EECE 310

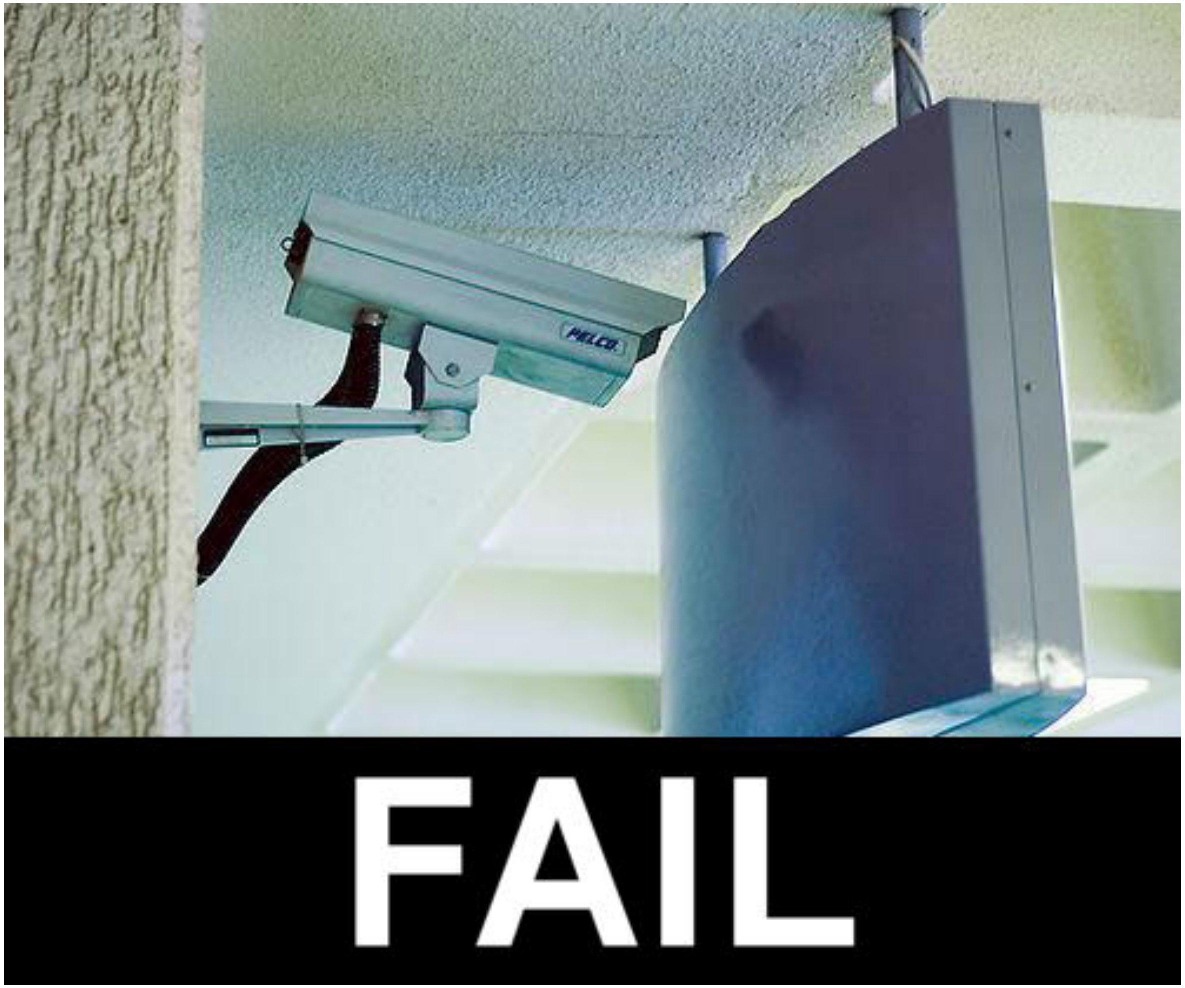
Software Design:

Detailed Design

Two common phases of Software Design

1. Architectural design
2. **Detailed** design

Recap: Bad Design



Port Mann Bridge

Buildup of ice and snow on cables

350 ICBC claims in 2013!

$400,000 worth of glass claims after vehicles were struck by the falling ice bombs.

Solution?

Proposed ideas: deicing, wax, teflon

Selected solution: cable rings

Clears snow as it travels down the cable

Upt0 30 installed at the top, 10 kg each, released with RC

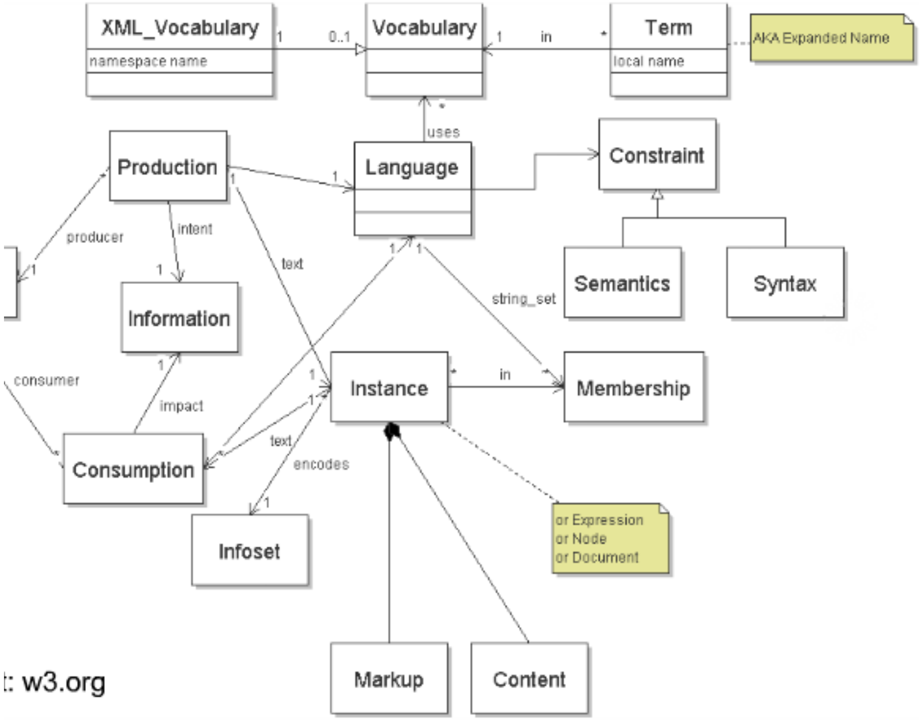
Collected manually from the bottom

Detailed Design

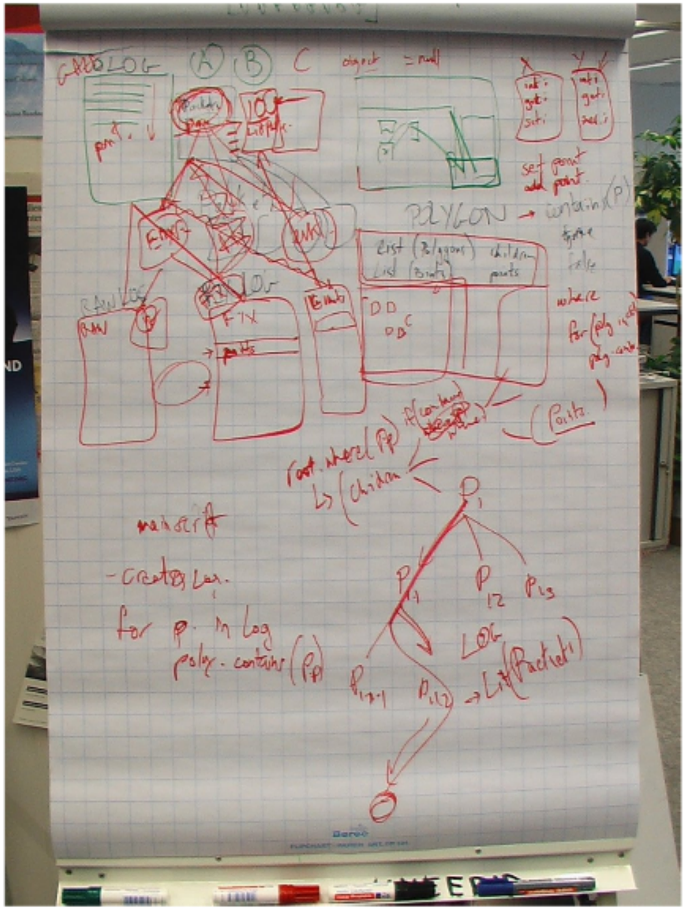
- Concerned with programming concepts
  - Classes, Packages
  - Files
  - …
- Mid-level design
  - Class diagrams (static)
- Low-level design
  - Sequence diagrams (dynamic)

Diagrams

- Vote: Which of these two diagrams is more useful to software developers?



t: w3.org



Diagrams

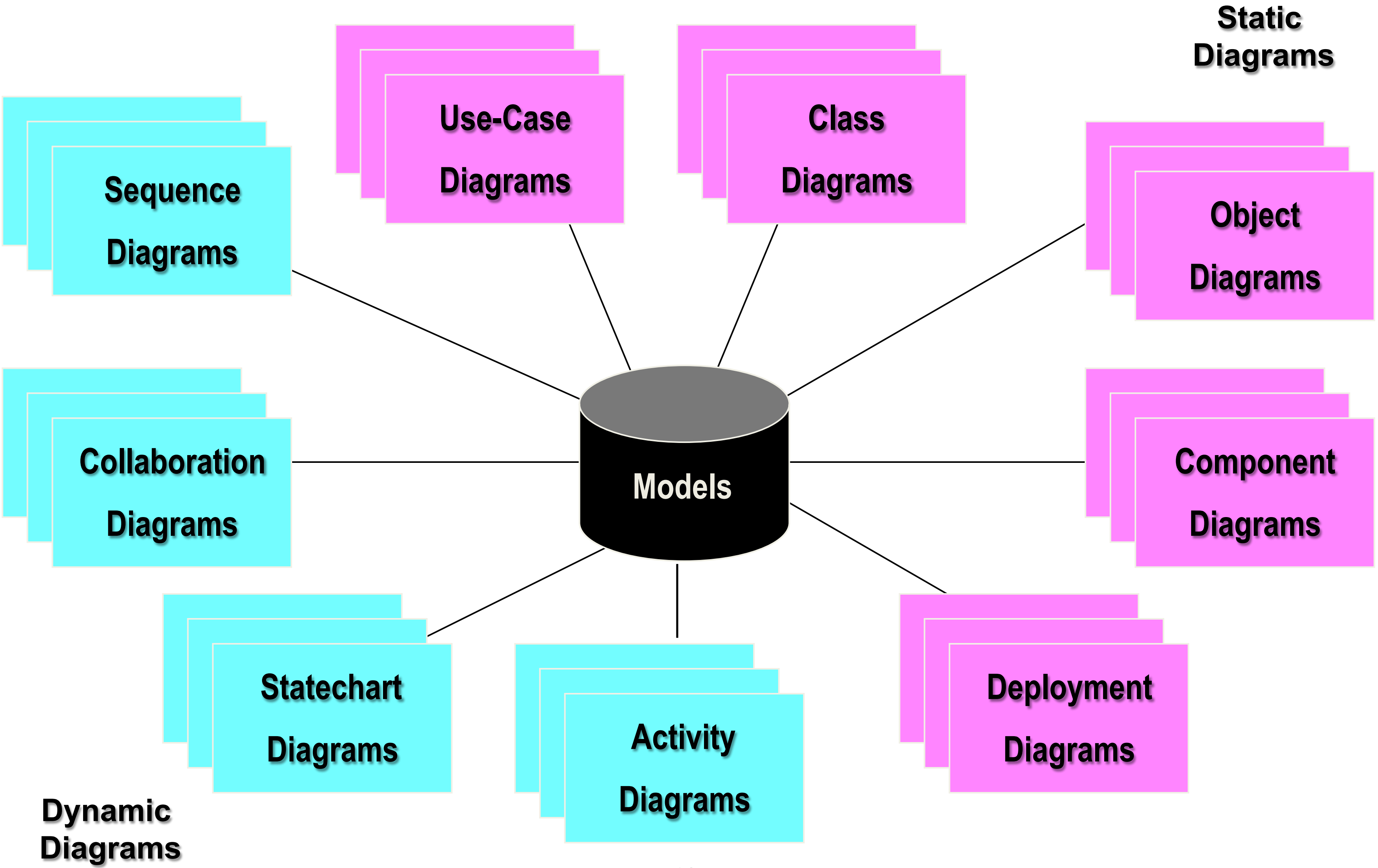
- Diagrams are a ***communication*** tool
  - End product is important, but **discussion** just as important

- In terms of diagrams:
  - Start with draft, hand-written diagrams that can change
  - Towards the end, clean-up and make more readable
  - Use a mutually understood language (a standard: UML)

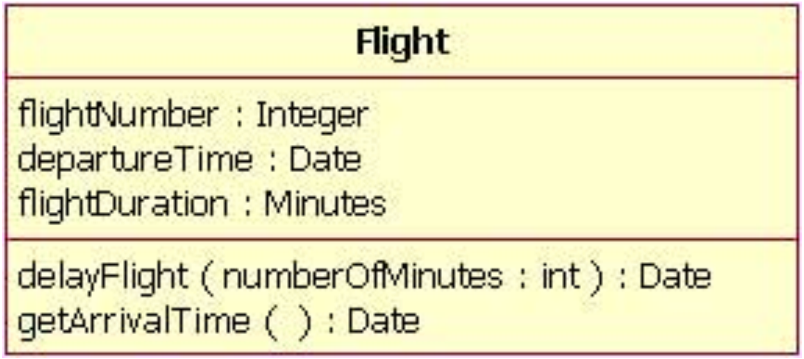
Unified Modeling Language (UML)

- What can we do with it?

