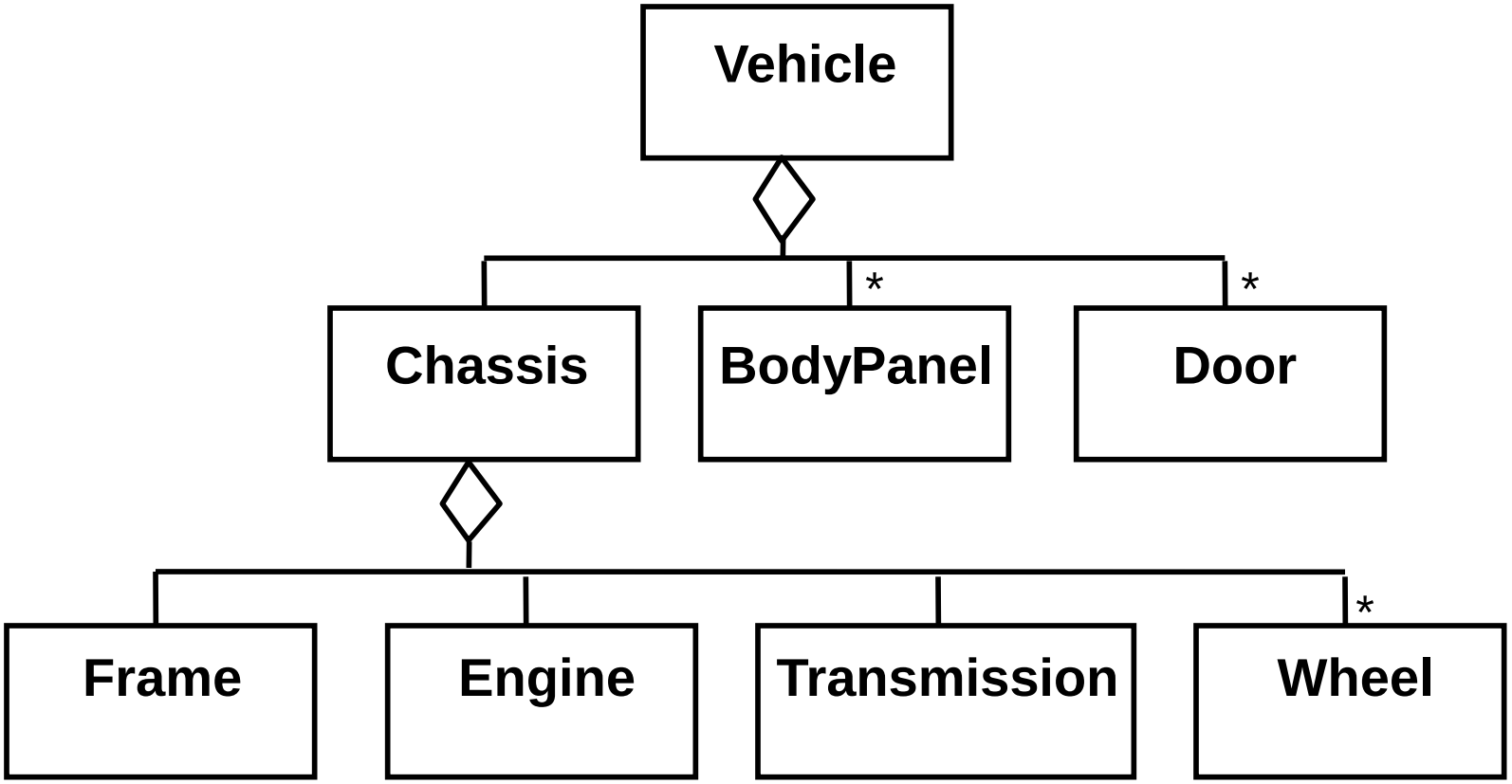
- A *composition* is a strong kind of aggregation
  - if the aggregate is destroyed, the parts are also destroyed



- Two alternatives for addresses

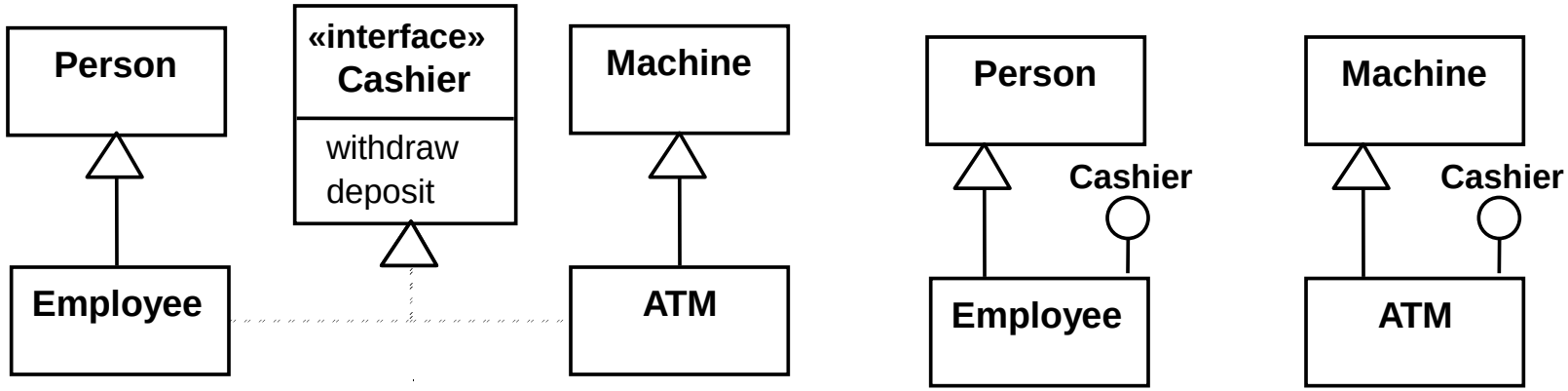


Aggregation hierarchy



Interfaces

- An interface describes a *portion of the visible behaviour* of a class.
- An *interface* is similar to a class, except it lacks instance variables and method bodies



Modelling Interactions and Behaviour

Interaction Diagrams

- Interaction diagrams are used to model the dynamic aspects of a software system
  - They help you to visualize how the system runs.
  - An interaction diagram is often built from a use case and a class diagram.
    - The objective is to show how a set of objects accomplish the required interactions with an actor.

Interactions and messages

- Interaction diagrams show how a set of actors and objects communicate with each other to perform:
  - The steps of a use case, or
  - The steps of some other piece of functionality.

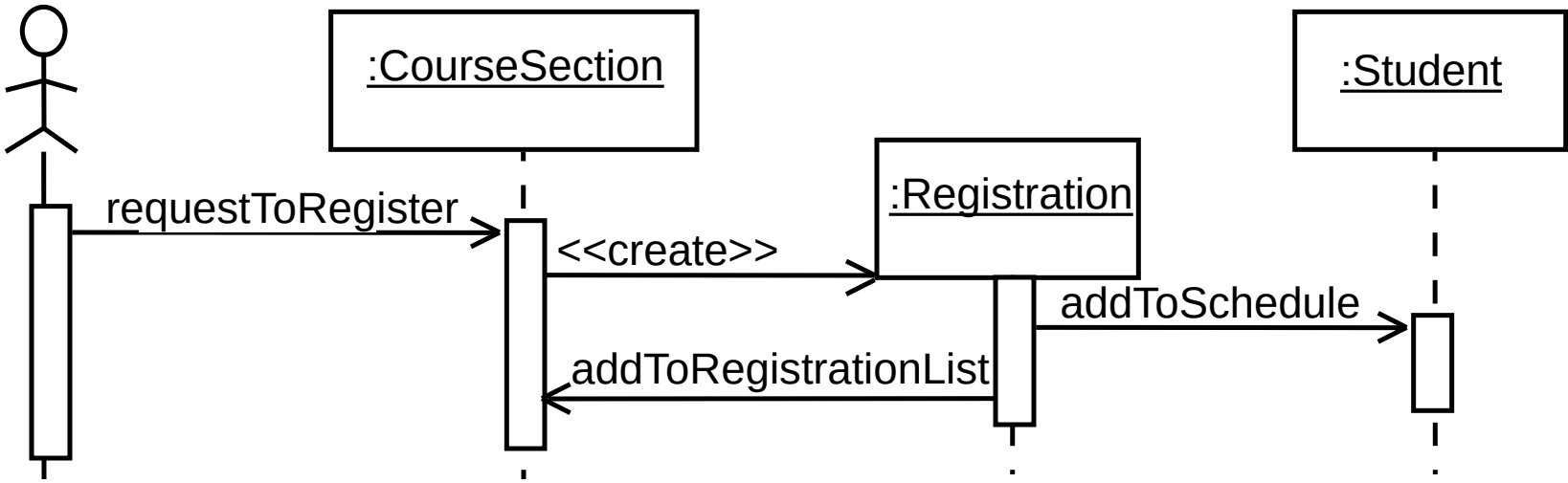
- The set of steps, taken together, is called an *interaction*.

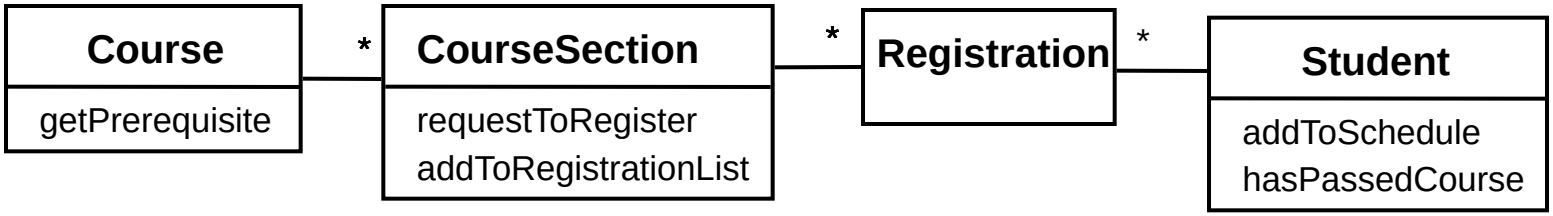
- Interaction diagrams can show several different types of communication.
  - E.g. method calls, messages send over the network
  - These are all referred to as *messages*.

