ECS505U
SOFTWARE ENGINEERING

MUSTAFA BOZKURT & LORENZO JAMONE

LECTURER IN SOFTWARE ENGINEERING

Week 4

Static and Dynamic Modelling with UML

SESSION OBJECTIVES

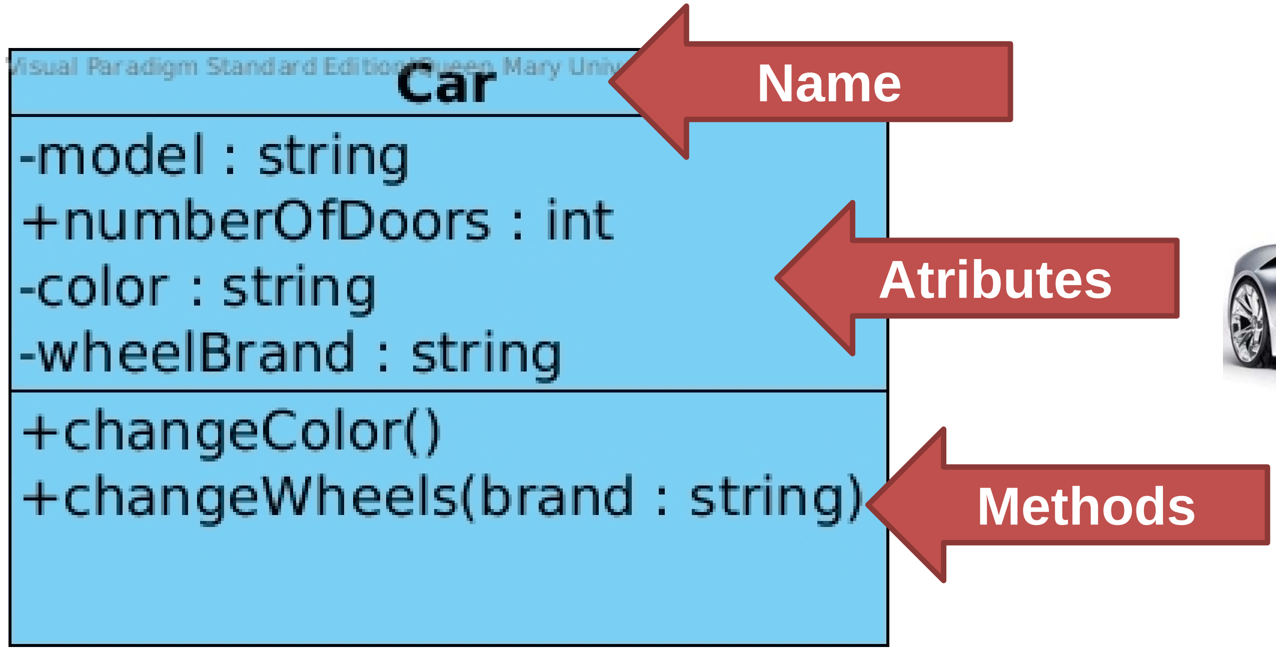
- Understand the key concepts of class diagrams
- Understand the key concepts of sequence diagrams
- Understand the key concepts of state diagrams