Types of UML diagrams



Class Diagrams: the Class



Flight

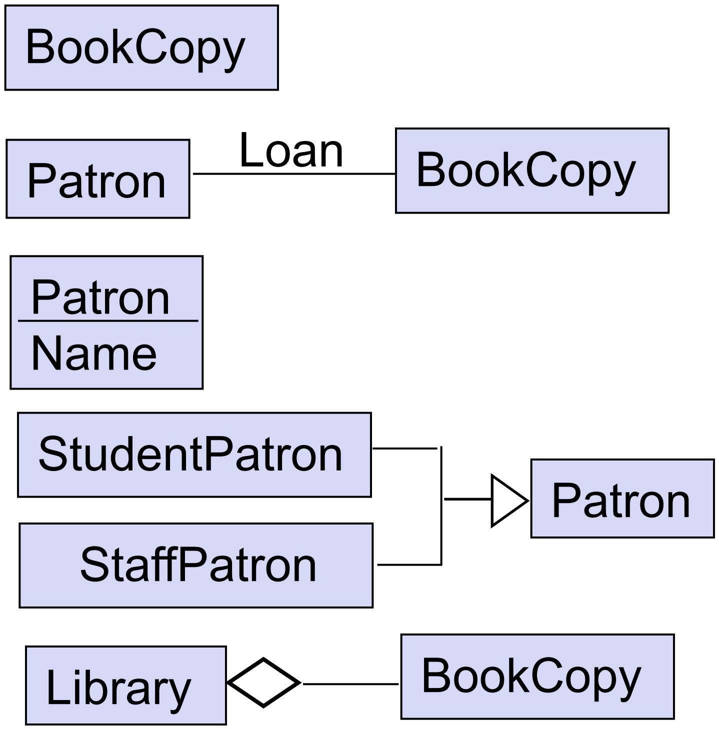
flightNumber : Integer
departureTime : Date
flightDuration : Minutes

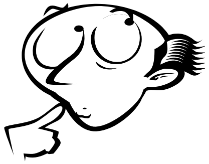
delayFlight (numberOfMinutes : int) : Date
getArrivalTime ( ) : Date

- Class name (*Italics* means abstract)
- Attributes (fields)
  - Name : Type
- Operations (methods)
  - Parameters : Return Type
- Can also be used for interfaces (without fields)

Outline

- What is a conceptual object?
- Entities
- Associations
- Attributes
- Specialization
- Aggregation
- MDE





What is a conceptual object?

Set of instances of a system-specific **concept** ...

- **distinctly identifiable**
  - immutable built-in identity

    e.g. 2 string instances “Justine” are the same,
    but 2 Student instances named Justine are different
- **can be enumerated in any system state**
  - in any state we can list all instances of the Student concept currently involved in the system
- **share similar features**
  - common **name**, **definition**, **type**, **domain properties**,
  - common **attributes**, **associations**: see details later

    e.g. *Email* attrib of Student; *Loan* assoc linking Student and BookCopy
- may differ in their **individual states** and state **transitions**

state of an instance of a conceptual object

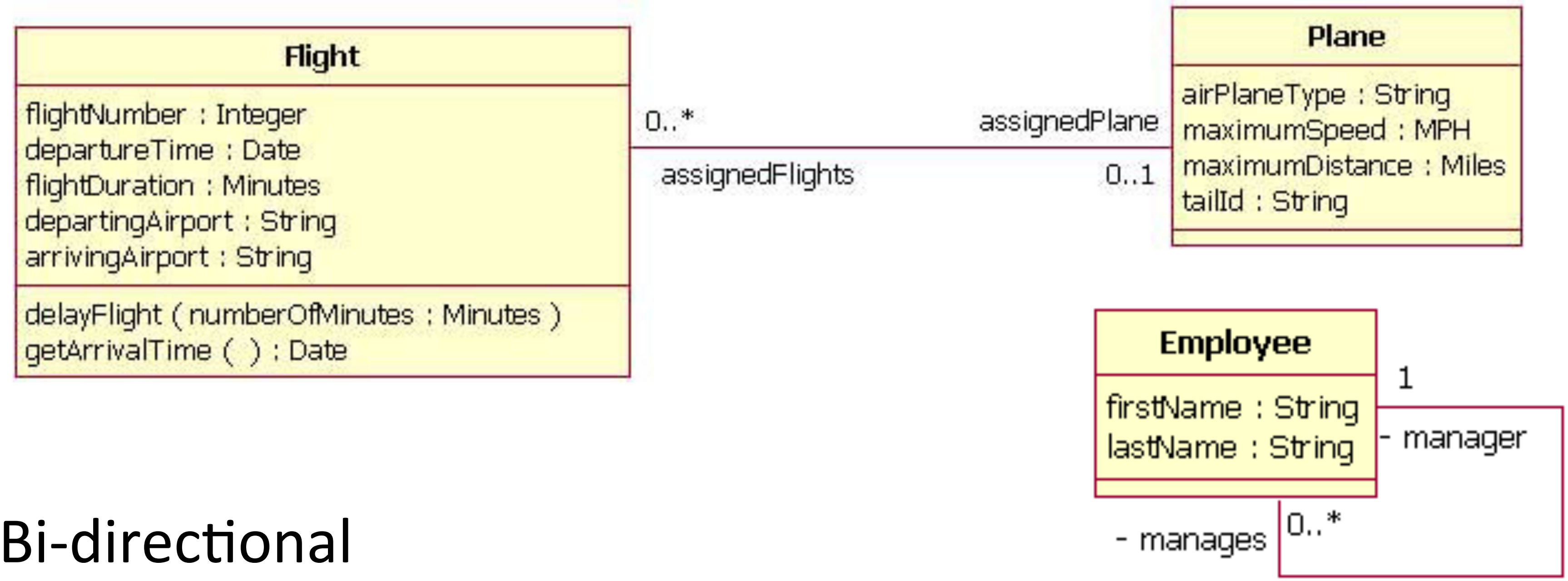
- E.g. instance *tr* of Train might be in state:

(*tr.Speed* ↦ *0*, *tr.Location* ↦ *9.25*, *tr.DoorsState* ↦ *Open*, *On* ↦ *(tr, block13)*, *At* ↦ *(tr, platform1)*)

Types of conceptual object

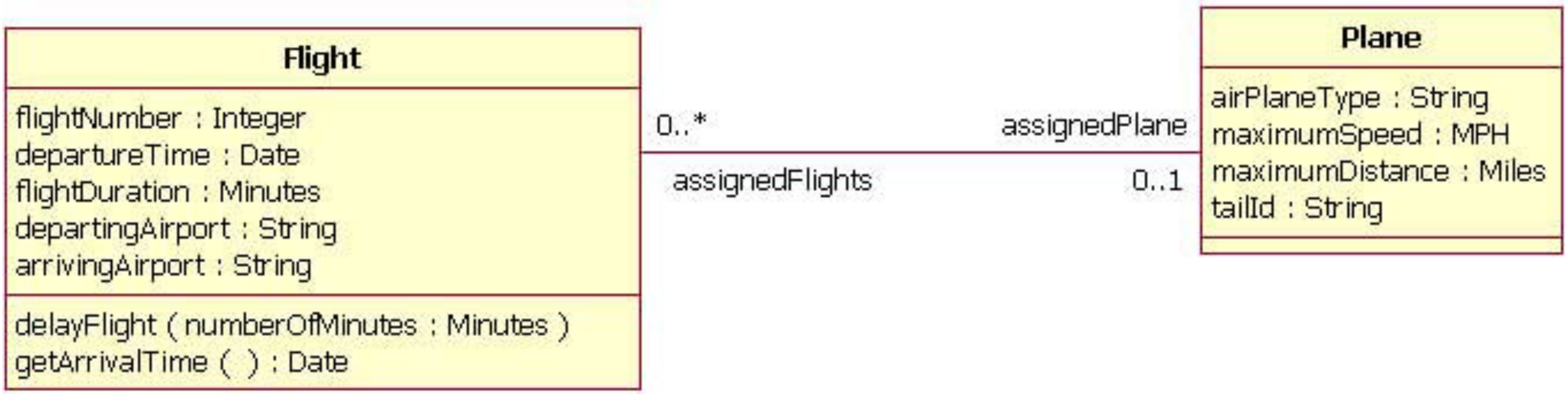
- **Entity**: autonomous, passive object
  - instances may exist in system independently of instances of other objects
  - instances cannot control behavior of other objects
  - e.g. Book, Train, Platform, ...
  - represented as **UML class**
- **Association**: object dependent on objects it links
  - instances are conceptual links among object instances
  - e.g. Loan linking Student & BookCopy
    At linking Train & Platform
  - represented as **UML association**

Association



- Bi-directional
  - **Both classes are aware of each other**
- Role
  - Usually maps to a field name
- Multiplicity
  - Indicates how many instances can be linked (*i.e.* a list of…)

Association: In Java?



Flight

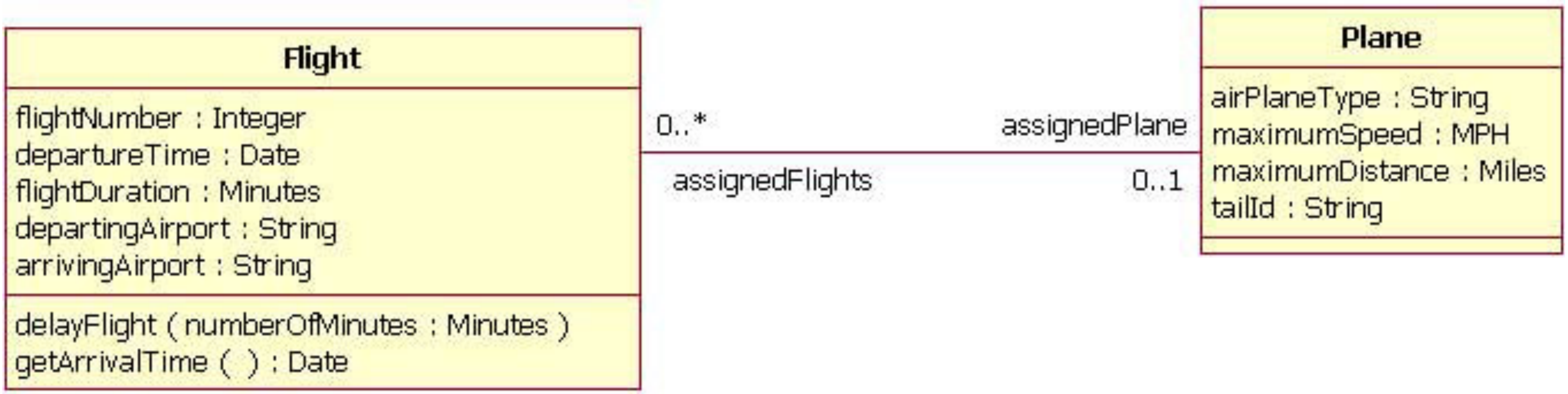
flightNumber : Integer
departureTime : Date
flightDuration : Minutes
departingAirport : String
arrivingAirport : String

delayFlight (numberOfMinutes : Minutes)
getArrivalTime ( ) : Date

Plane

airPlaneType : String
maximumSpeed : MPH
maximumDistance : Miles
tailId : String

Association: In Java



Flight

flightNumber : Integer
departureTime : Date
flightDuration : Minutes
departingAirport : String
arrivingAirport : String

delayFlight (numberOfMinutes : Minutes)
getArrivalTime ( ) : Date

0..*

assignedFlights

assignedPlane

0..1

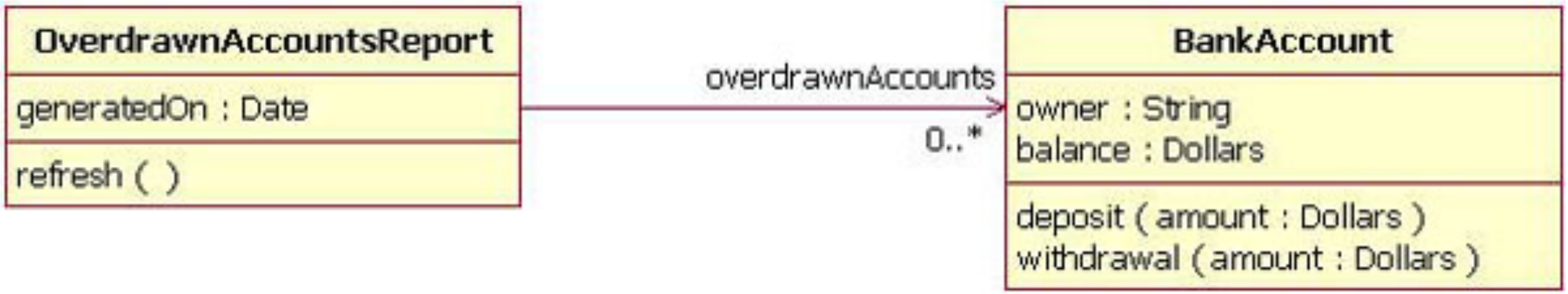
Plane

airPlaneType : String
maximumSpeed : MPH
maximumDistance : Miles
tailId : String

```
public class Flight {
   private Plane assignedPlane;
   ....
}
```

```
public class Plane {
  private Collection<Flight> assignedFlights;
 …
}
```