Elements found in interaction diagrams

- Instances of classes
  - Shown as boxes with the class and object identifier underlined
- Actors
  - Use the stick-person symbol as in use case diagrams
- Messages
  - Shown as arrows from actor to object, or from object to object

Sequence diagrams – an example

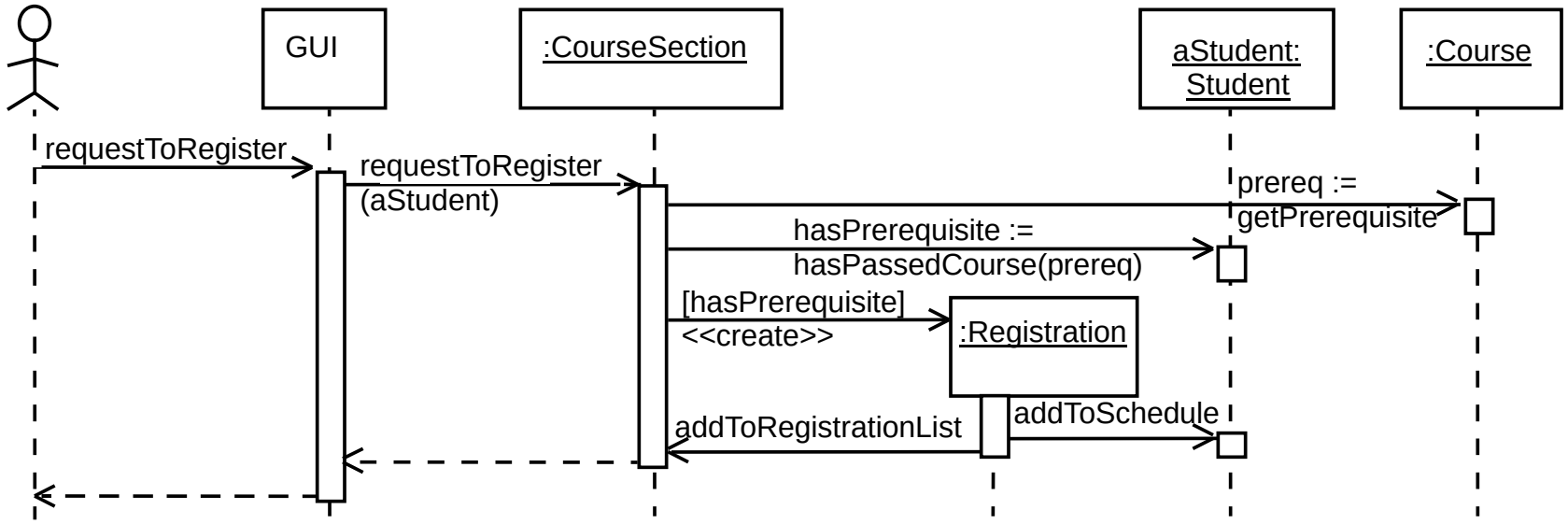




Sequence diagrams

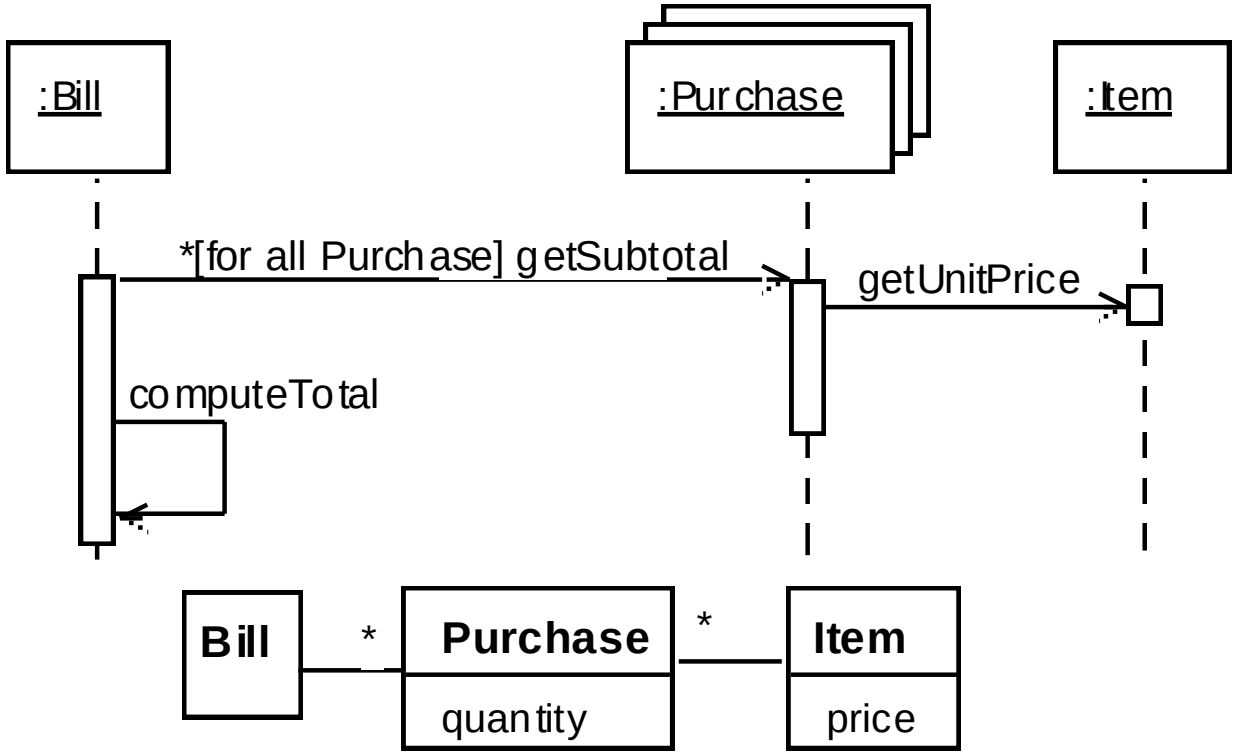
- A sequence diagram shows the sequence of messages exchanged by the set of objects performing a certain task
  - The objects are arranged horizontally across the diagram.
  - An actor that initiates the interaction is often shown on the left.
  - The vertical dimension represents time.
  - A vertical line, called a *lifeline*, is attached to each object or actor.
  - The lifeline becomes a broad box, called an *activation box* during the *live activation* period.
  - A message is represented as an arrow between activation boxes of the sender and receiver.
    - A message is labelled and can have an argument list and a return value.

Sequence diagrams – same example, more details



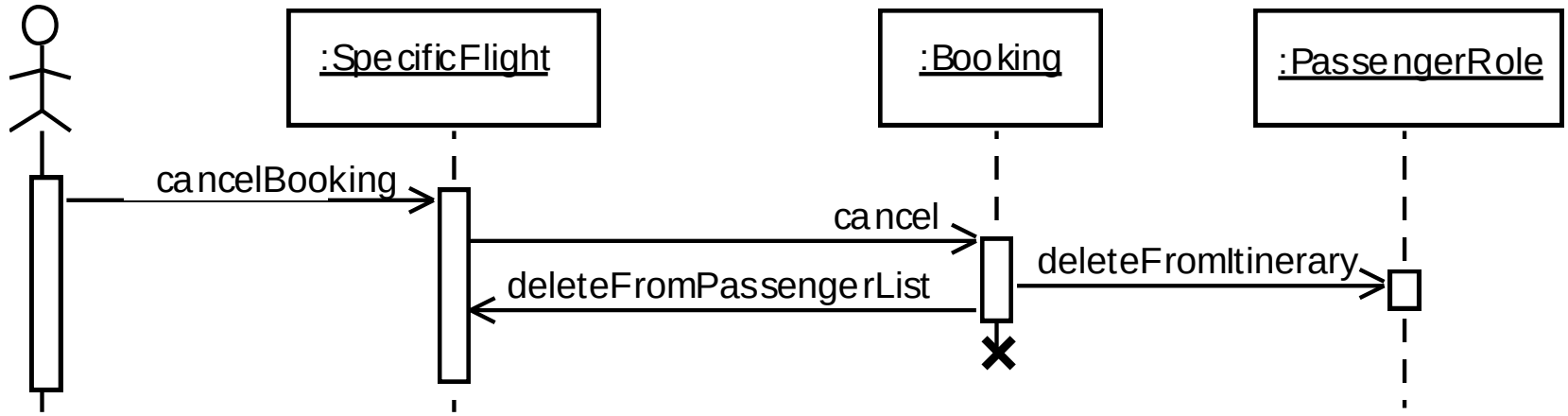
Sequence diagrams

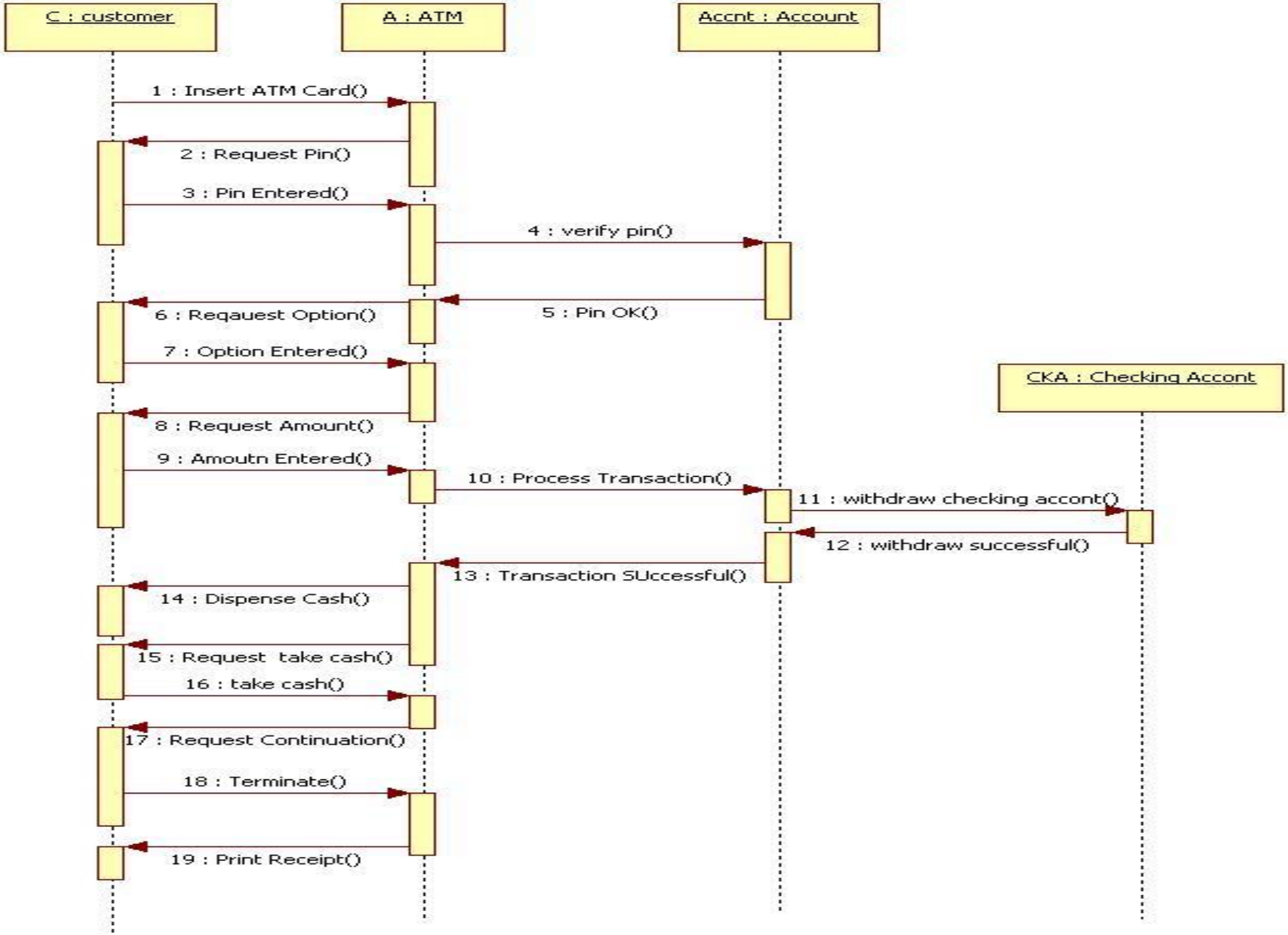
– An *iteration* over objects is indicated by an asterisk preceding the message name