STRUCTURAL AND BEHAVIOURAL VIEWS

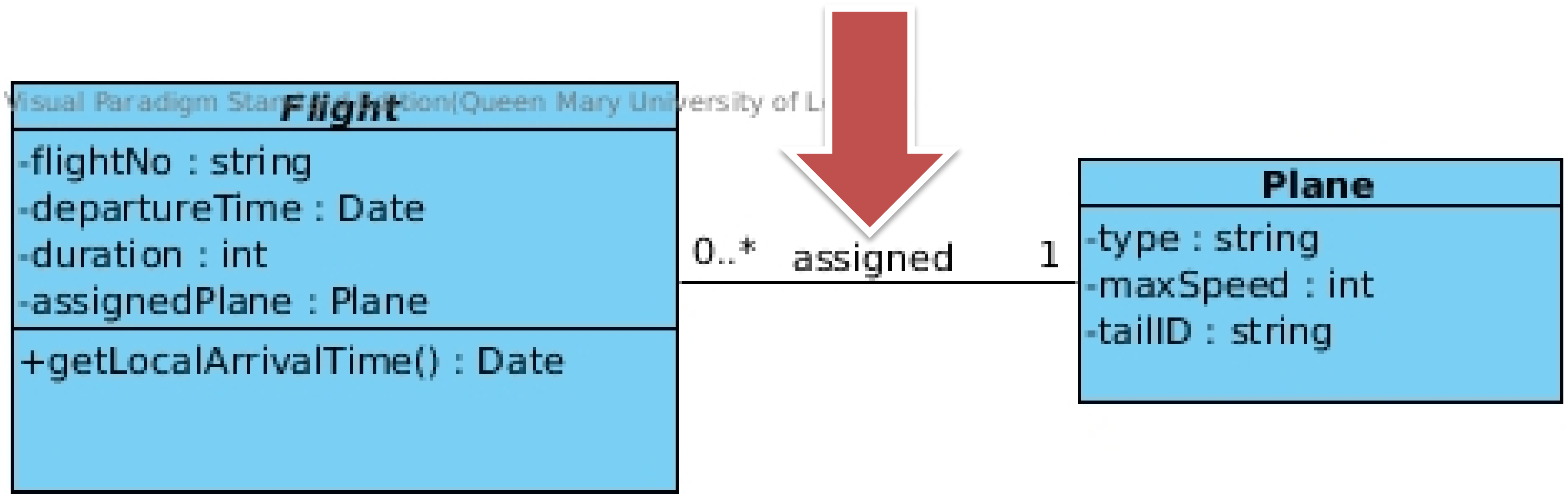
Static (or structural) view: emphasizes the static structure of the system using objects, attributes, operations and relationships.

Dynamic (or behavioral) view: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects.

UML CLASS DIAGRAM



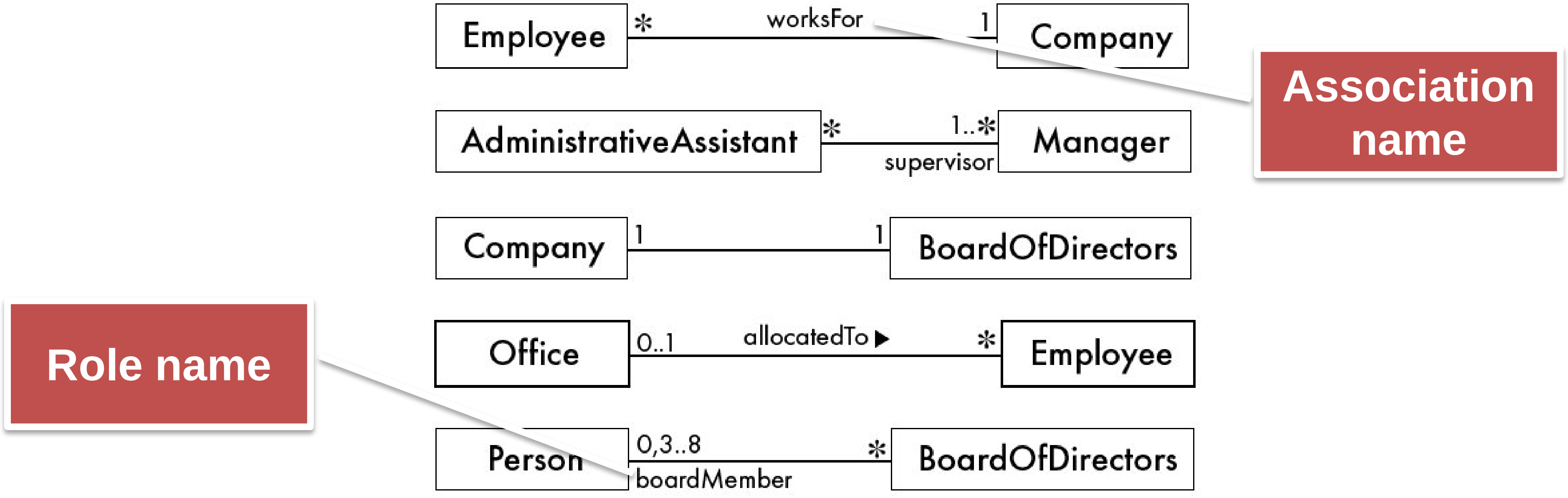
CLASS DIAGRAM: ASSOCIATION



An *association* is used to show how instances of two classes will reference each other.