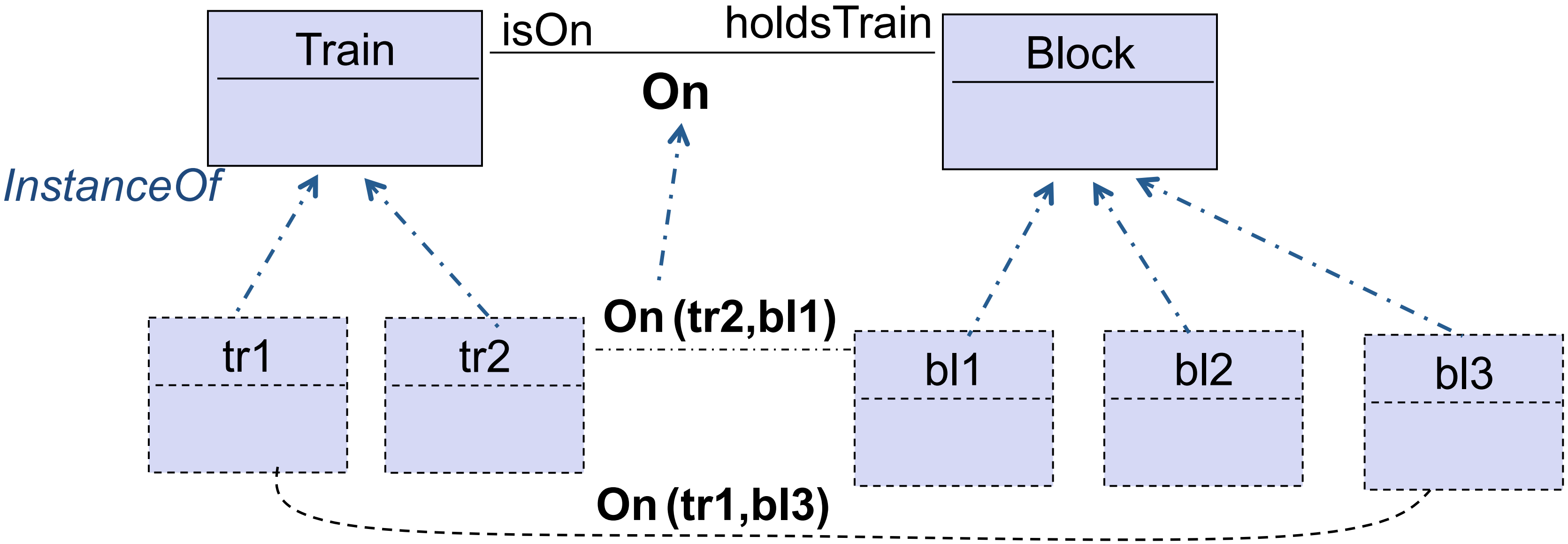
Uni-directional Association



- Only one class knows of the other
- Role
  - Only in one direction
- Multiplicity
  - Only on one end (BankAccount doesn't know report)

Association instances

- linked object instances at *runtime*

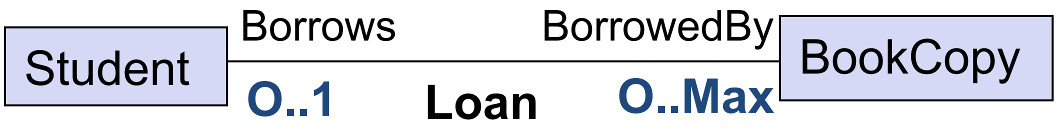


Multiplicities of n-ary association

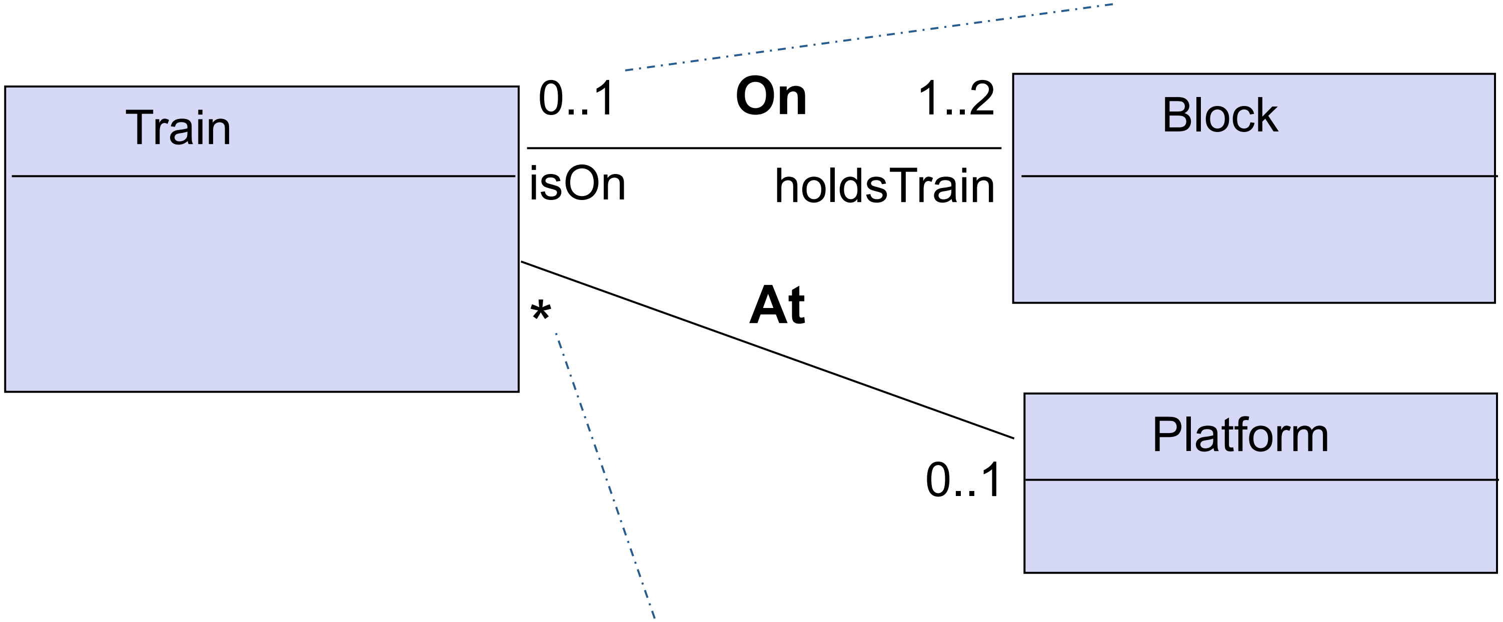
- **min/max** number of linked **target** instances
  - attached to role of **target** instance
- For binary associations, express standard constraints ...
  - min = 0: optional link (possibly no link in some states)
  - min = 1: mandatory link (at least one link to target in any state)
  - max = 1: uniqueness (at most one link to target in any state)
  - max = *: arbitrary number *N* of target instances linked to source instance, in any state (*N* > 0)

Notation: "n..m" for min = n, max = m

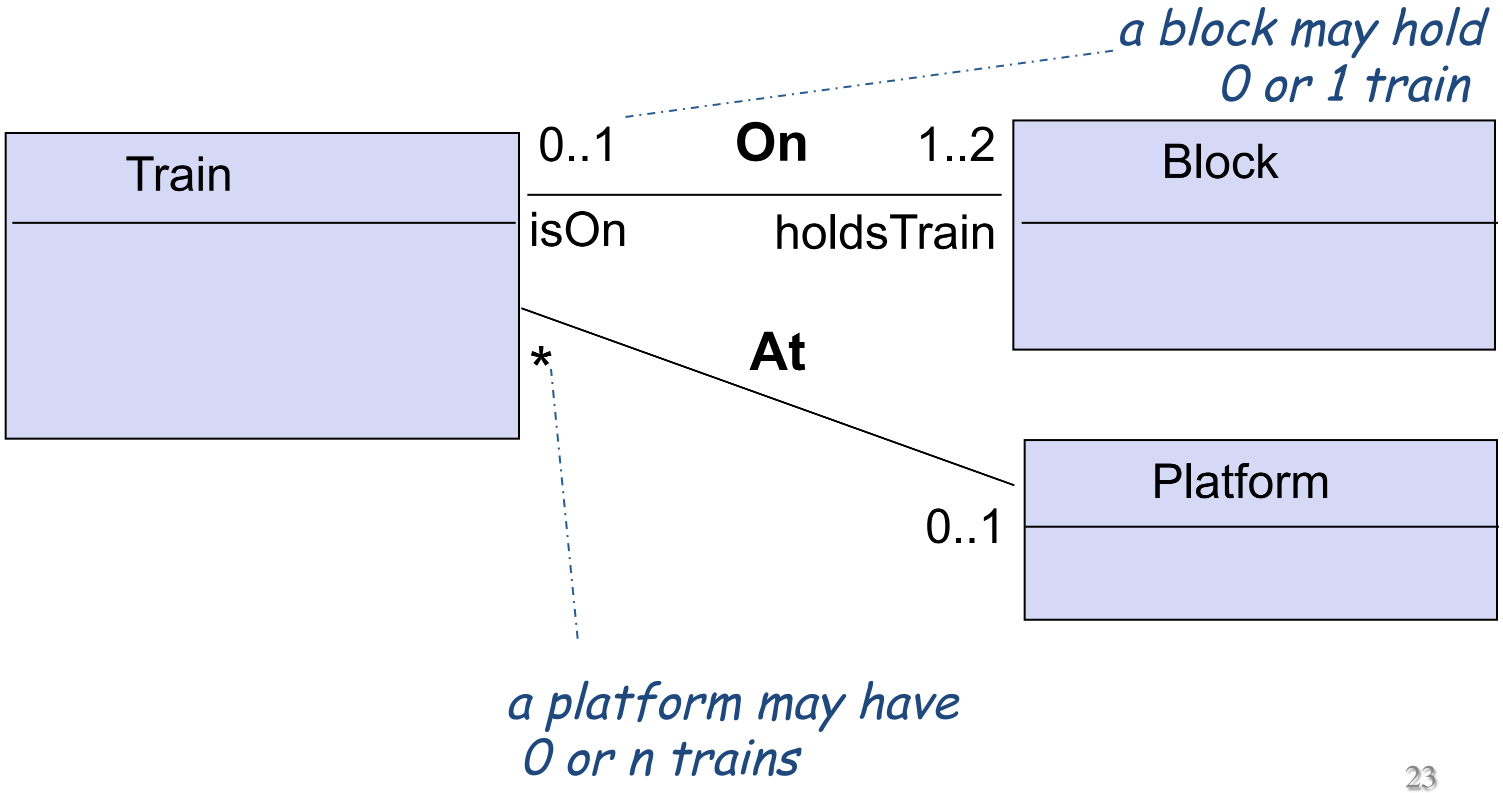
"*" for "0..*"