Object deletion

– If an object's life ends, this is shown with an X at the end of the lifeline

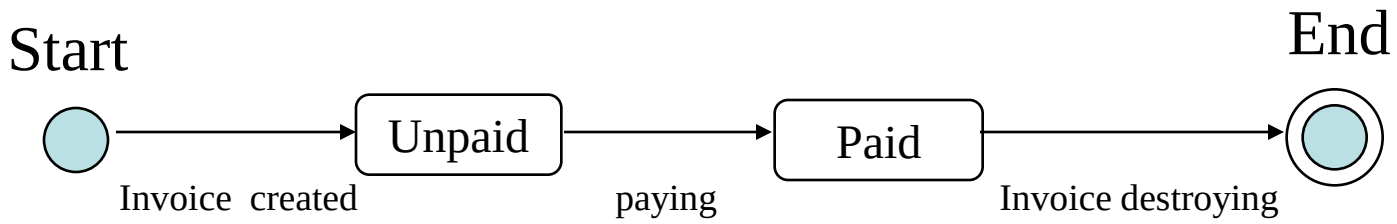



C : customer
A : ATM
Accnt : Account
CKA : Checking Accont
1 : Insert ATM Card()
2 : Request Pin()
3 : Pin Entered()
4 : verify pin()
5 : Pin OK()
6 : Reqauest Option()
7 : Option Entered()
8 : Request Amount()
9 : Amoutn Entered()
10 : Process Transaction()
11 : withdraw checking accont()
12 : withdraw successful()
13 : Transaction SUccessful()
14 : Dispense Cash()
15 : Request take cash()
16 : take cash()
17 : Request Continuation()
18 : Terminate()
19 : Print Receipt()

Statechart diagram

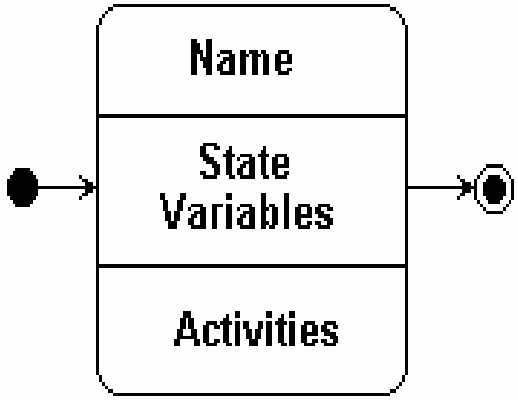
State Diagrams

State Diagrams show the sequences of states an object goes through during its life cycle in response to stimuli, together with its responses and actions; an abstraction of all possible behaviors.



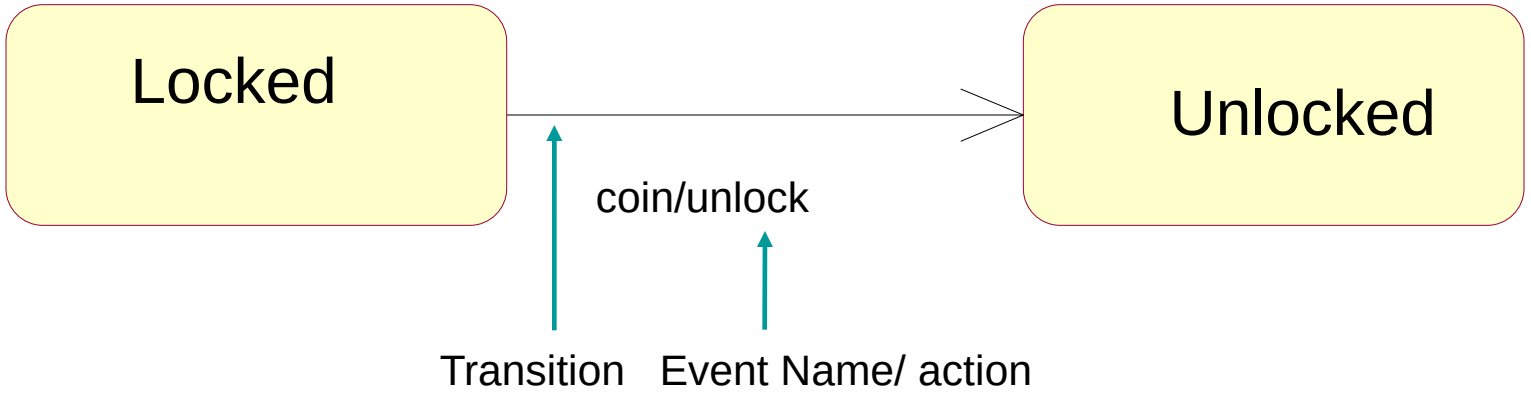
Special States

- The initial state is the state entered when an object is created.
  - An initial state is mandatory.
  - Only one initial state is permitted.
  - The initial state is represented as a solid circle.
- A final state indicates the end of life
  - A final state is optional.
  - A final state is indicated by a bull's eye.
  - More than one final state may exist.



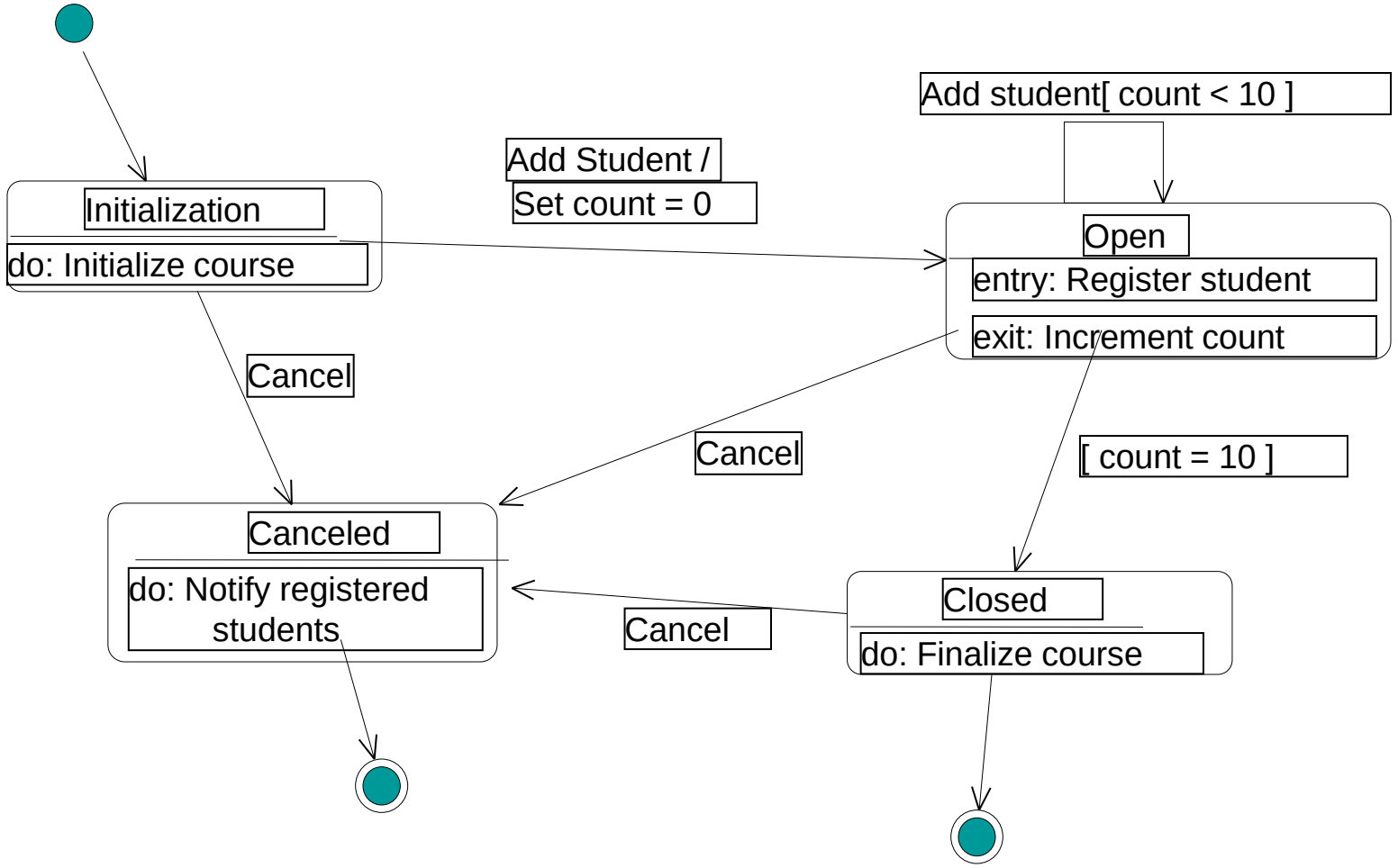
Events, Actions & Transitions

- An event: stimulus that can trigger a state transition.
- A transition: is a change from an originating state to a successor state as a result of some stimulus.
  - The successor state could possibly be the originating state.
- A transition may take place in response to an event.
- Transitions can be labeled with event names.