The association is drawn as a line between the classes.

LABELLING ASSOCIATIONS

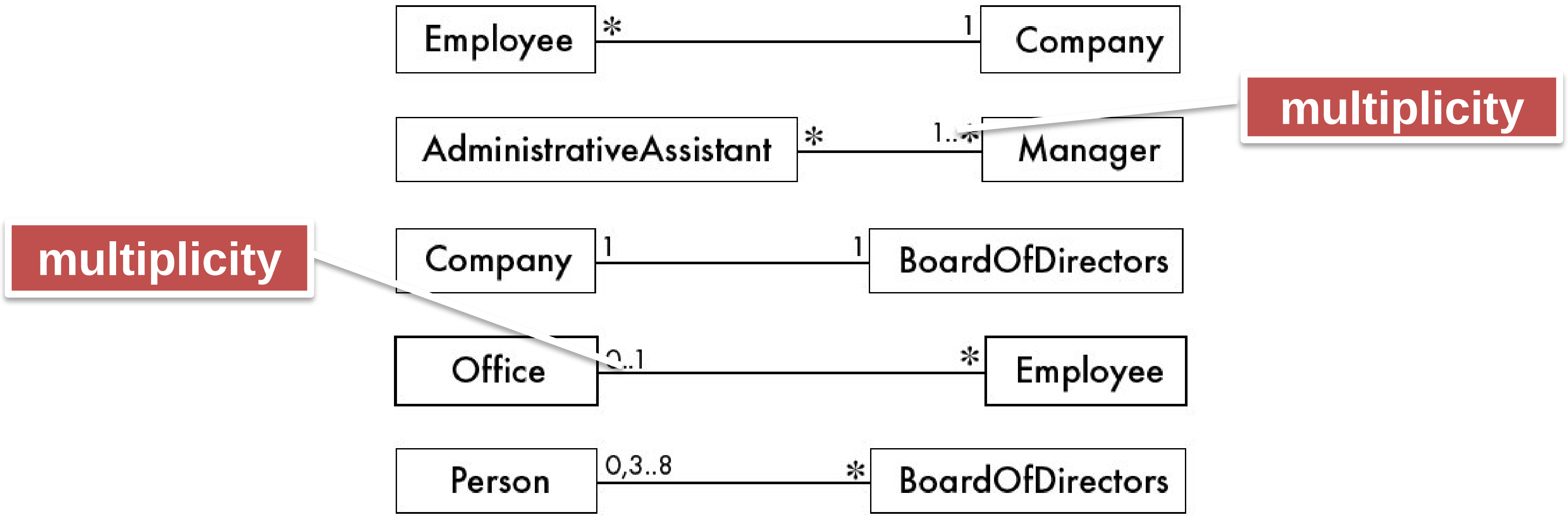


Two types of labels:

1. Association names
2. Role names

Add sufficient names to make the association clear and unambiguous

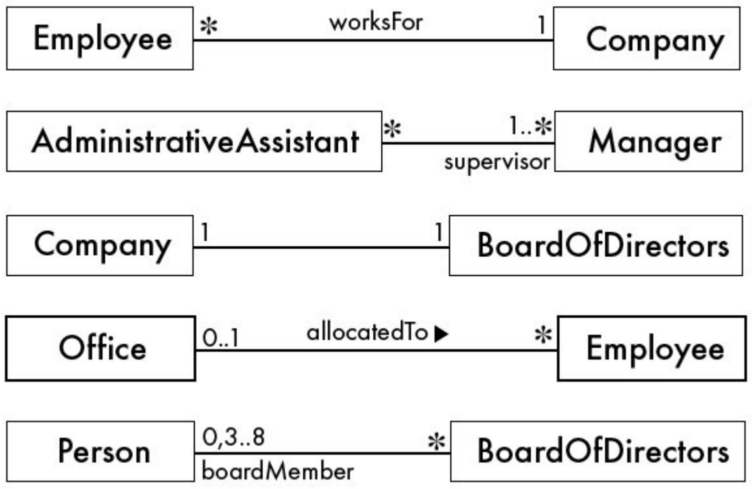
CLASS DIAGRAM: MULTIPLICITY



Multiplicity indicates how many instances of the class at this end of the association can be linked to an instance of the class at the other end of the association.

0: Zero, 1: One, *: Many and 0…* (interval or range)

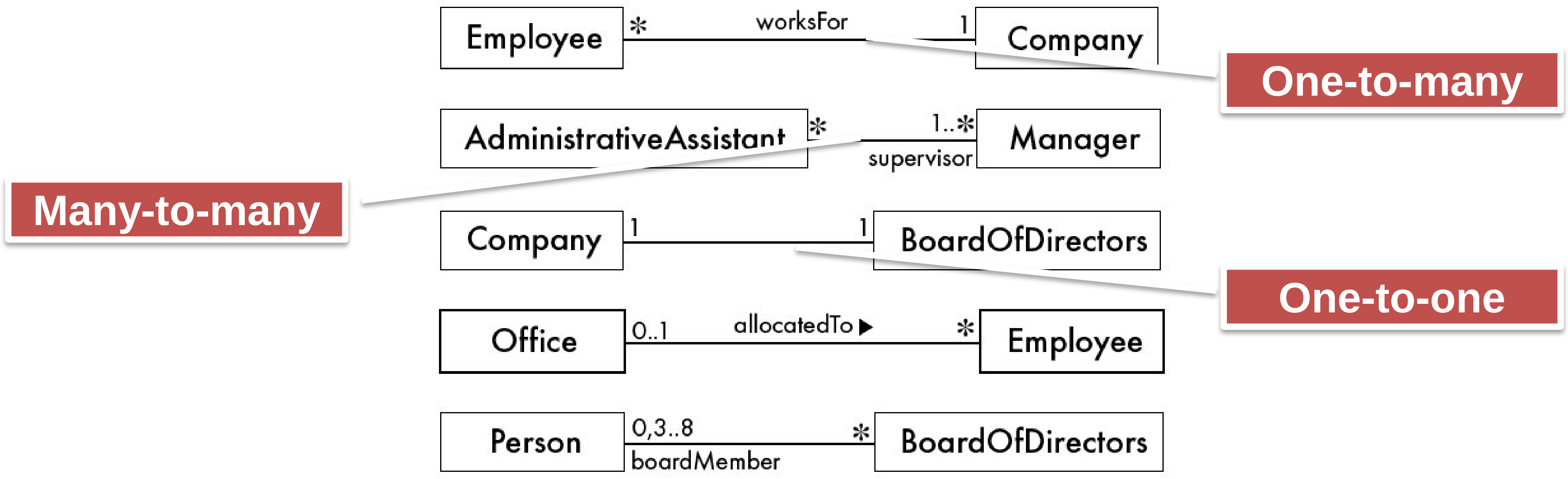
CLASS DIAGRAM: MULTIPLICITY



Indicator	Meaning
0..1	Zero or one
1	One only
0..*	Zero or more
*	Zero or more
1..*	One or more
3	Three only
0..5	Zero to Five
5..15	Five to Fifteen