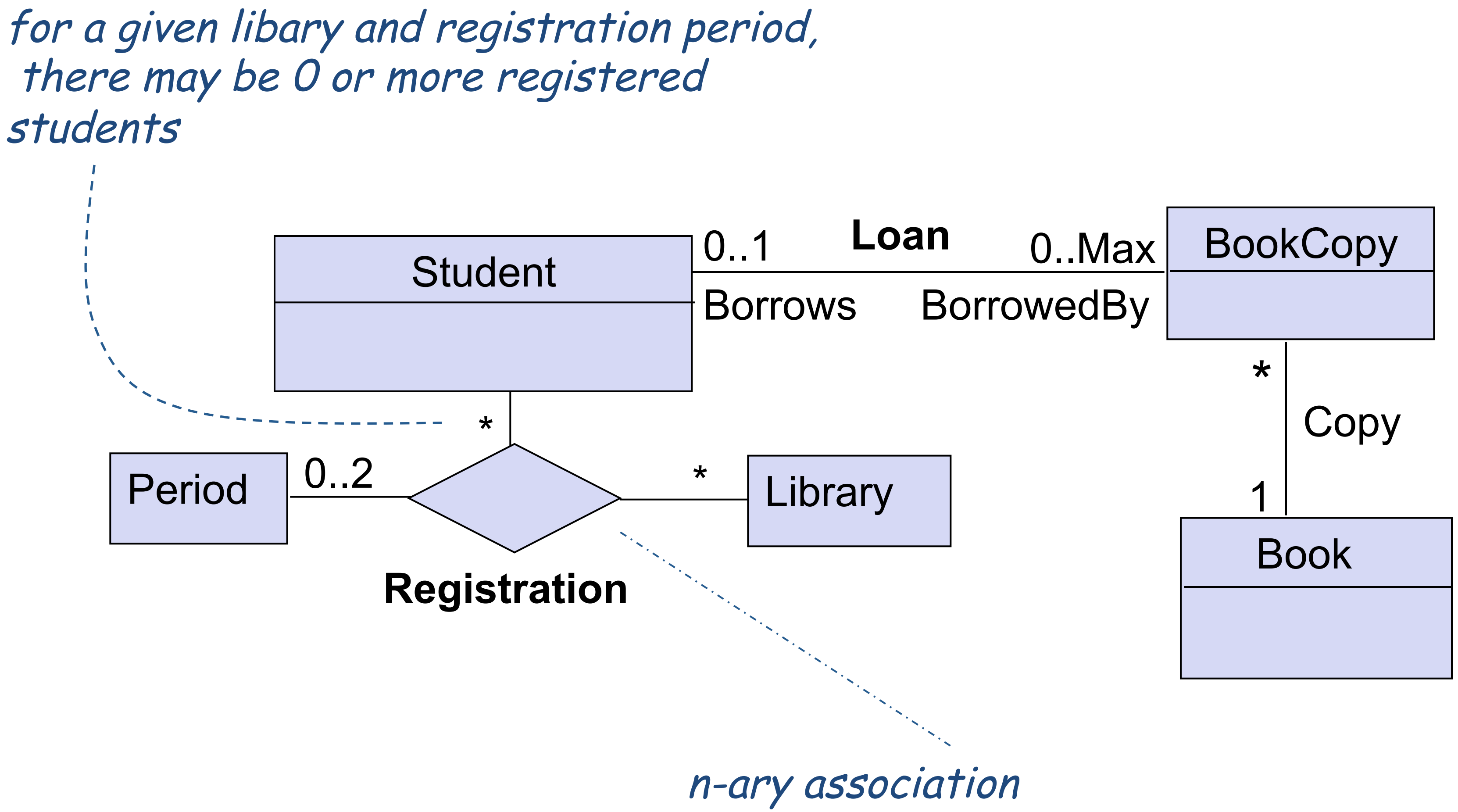
What do the associations mean?



What do the associations mean?

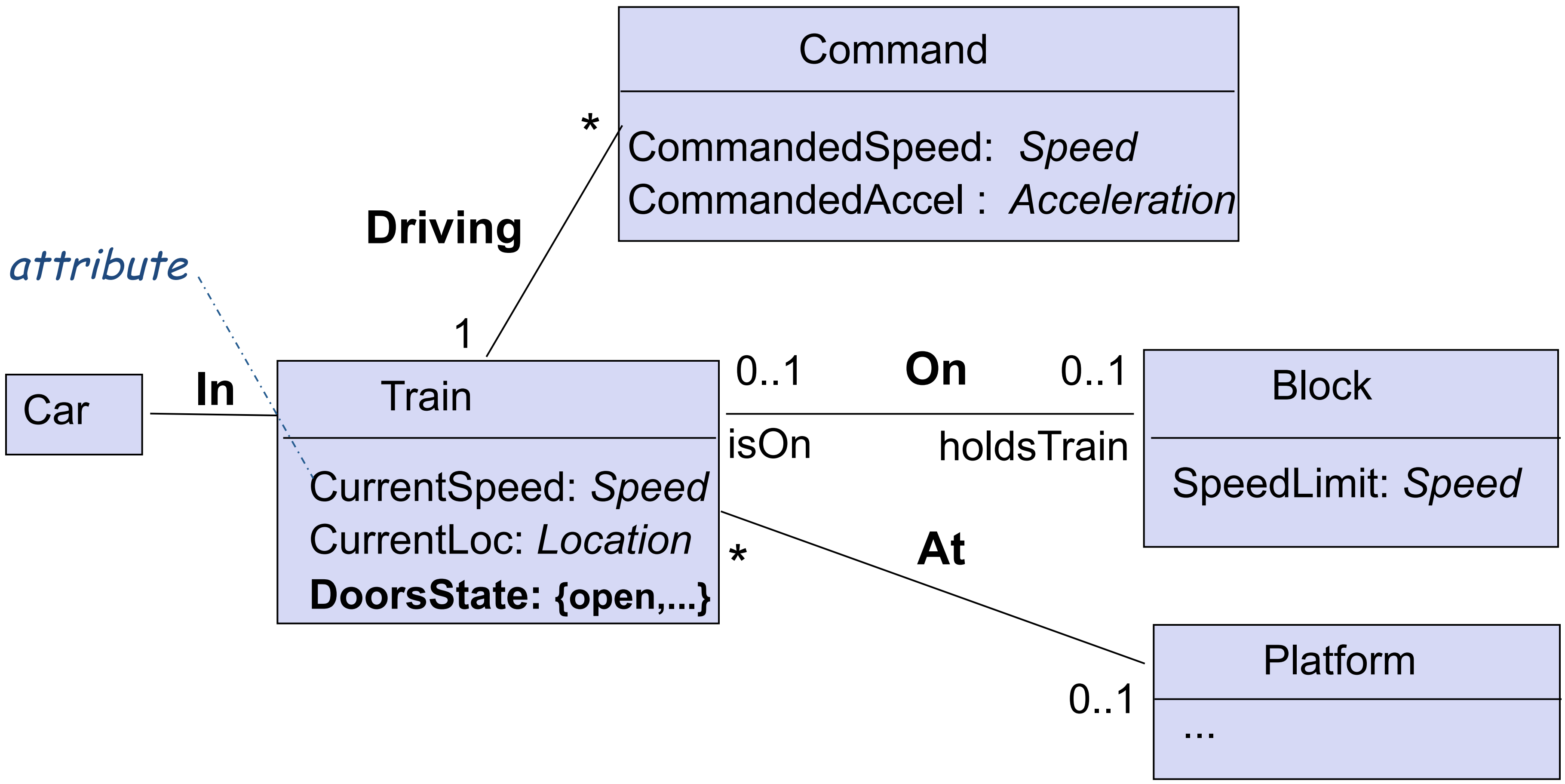


N-ary associations

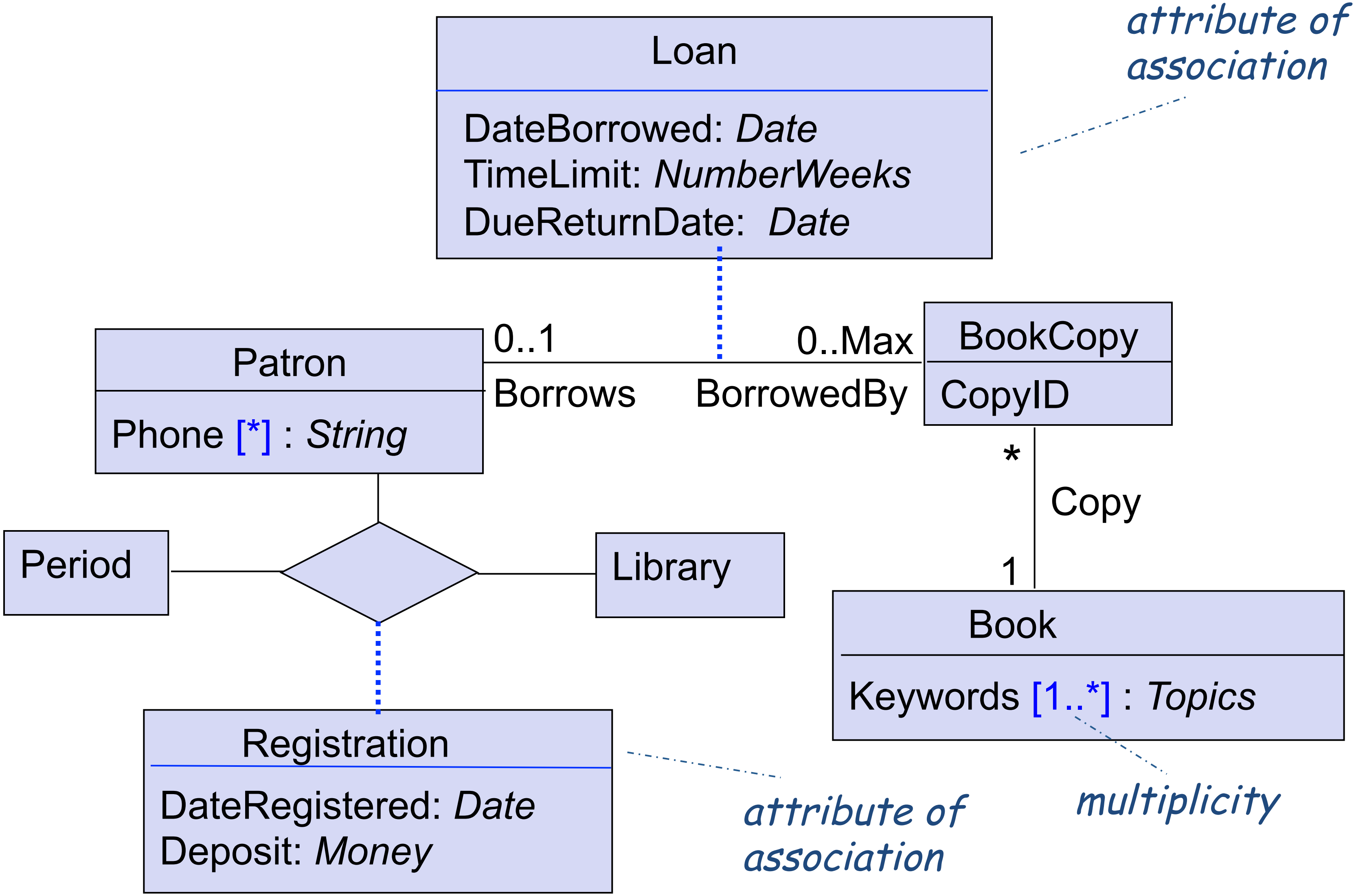

for a given libary and registration period, there may be 0 or more registered students

n-ary association

Attributes in UML



Association attributes in UML



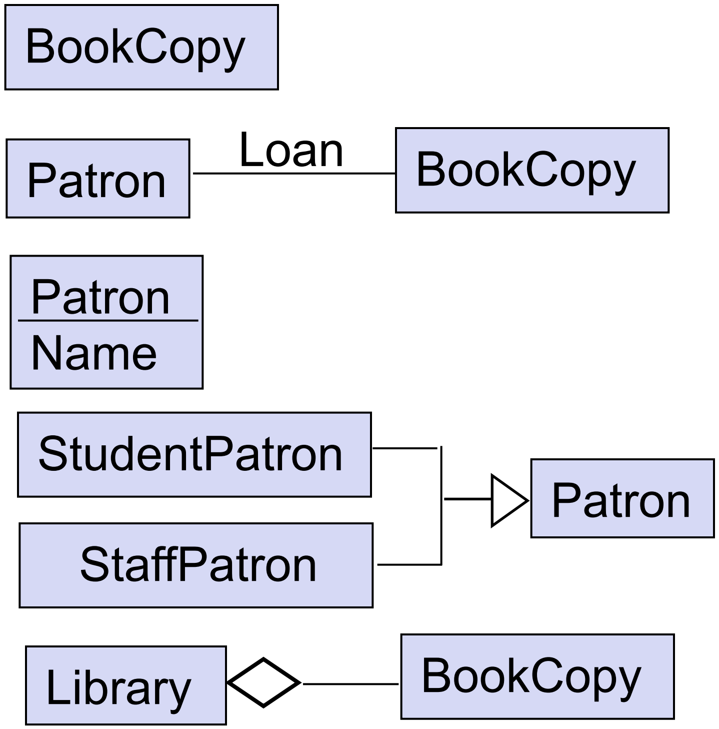
attribute of association

attribute of association

multiplicity

Outline

- What is a conceptual object?
- Entities
- Associations & multiplicities
- Attributes
- **Specialization**
- **Aggregation**
- **More on class diagrams**
  - **derived attributes, OR-associations, associations of associations**
- Building object models: heuristic rules