Statechart Diagram

- A statechart diagram shows the lifecycle of a single class.

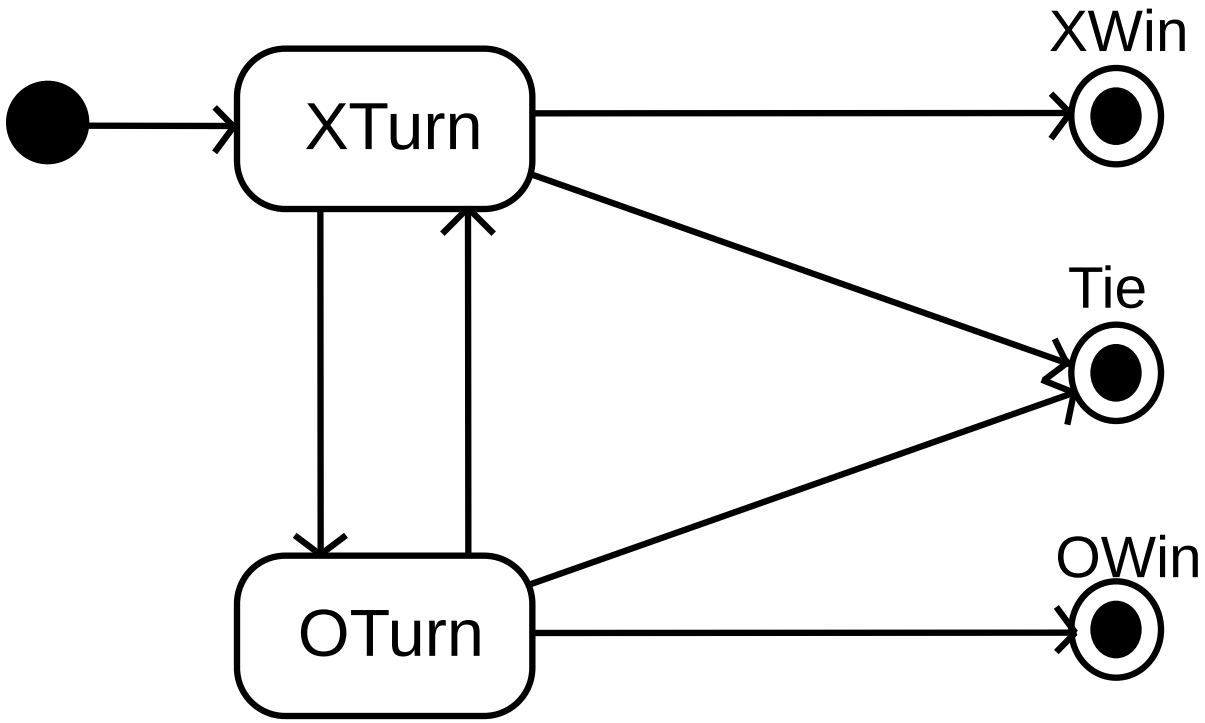


State Diagrams

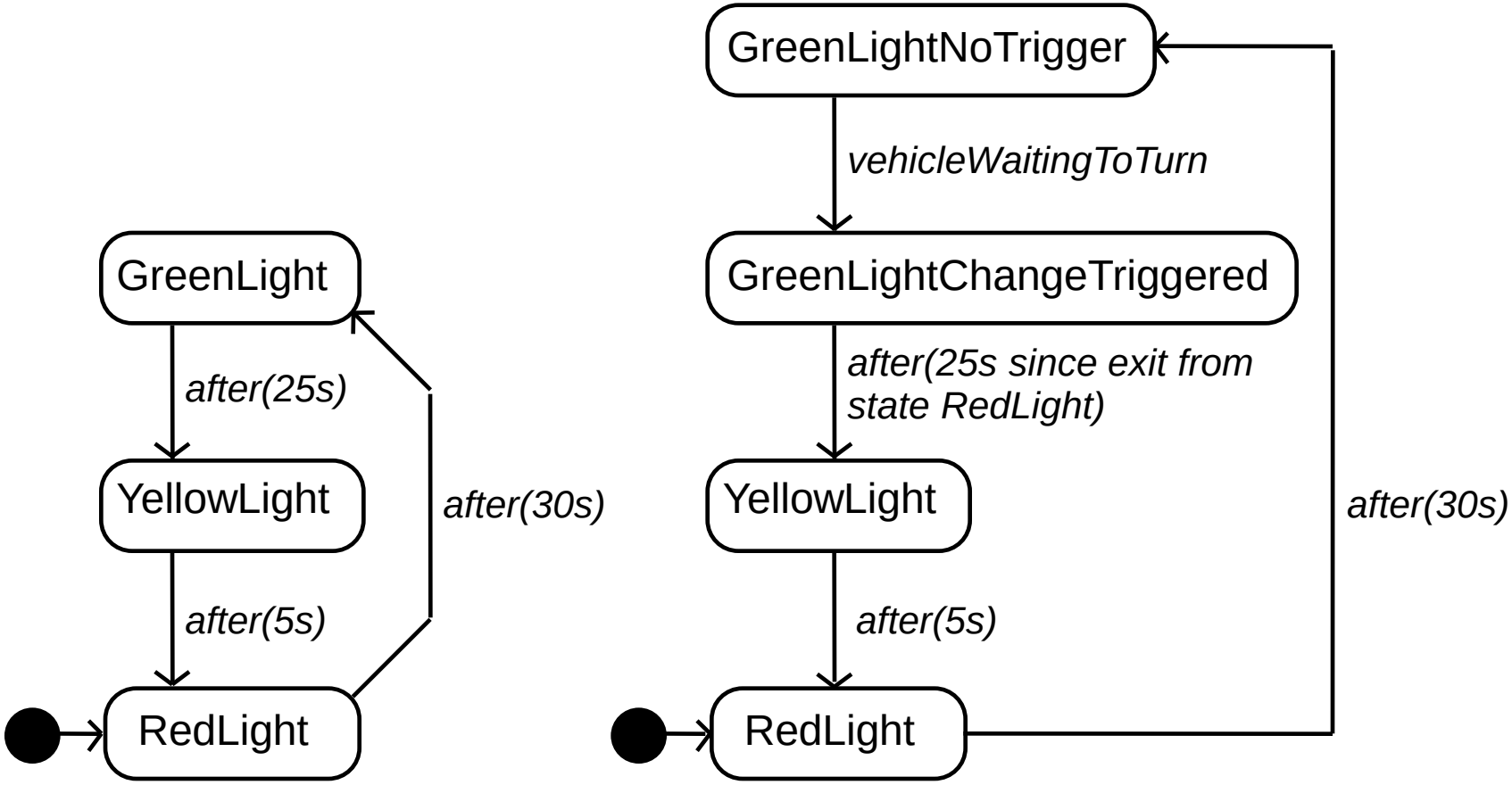
- A state diagram describes the behaviour of a *system*, some *part* of a system, or an *individual object*.
  - At any given point in time, the system or object is in a certain state.
    - Being in a state means that it is will behave in a *specific way* in response to any events that occur.
  - Some events will cause the system to change state.
    - In the new state, the system will behave in a different way to events.
  - A state diagram is a directed graph where the nodes are states and the arcs are transitions.

State diagrams – an example

e



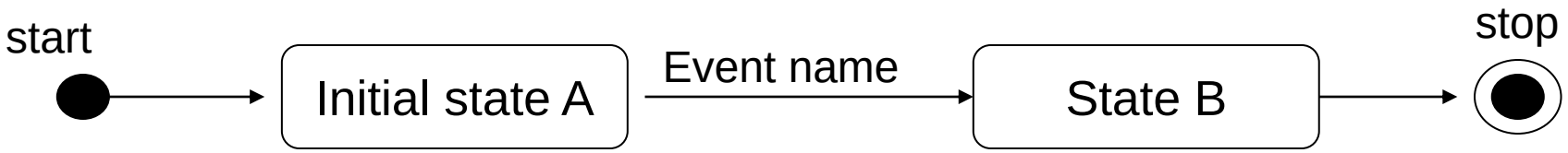
State diagrams – an example of transitions with time-outs and conditions



Activities in state diagrams

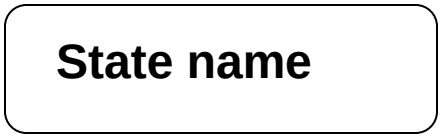
- An *activity* is something that takes place while the system is *in* a state.
  - It takes a period of time.
  - The system may take a transition out of the state in response to completion of the activity,
  - Some other outgoing transition may result in:
    - The interruption of the activity, and
    - An early exit from the state.