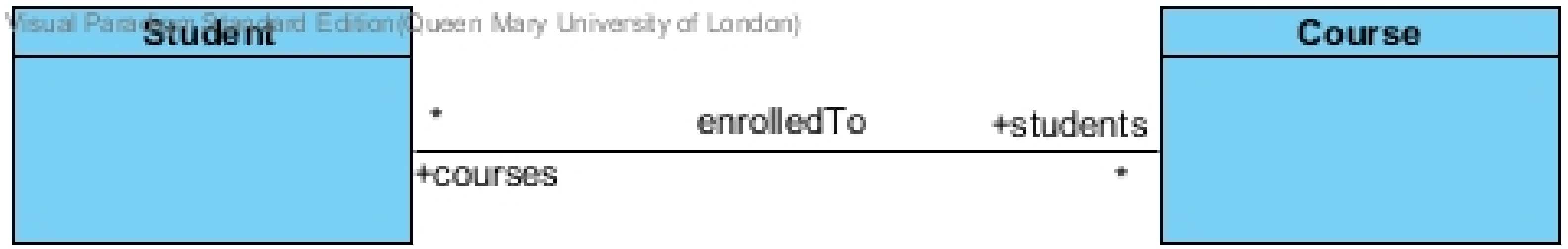
multiplicity-range ::= [lower-bound '..'] upper-bound

CLASS DIAGRAM: MULTIPLICITY



Pattern	Notes
One-to-many	Many side can be optional if not specified
Many-to-many	Both sides can be optional if not specified
One-to-one	Both class must exist at the same time

CLASS DIAGRAM: UML PROPERTY

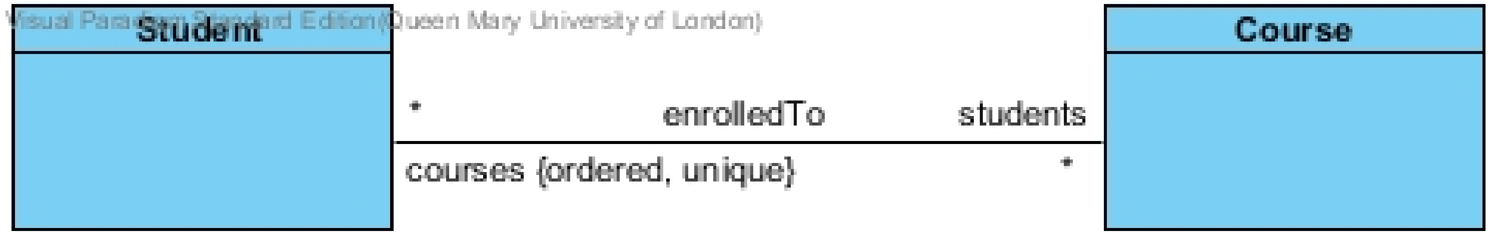

Visual Paradigm Standard Edition(Queen Mary University of London)
Student
Course
*
enrolledTo
+students
+courses
*

Can students be enrol to the same course more than once?