Built-in associations for structuring object models

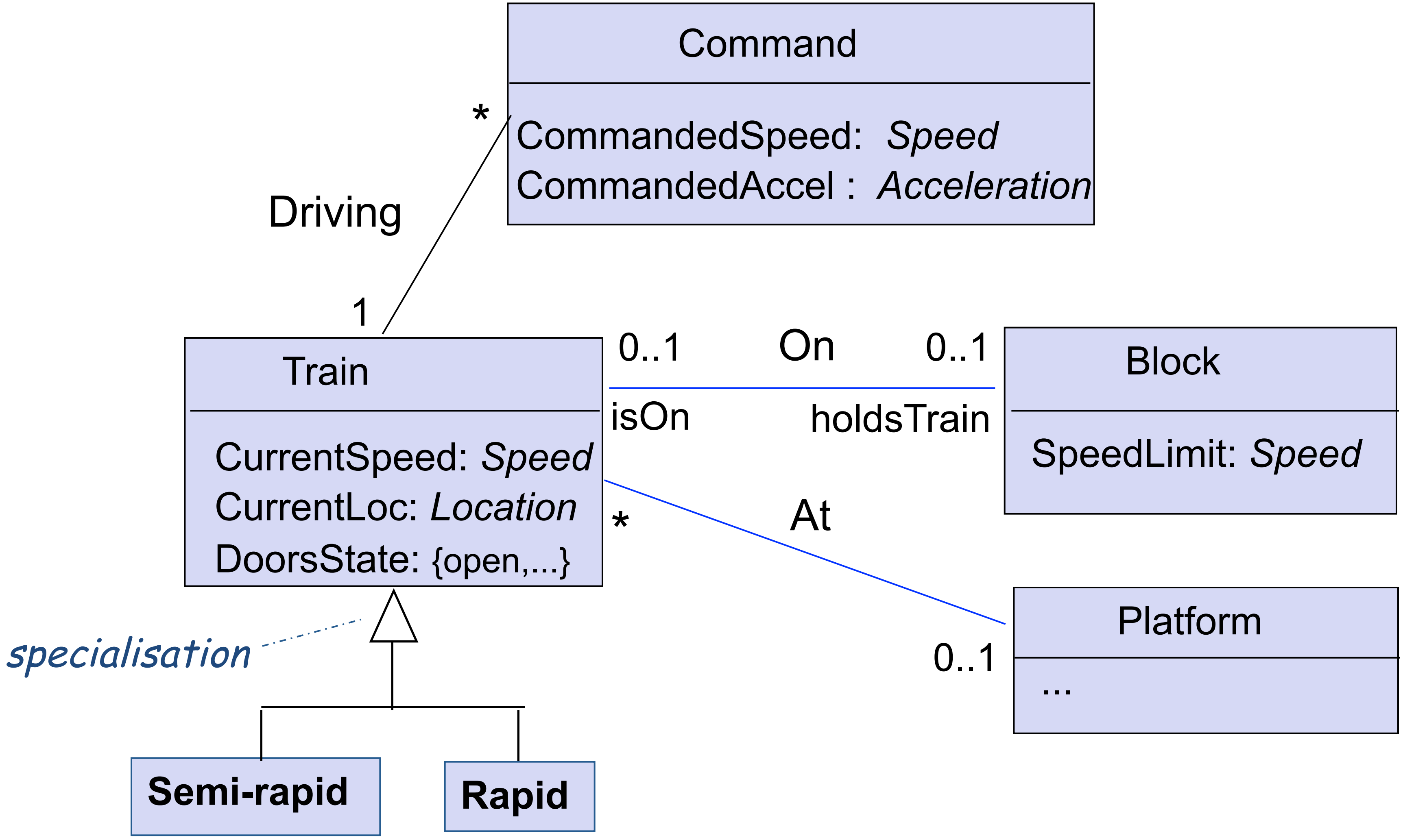
- **Specialization** = sub-classing: object *SubOb* is a specialization of object *SuperOb* **iff** for any individual *o*:

  InstanceOf (*o*, SubOb) ⇒ InstanceOf (*o*, SuperOb)

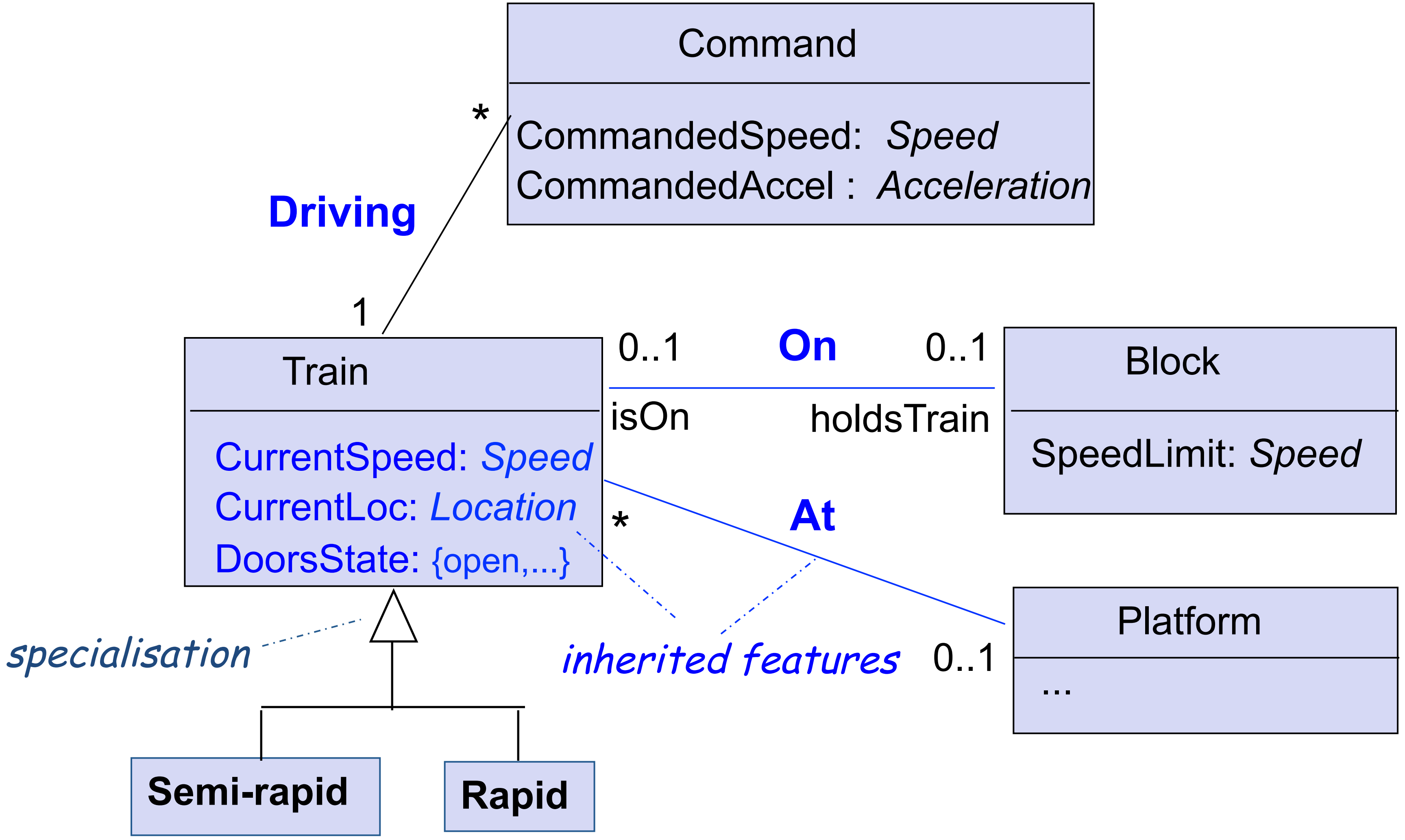
  - SubOb **specializes** SuperOb, SuperOb **generalizes** SubOb

- **Feature inheritance** as a consequence ...
  - by default, *SubOb* inherits from *SuperOb* all its **attributes**, **associations**, while has its own distinguishing features
  - may be inhibited by compatible redefinition of feature with same name within specialized SubOb (“override”)

What is inherited in Rapid?



What is inherited in Rapid?



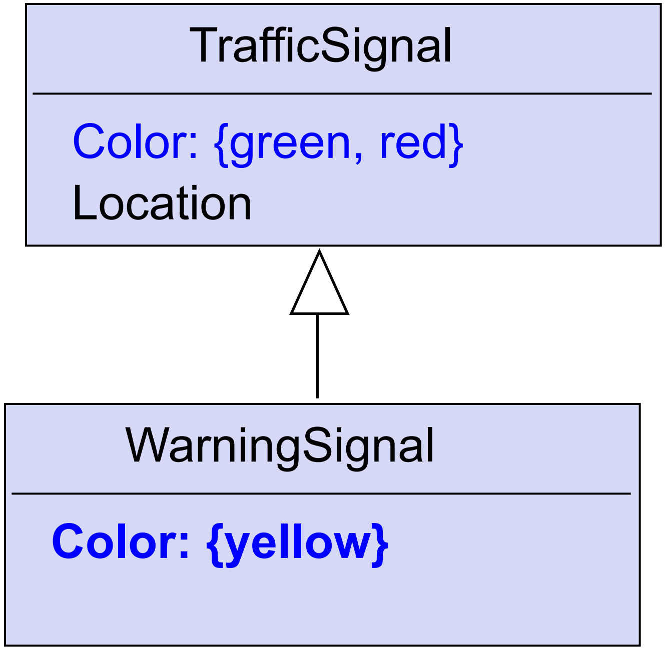
Question: Do subclasses inherit private fields in Java?

- A subclass does **not** inherit the private members of its parent class. However, if the superclass has **public** or **protected** methods for accessing its private fields, these can also be used by the subclass.

- The private fields are there in an object of the subclass, just not directly accessible.

Inhibiting inheritance

Color of an instance of WarningSignal?



TrafficSignal

Color: {green, red}

Location

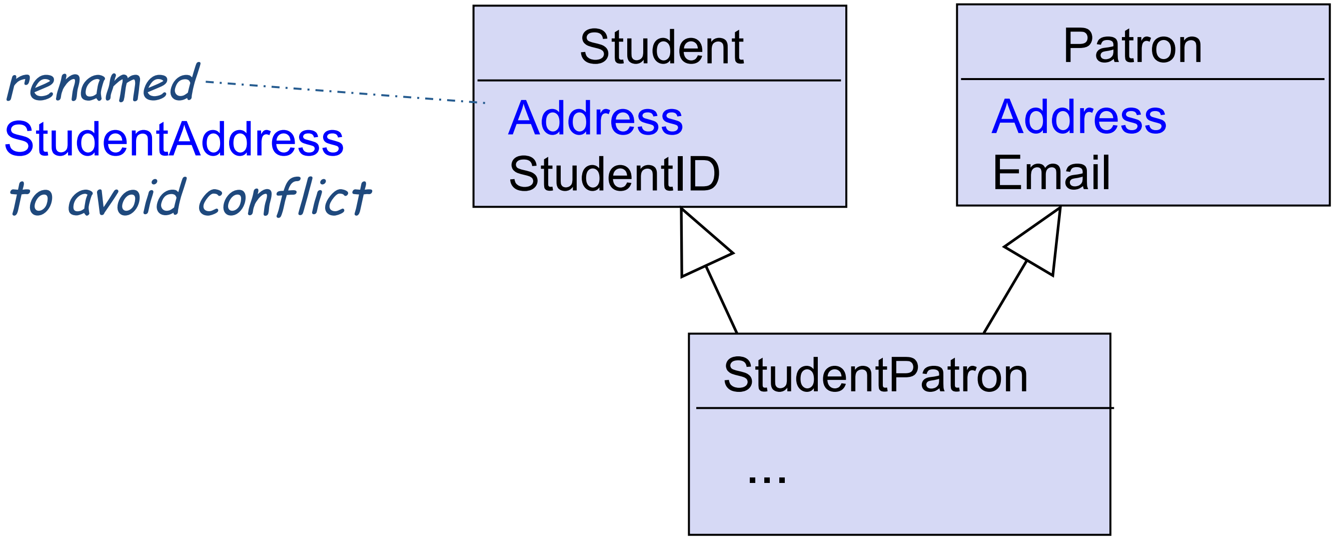
WarningSignal

Color: {yellow}

The more specific feature always **overrides** the more general one

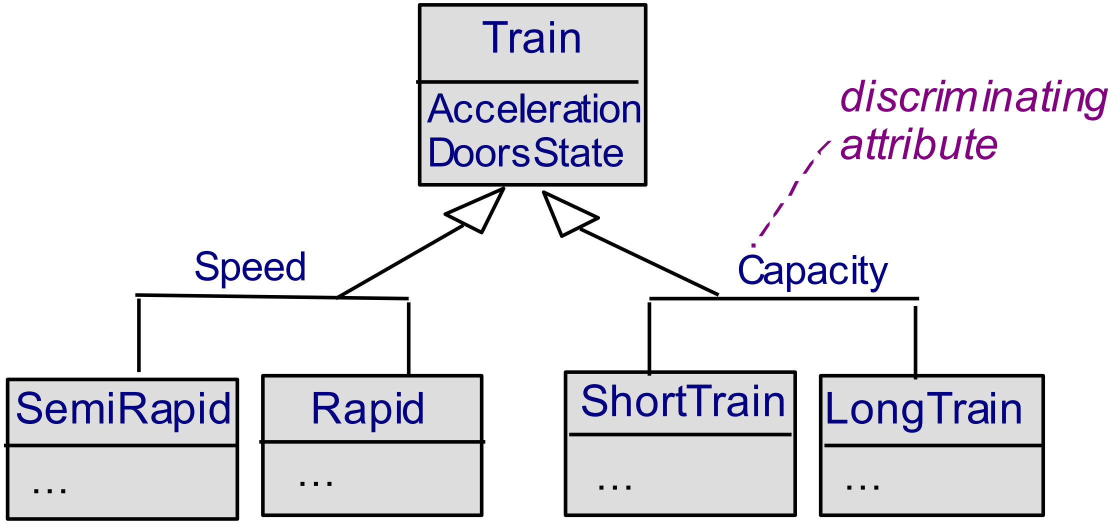
Multiple inheritance

- Same object may be specialization of multiple super-objects
  - by default, inheritance of all features from all super-objects
- Can result in inheritance conflicts
  - different features with same name inherited from different super-objects
  - conflicting features first renamed to avoid this
- C++, Python, Scala (but not Java)