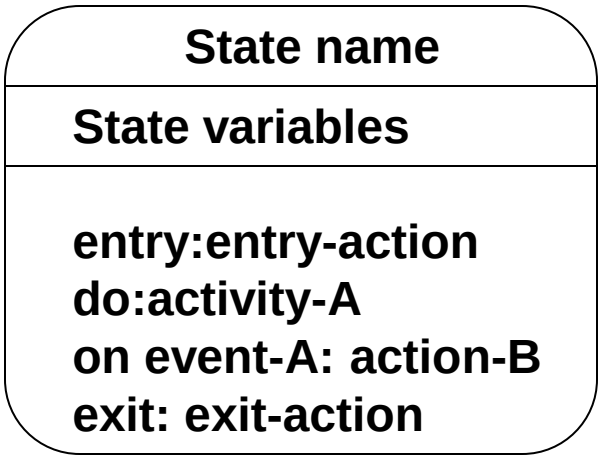
State Diagrams



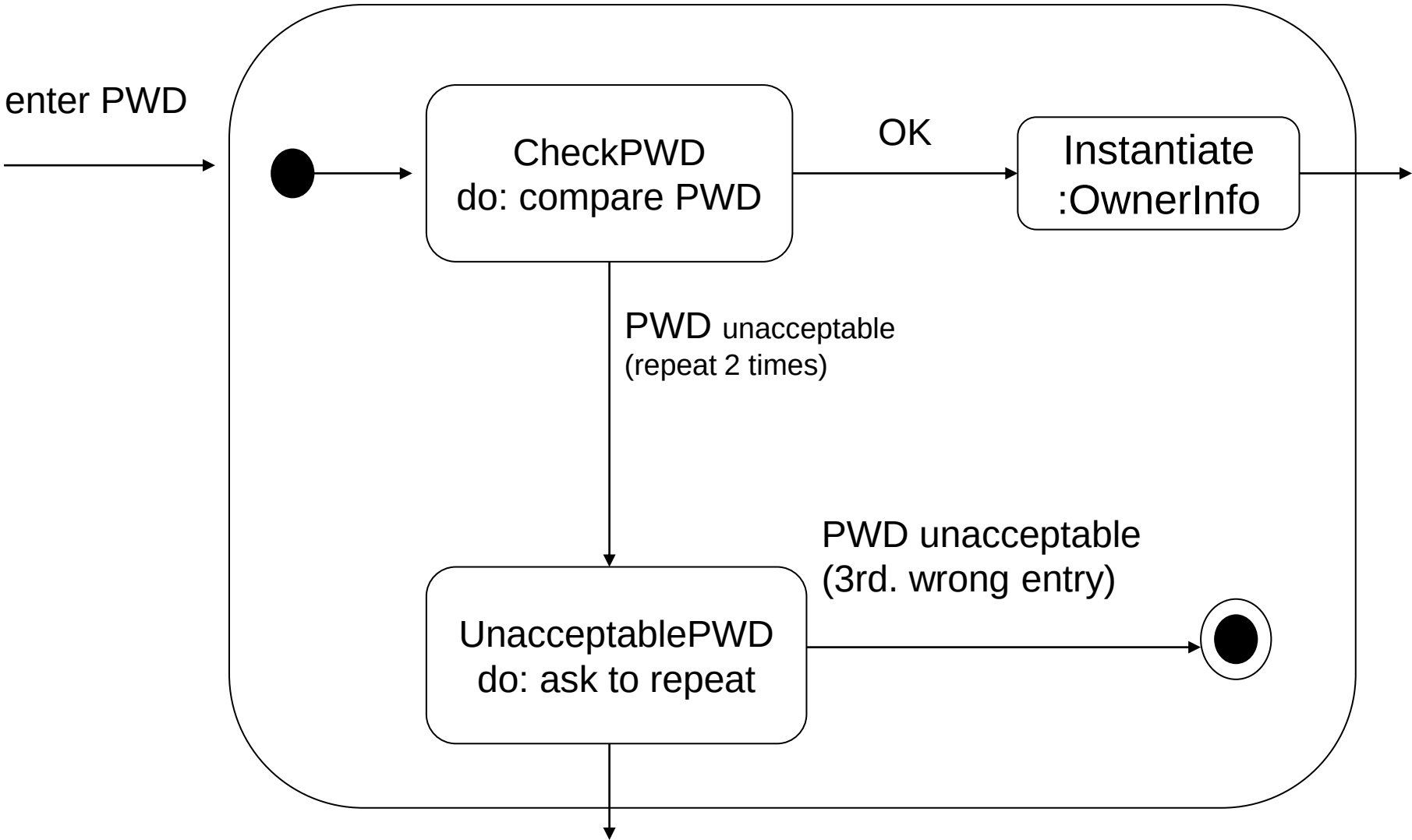
Reserved actions:

- entry: a specific action performed on the entry to the state
- do: an ongoing action performed while in the state
- on: a specific action performed as a result of a specific event
- Exit: a specific action performed on exiting the state