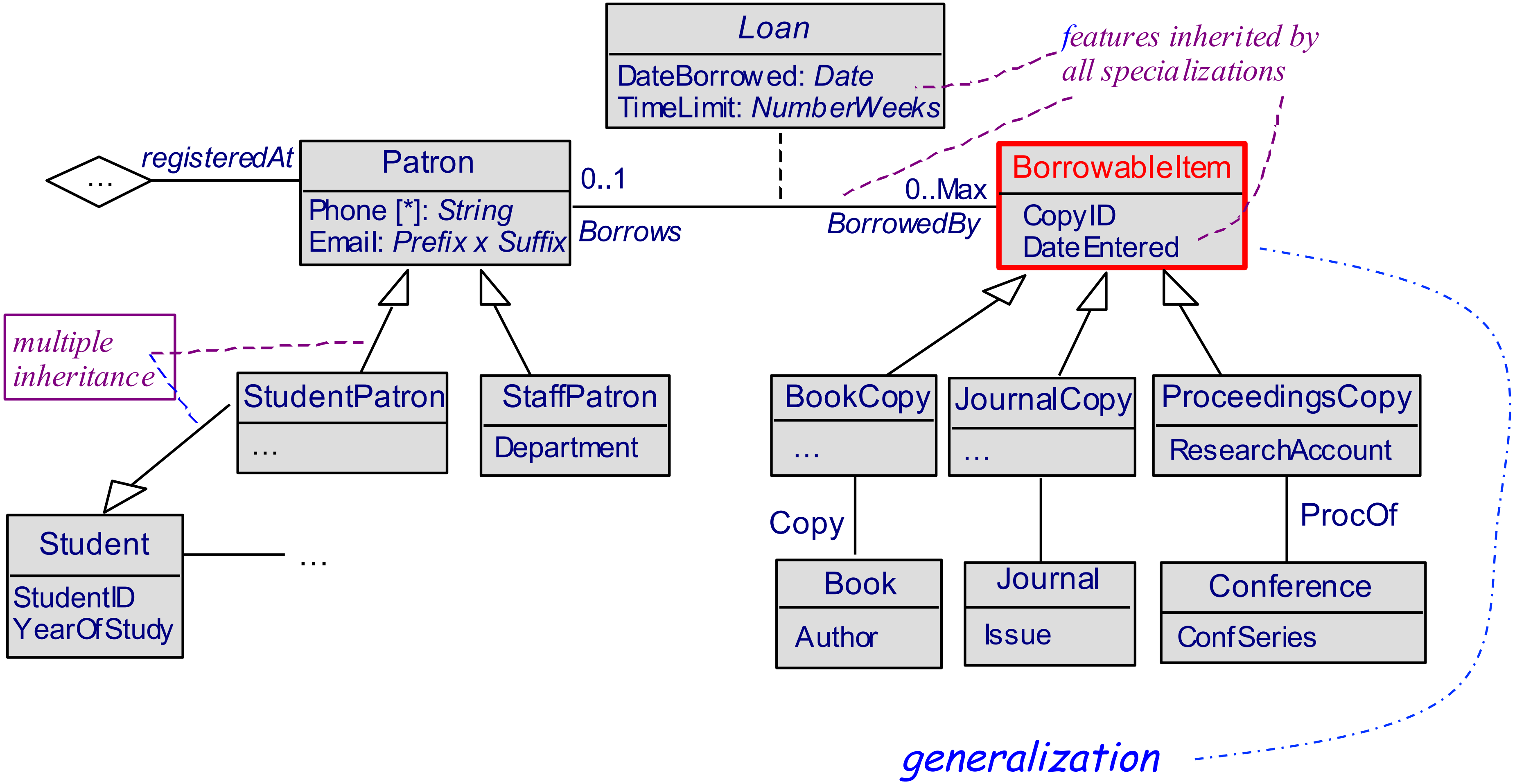


Multiple specializations

- Same object may have multiple specializations
  - Different subsets of object instances associated with different criteria
  - Same object instance may be member of different subsets (one per criterion)
- **Discriminator** = attribute of super-object whose values define different specializations (differentiation criterion)



Object generalization

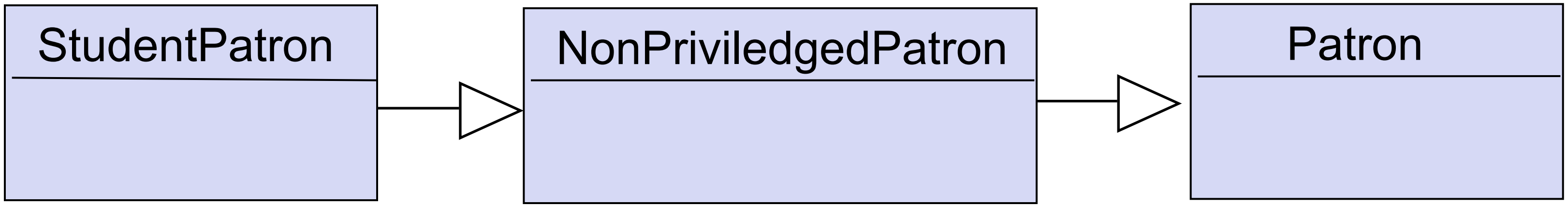


Benefits of generalization-based structuring

- Common features in multiple objects are factored out into single generalized object

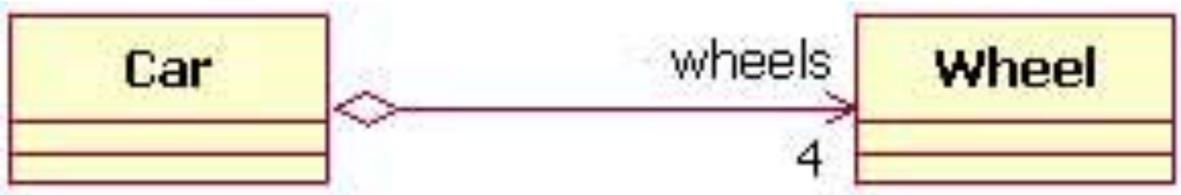
  => simpler model, no duplication

- Generalized objects & their structure are reusable in different contexts & systems (by specialization)
  - e.g. BorrowableItem --> CDCopy , VideoCopy
- Increased modifiability of large models
  - modifications of more general features are localized in more general objects, down-propagated to specialized objects

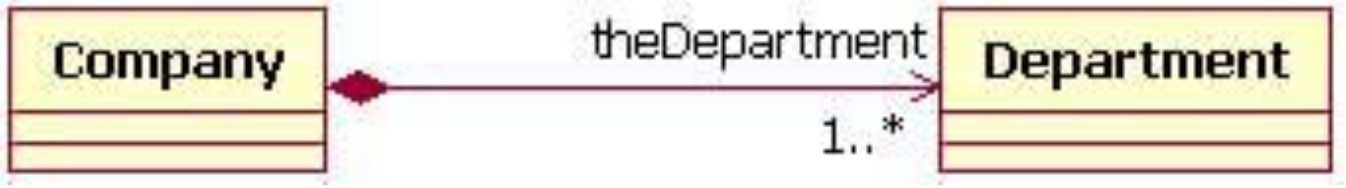


Aggregation

- An advanced type of association
- The contained object is *part* of the container
- Two types:
  - Aggregation: wheels can outlive cars

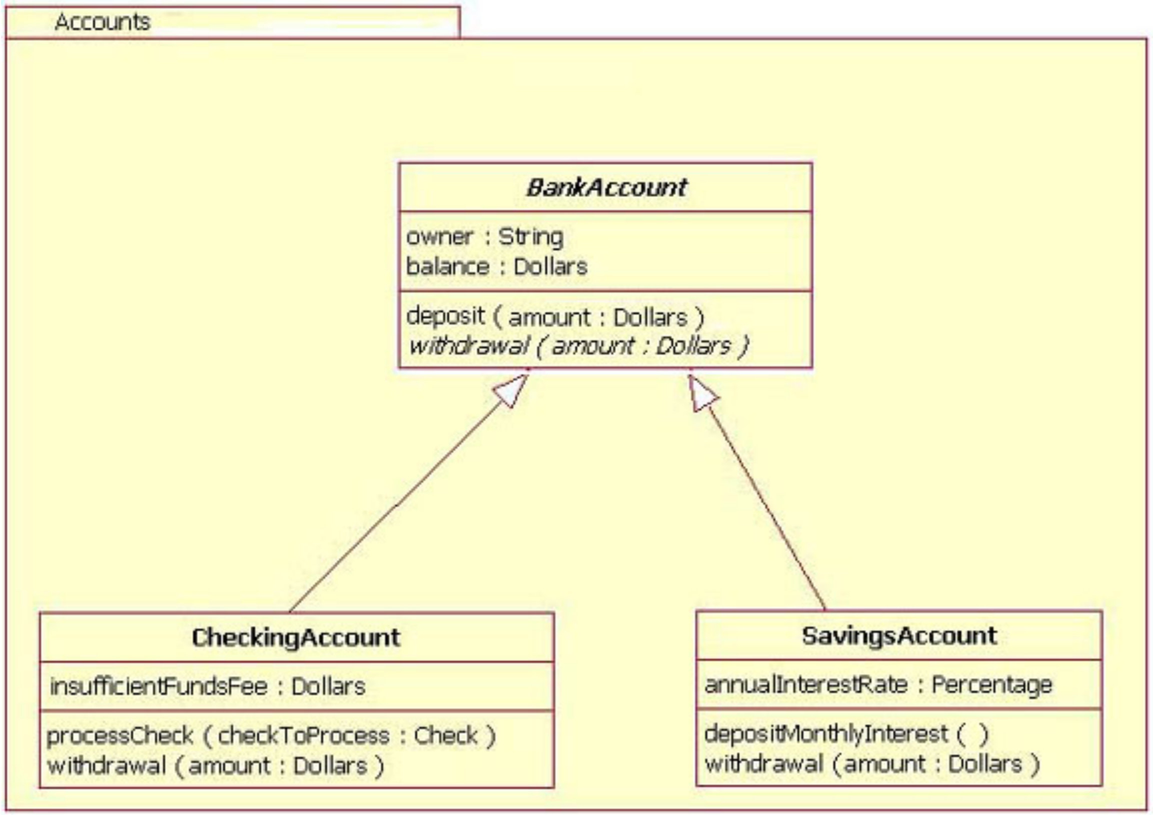


  - Composition: department's life depends on company



Class Diagrams: Packages

- Group classes together:
    - UI classes together in a package


Accounts
BankAccount
owner : String
balance : Dollars
deposit (amount : Dollars)
withdrawal (amount : Dollars)
CheckingAccount
insufficientFundsFee : Dollars
processCheck (checkToProcess : Check)
withdrawal (amount : Dollars)
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal (amount : Dollars)

Class Activity

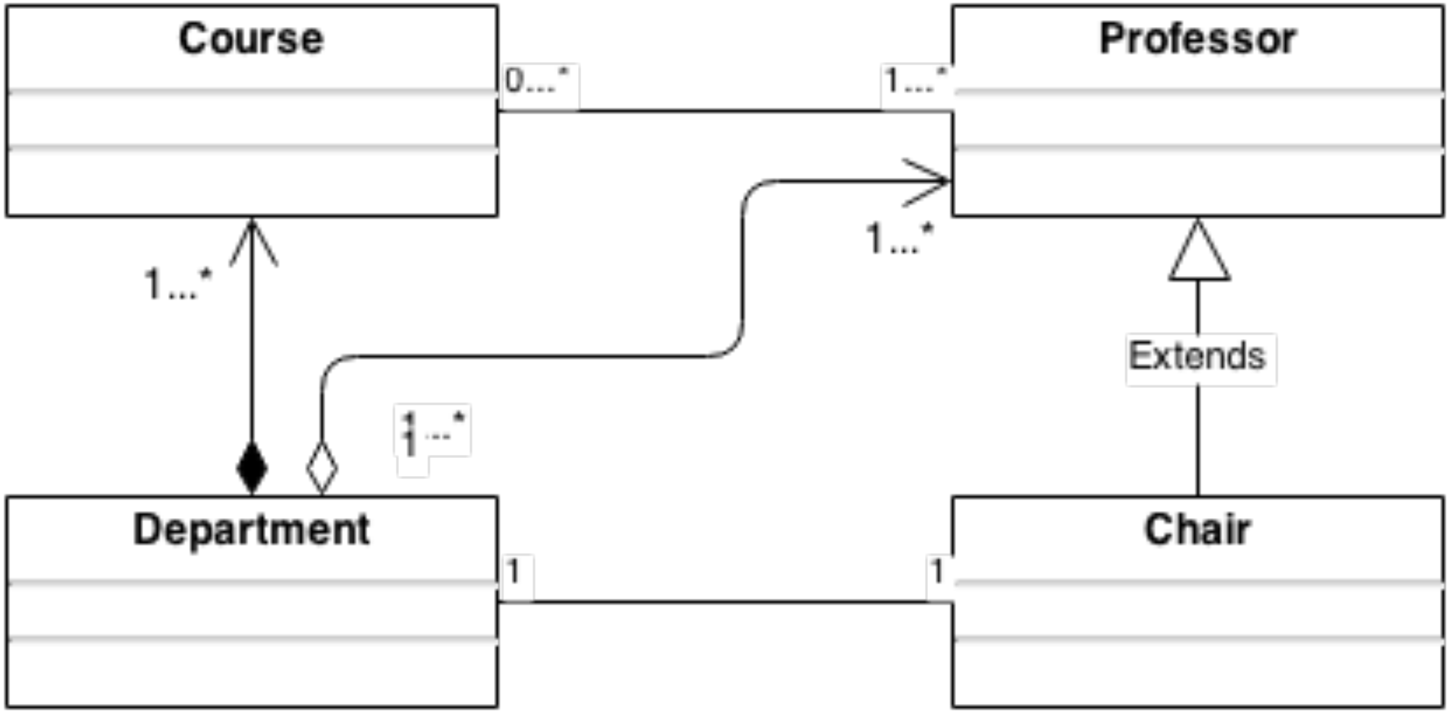
For your groups, draw a **class diagram** for

- The structure of APSC departments in terms of: *professors*, *departments*, dept. *chairs (AKA heads)*, and *courses*..

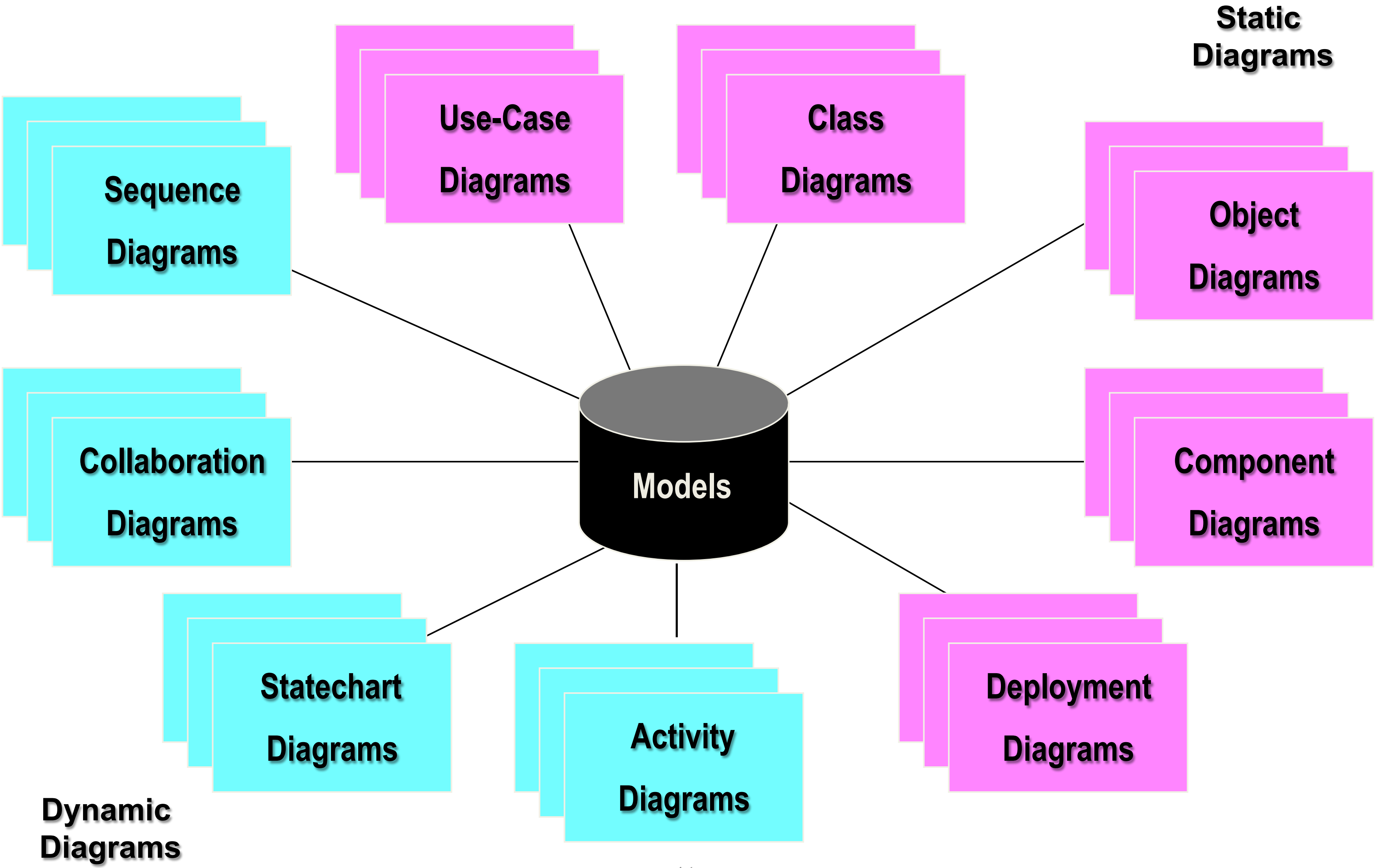
- Think about: **inheritance**, **aggregation/ composition**, **associations**, and **multiplicities**!

- Hand in your design!

Solution

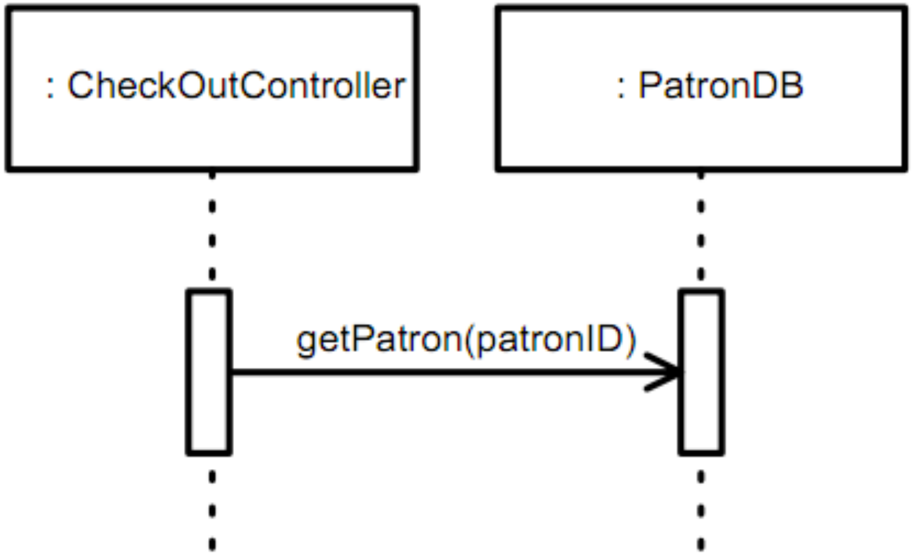


Types of UML diagrams



Sequence Diagrams

- What is a sequence diagram?
- In a sequence diagram, what does a box depict? What does a vertical dashed line depict?
- What does an arrow between boxes depict?

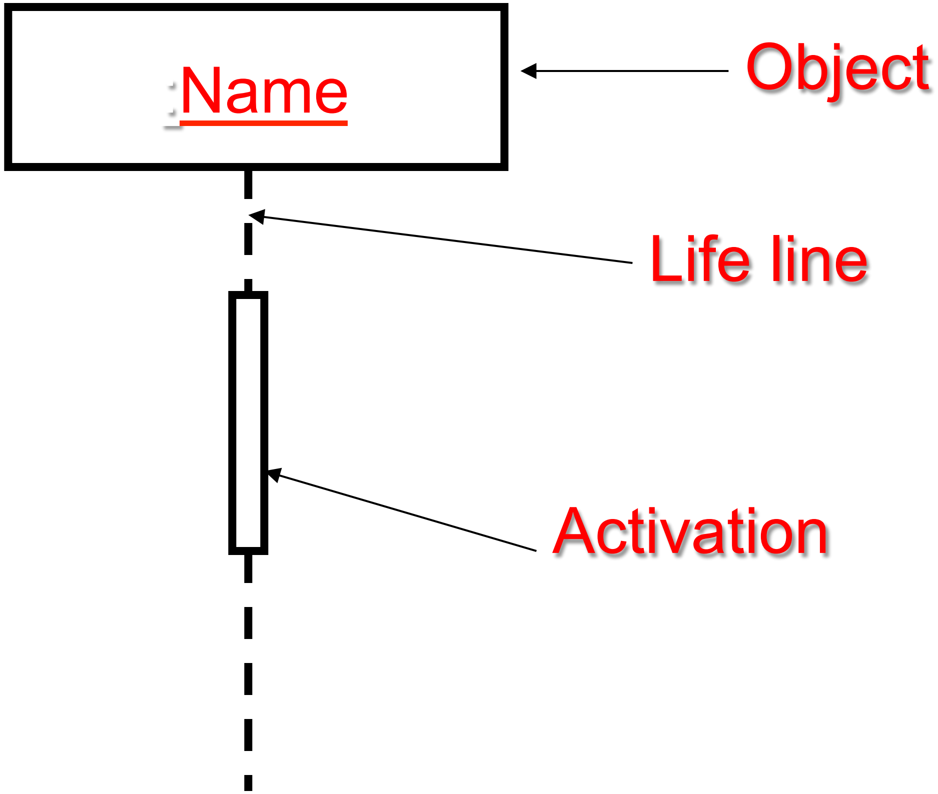


Sequence Diagrams

- Low-Level design tool
- Used to describe sequences of invocations between the objects that comprise the system
  - Focus less on *type of messages*, more on the *sequence* in which they are received
- Elements of UML sequence diagrams:
  - Lifelines
  - Messages

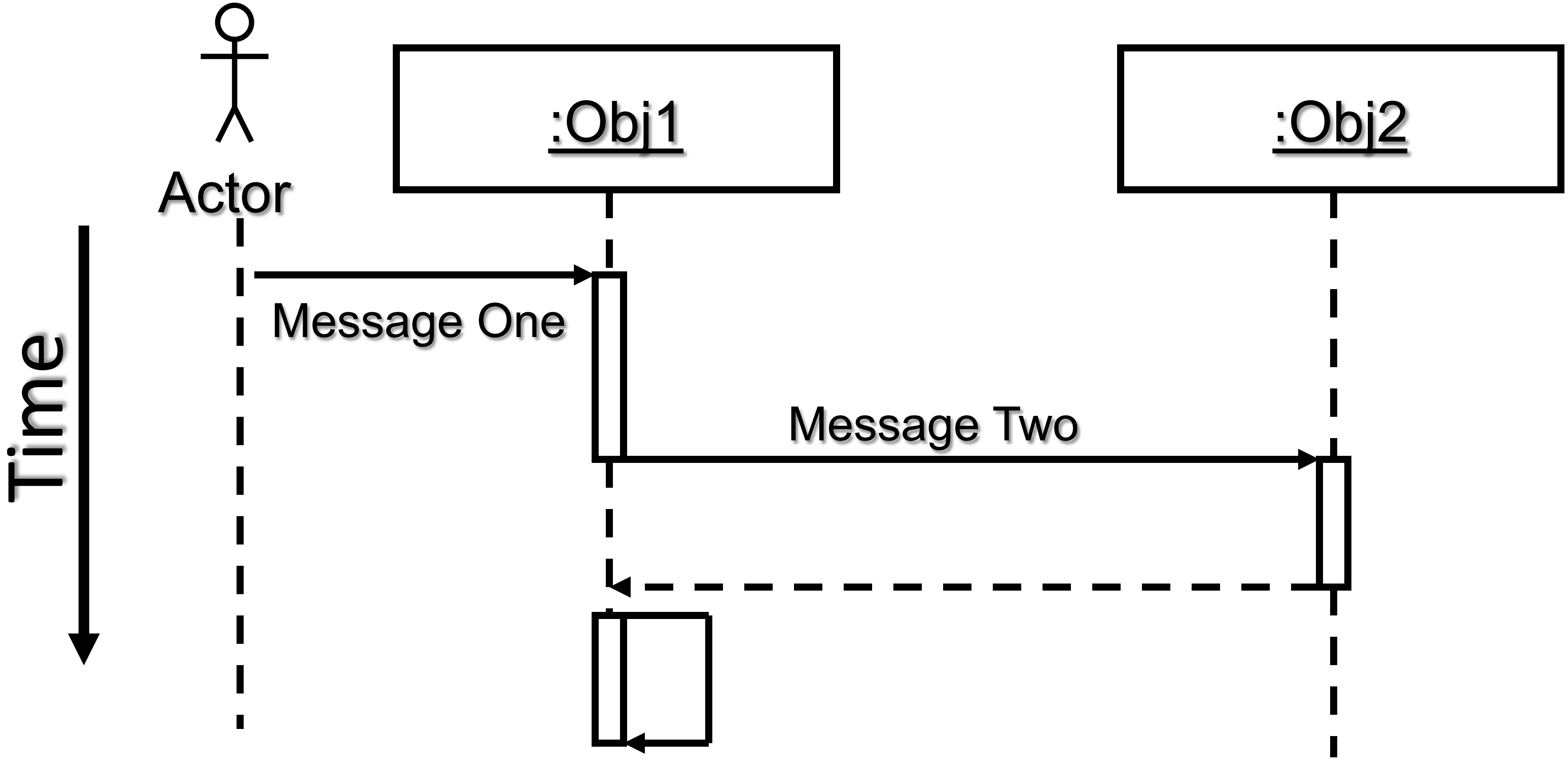
Sequence Diagram - Objects

- A life line illustrates what is happening to an object in a chronological fashion.