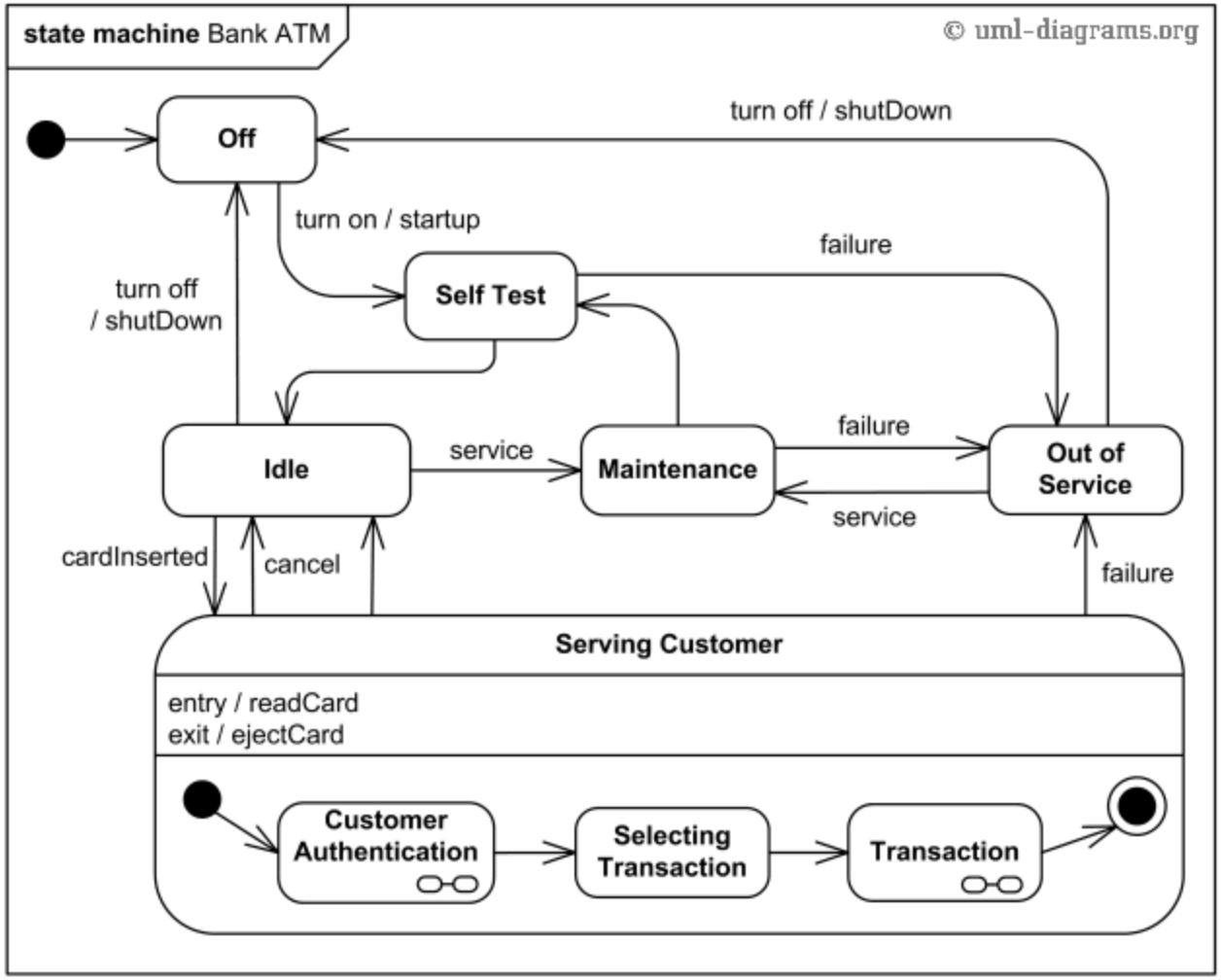




State Diagram Example

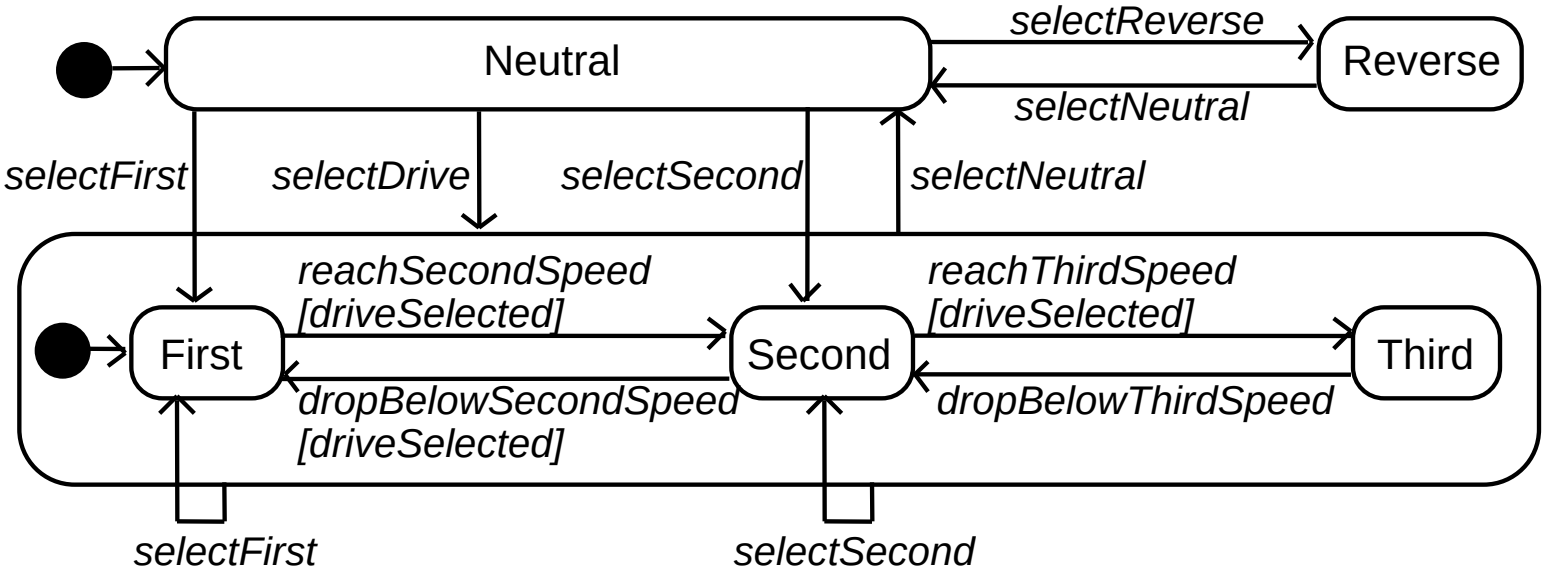


State Diagram Example

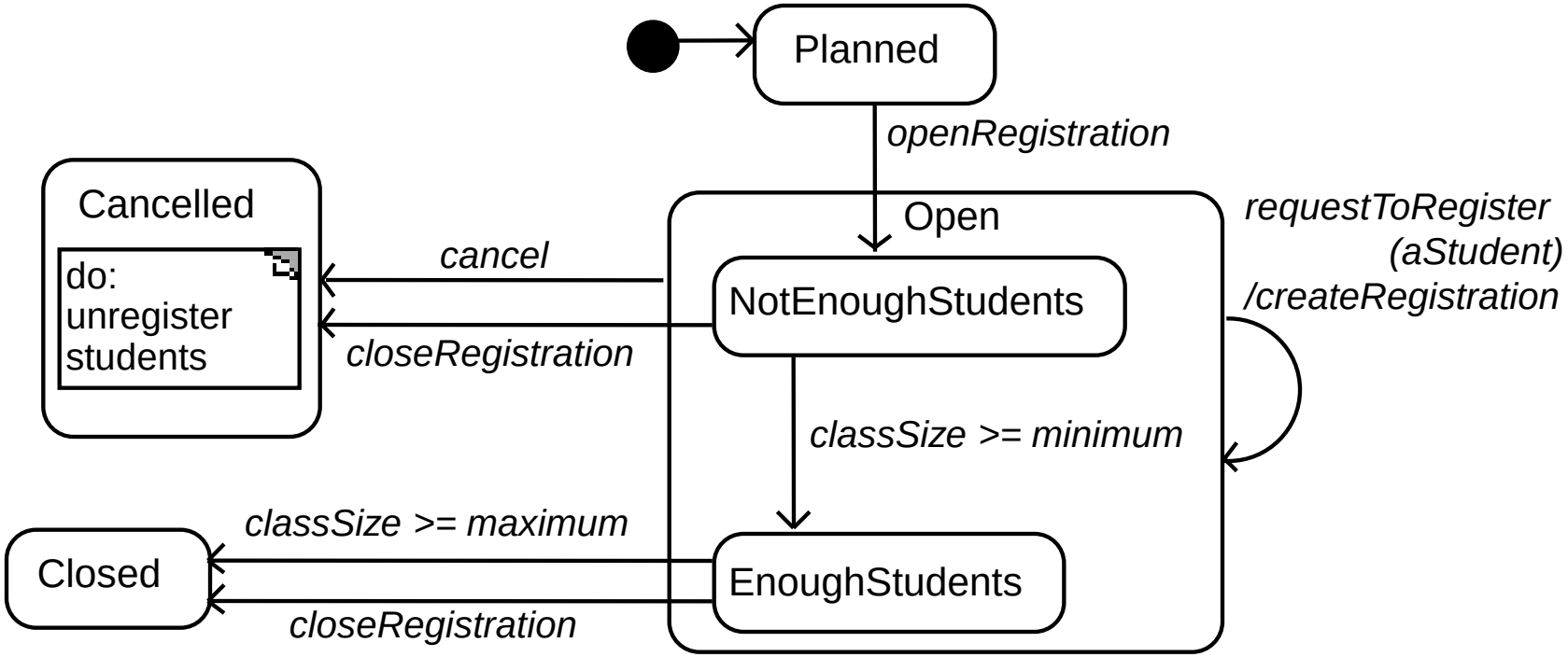


Nested substates and guard conditions

- A state diagram can be nested inside a state.
  - The states of the inner diagram are called *substates*.



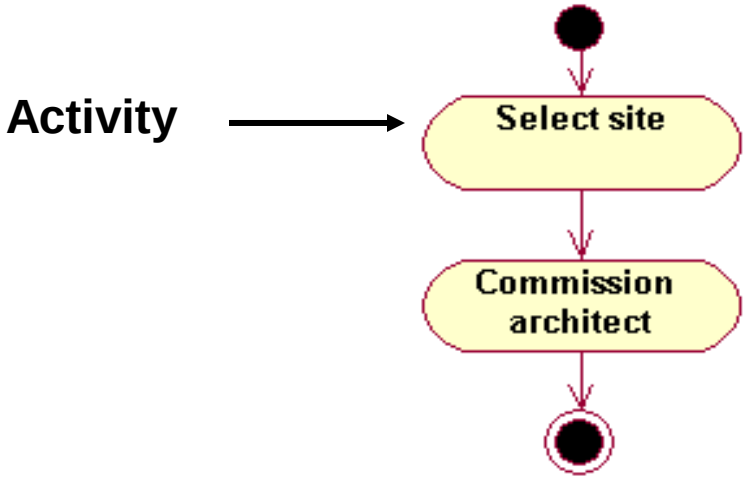
State diagram –substates



Activity Diagrams

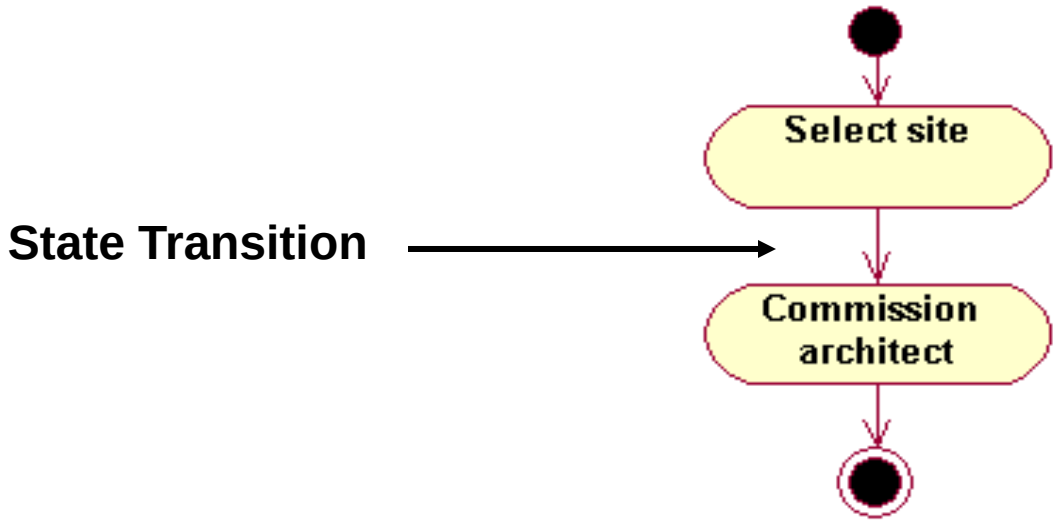
Activity

- An activity represents the performance of a task within the workflow.
- In the UML, an activity is represented by a lozenge (horizontal top and bottom with convex sides).



State Transitions

- A state transition shows what activity follows after another.
- In the UML, a state transition is represented by a solid line with an arrow.



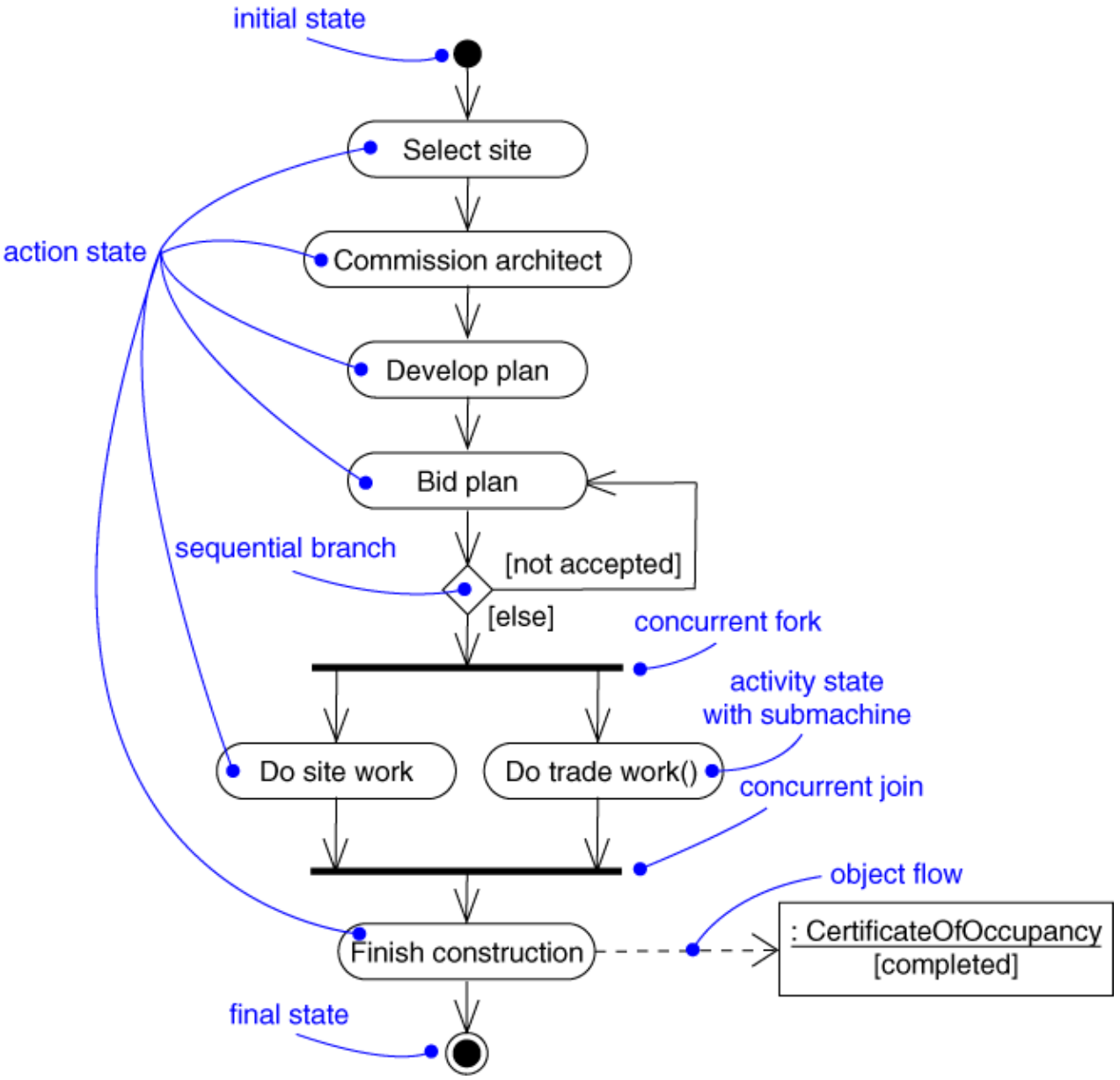
Decisions

- A decision is a point in an activity diagram where guard conditions are used to indicate different possible transitions.
- In the UML, a decision is represented by a diamond.