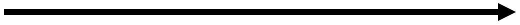
Sequence Diagram – Time & Messages

- Messages are used to illustrate communication between different active objects of a sequence diagram.

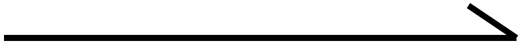


Types of Messages

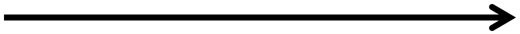
- Synchronous (flow interrupts until the message has completed.



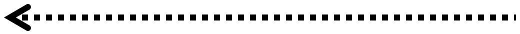
- Asynchronous (don’t wait for response)

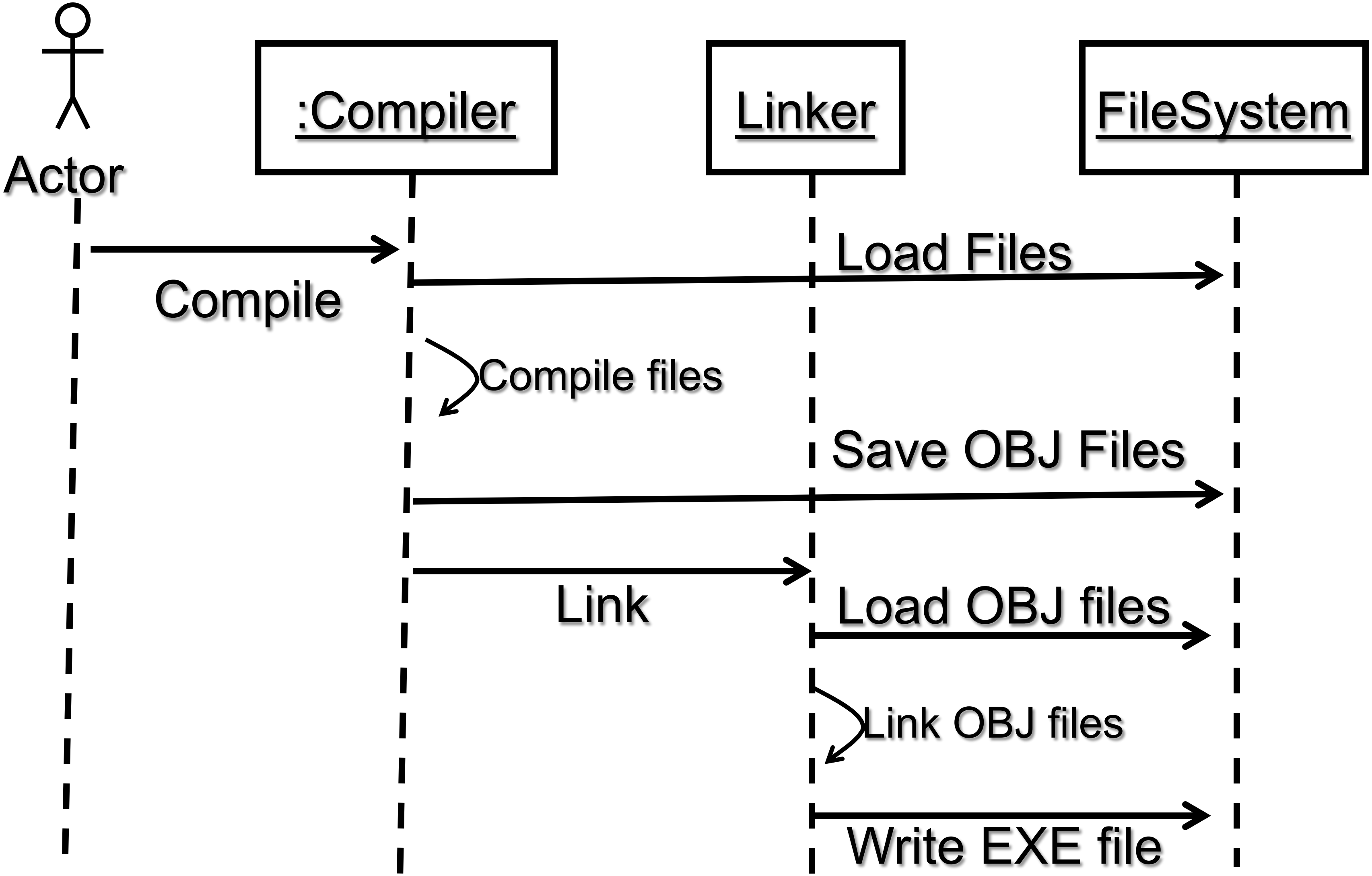


- Flat – no distinction between sysn/async



- Return – control flow has returned to the caller.





Actor

:Compiler

Linker

FileSystem

Compile

Load Files

Compile files

Save OBJ Files

Link

Load OBJ files

Link OBJ files

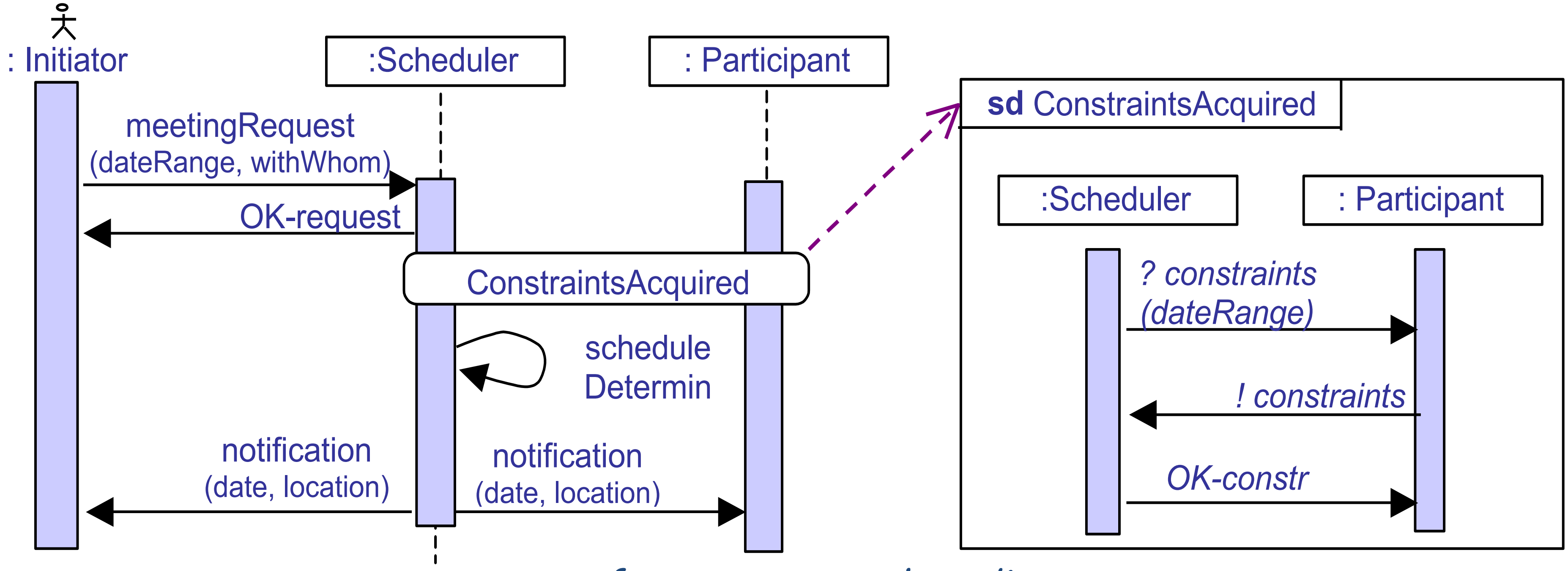
Write EXE file

How do you start? (Sequence Diagram)

1. Identify process/algorithm/activity you want to capture
   - A user story's scenario
2. Identify major objects involved
3. Map out flow of control/messages to achieve the result

Scenario refinement: episodes

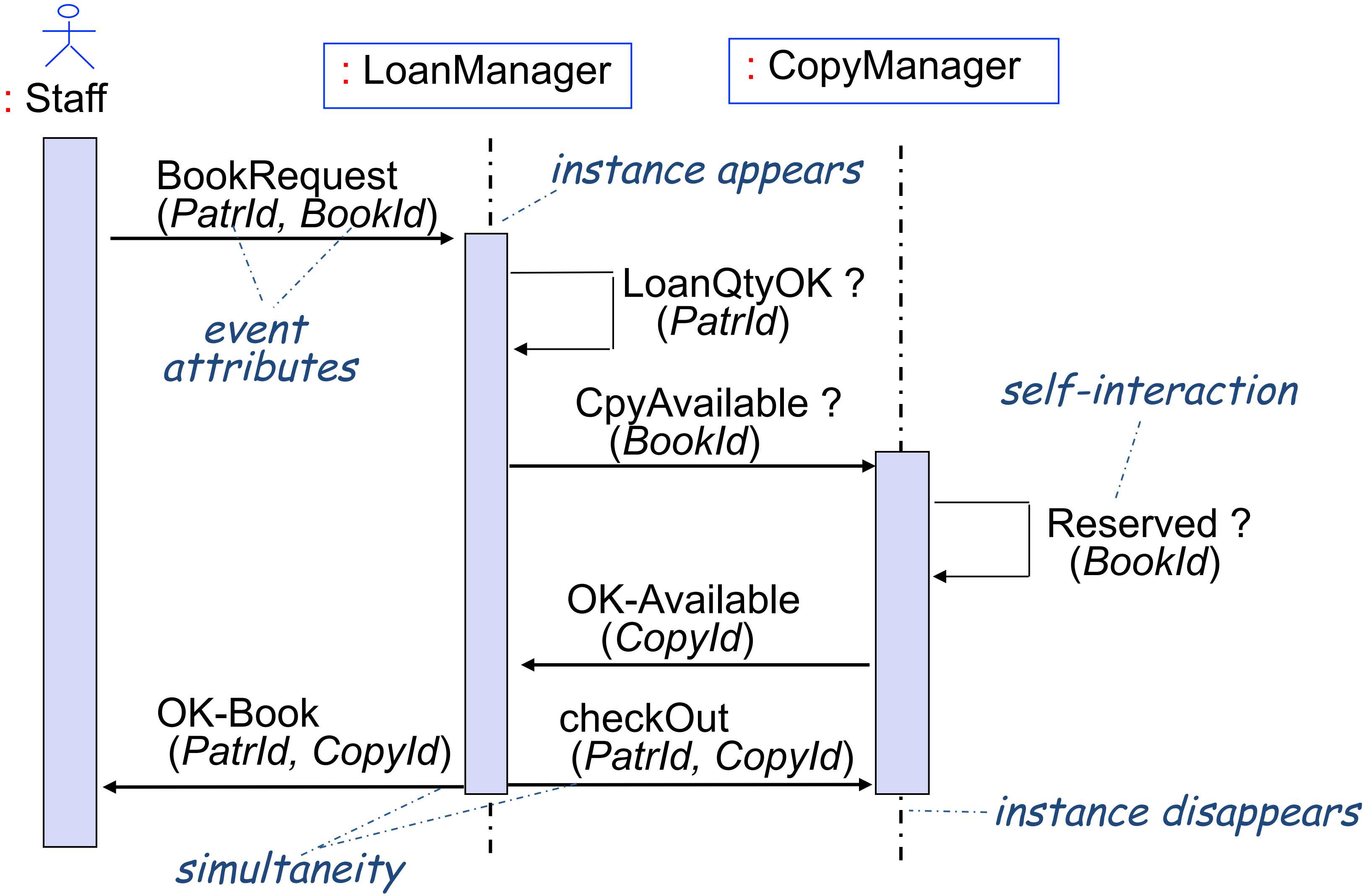
- Episode = subsequence of interactions for specific subgoal
- Appears as coarse-grained interaction
- To be detailed in another diagram with specific interactions
- Helpful for incremental elaboration of complex scenarios



reference to another diagram

Draw a **sequence diagram** for borrowing a book from the library

Scenarios as UML sequence diagrams


: Staff
: LoanManager
: CopyManager
BookRequest
(PatrId, BookId)
instance appears
event
attributes
LoanQtyOK ?
(PatrId)
CpyAvailable ?
(BookId)
self-interaction
Reserved ?
(BookId)
OK-Available
(CopyId)
OK-Book
(PatrId, CopyId)
checkOut
(PatrId, CopyId)
instance disappears
simultaneity