Synchronization Bars

- A synchronization bar allows you to show concurrent threads in a workflow of a use case.
- In the UML, a synchronization bar is represented by a thick horizontal or vertical line.

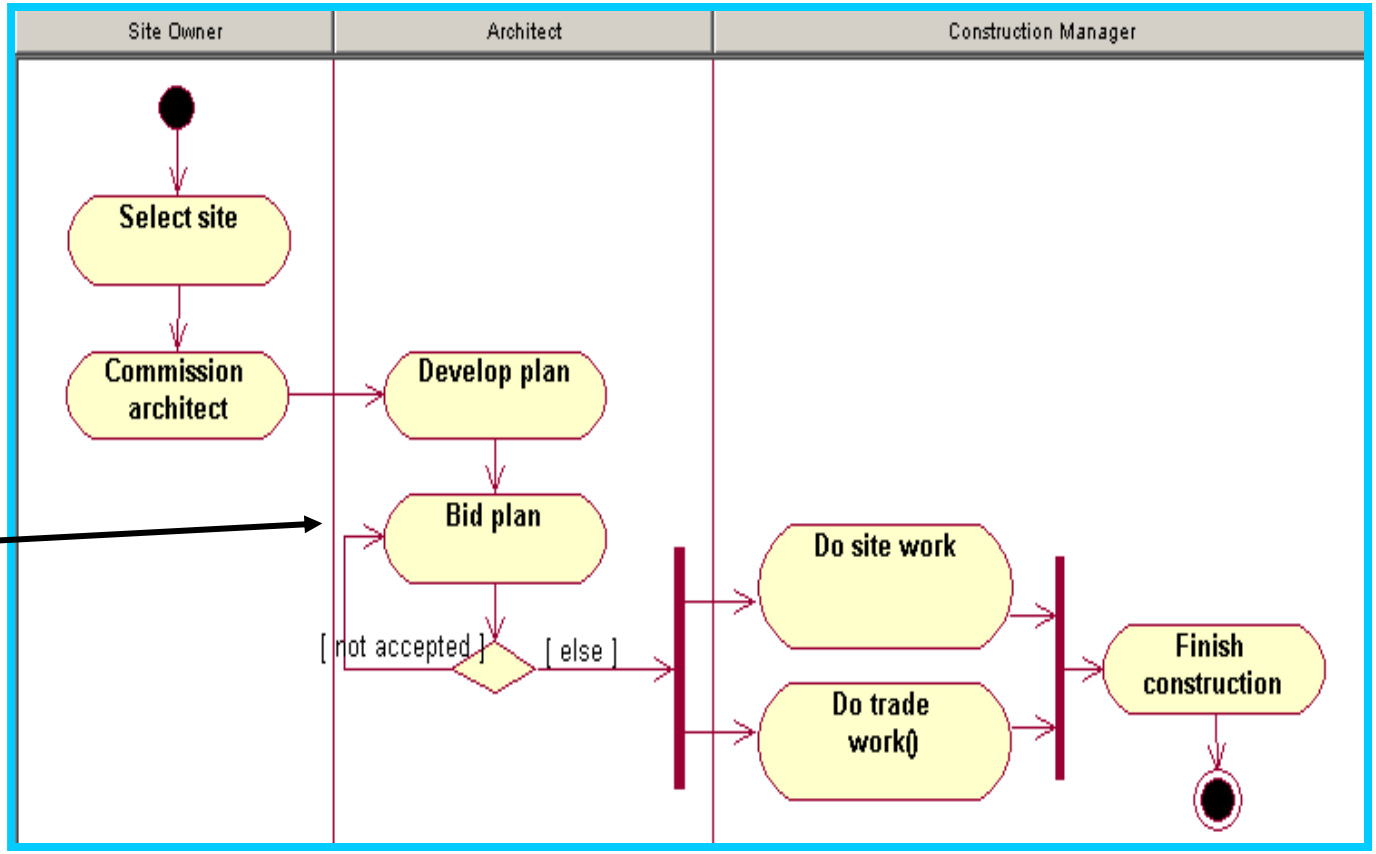
Activity Diagram



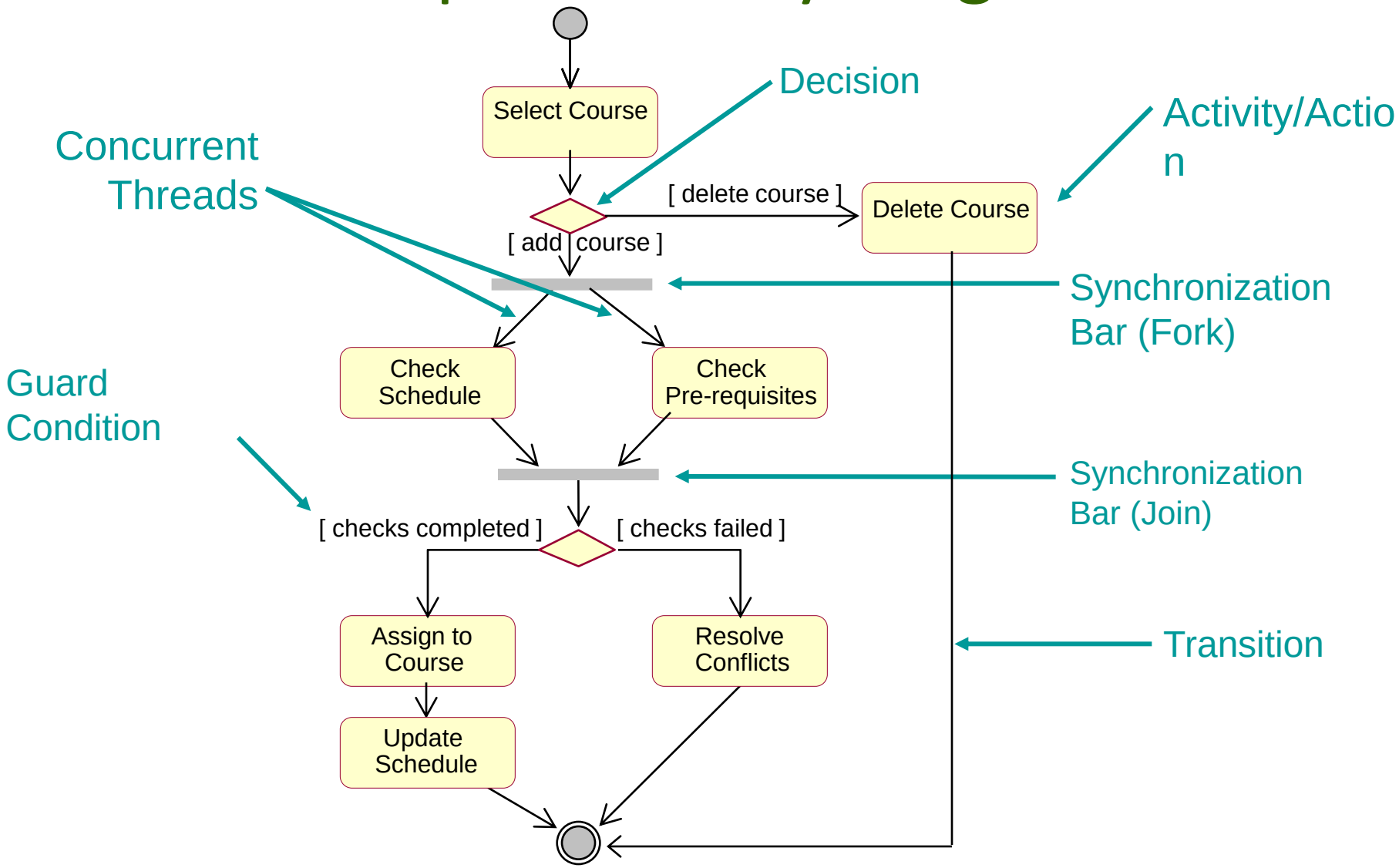
Swimlanes

- A swimlane is used to partition an activity diagram to help us better understand who or what is initiating the activity.

Swimlane



Example: Activity Diagram



Activity Diagrams

- An *activity diagram* is like a state diagram.
  - Except most transitions are caused by ***internal* events**, such as the completion of a computation.

- An activity diagram
  - Can be used to understand the flow of work that an object or component performs.
  - Can also be used to visualize the interrelation and interaction between different use cases.
  - Is most often associated with several classes.

- One of the strengths of activity diagrams is the representation of *concurrent* activities.

Watch: Diagrams

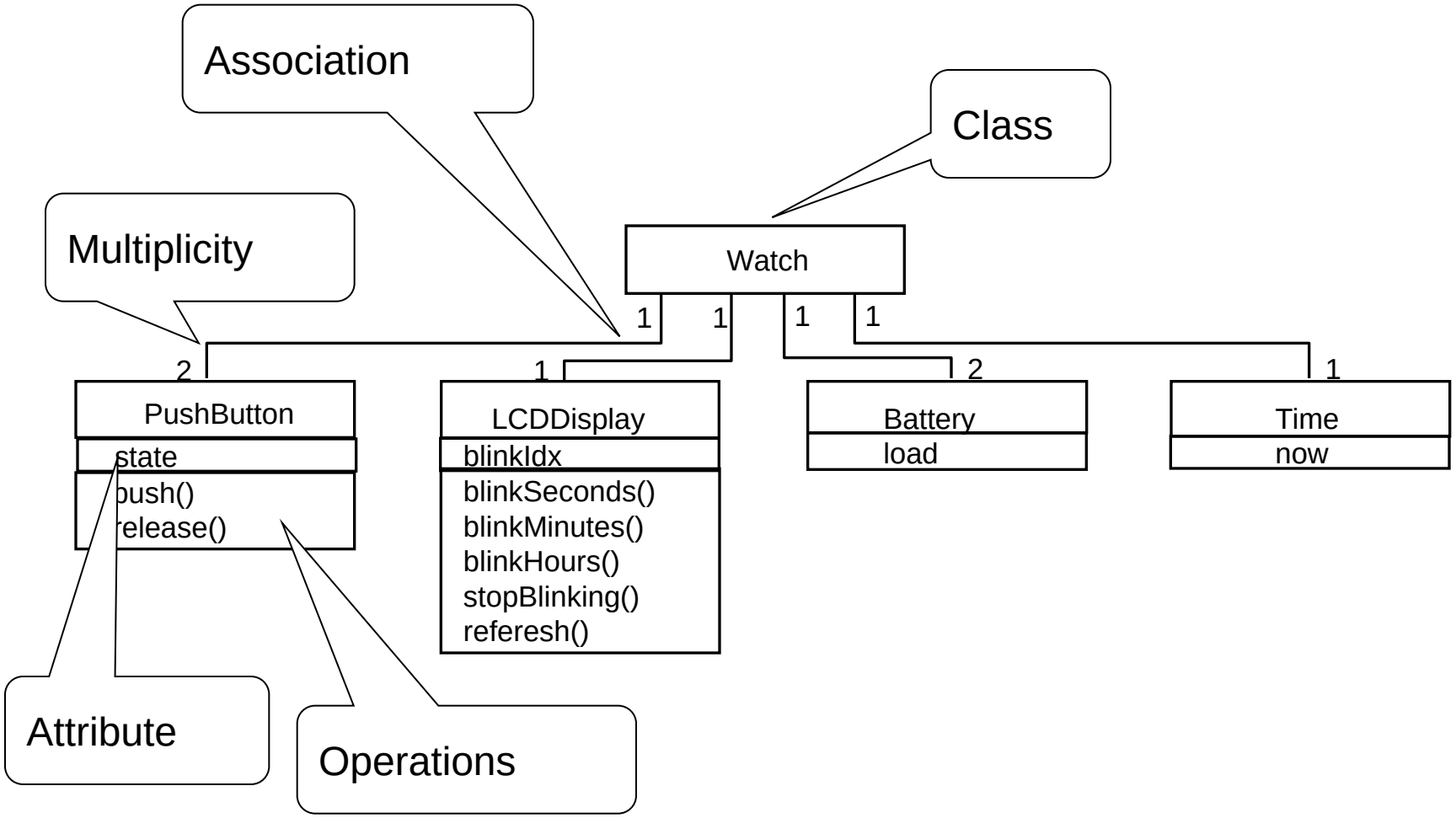
- Electronic watch with a display and two buttons
- Button 1- One press goes to Display hours state
- Button 1- On second press Display minutes state
- Button 1- On third press Display Seconds state
- Button 2- on press increments hours/minutes/seconds depending on state
- Button 1 and 2- simultaneously pressed committ the change.

Watch: Diagrams

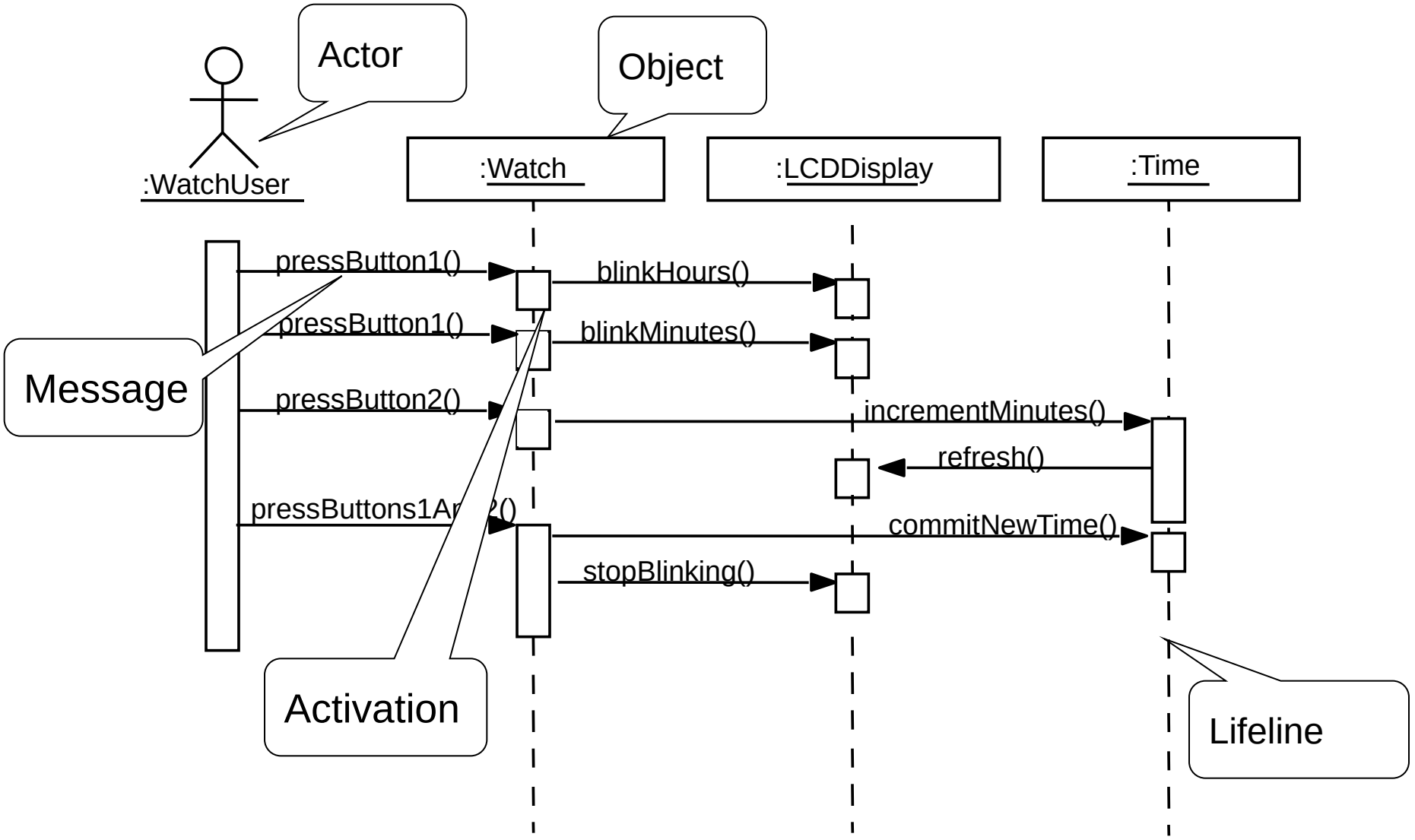
- Electronic watch with a display and two buttons
- Button 1- One press goes to Display Hours state
- Button 1- On second press Display Minutes state
- Button 1- On third press Display Seconds state
- Button 2- on press increments hours/minutes/seconds depending on state
- Button 1 and 2- simultaneously pressed commit the change.

Watch: Class diagrams

Class diagrams represent the structure of the system

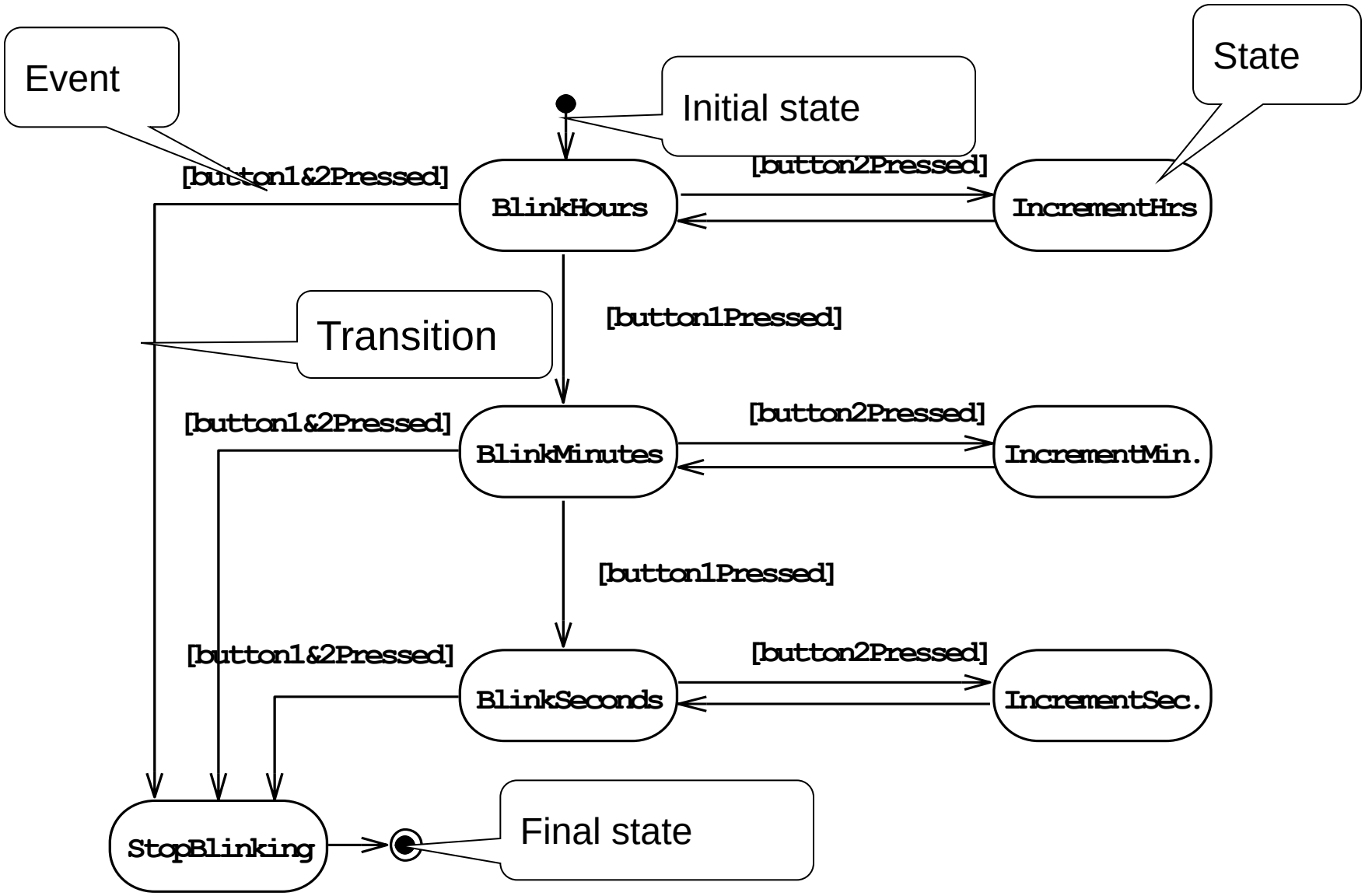


Watch- Increment minutes: Sequence diagram



Sequence diagrams represent the behavior as interactions. Here it is giving details of incrementing minutes in setTime module

Watch: Statechart diagram



Represent behavior as states and transitions

Design Process