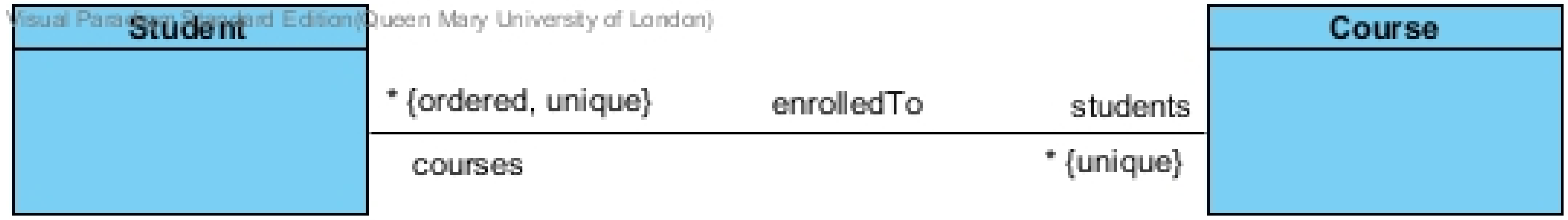
CLASS DIAGRAM: UML PROPERTY

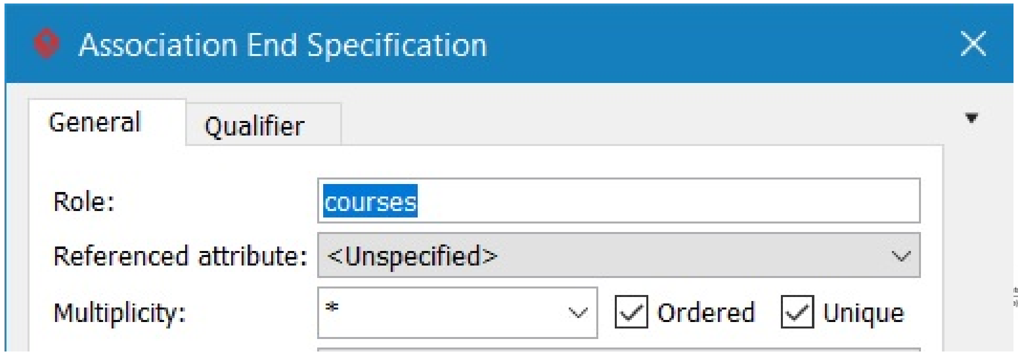
UML property allow us to set order and uniqueness of the association

- <order-designator> ::= 'ordered' | 'unordered'
- <uniqueness-designator> ::= 'unique' | 'nonunique'



CLASS DIAGRAM: UML PROPERTY

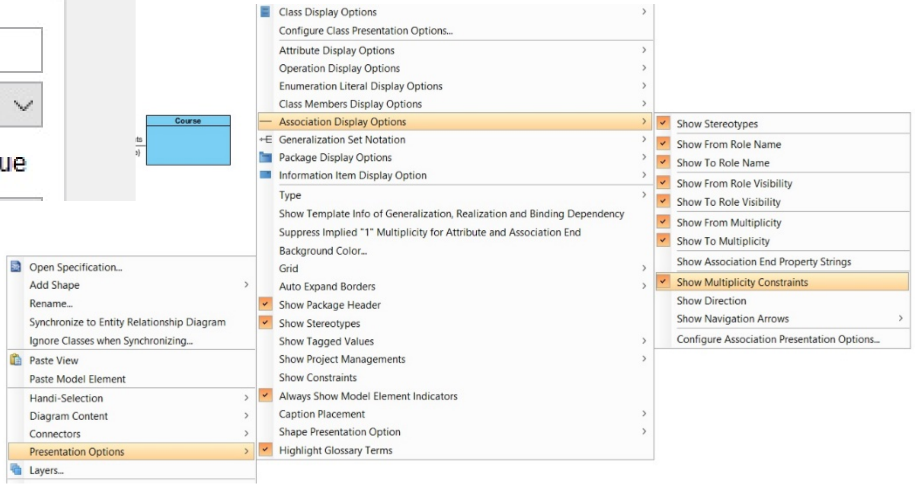




Role: courses

Referenced attribute: <Unspecified>

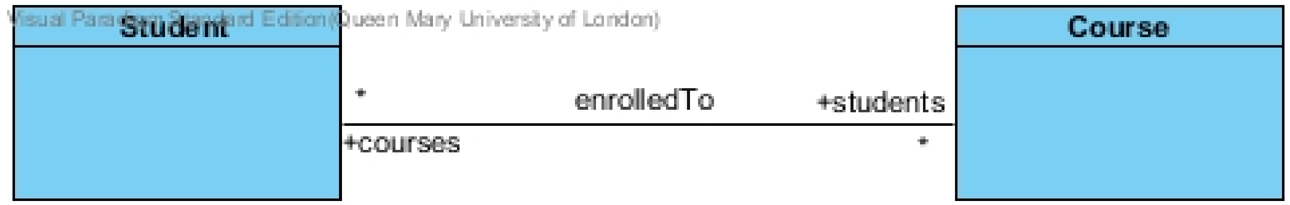
Multiplicity: * [✓ Ordered] [✓ Unique]


Course
Open Specification...
Add Shape
Rename...
Synchronize to Entity Relationship Diagram
Ignore Classes when Synchronizing...
Paste View
Paste Model Element
Handi-Selection
Diagram Content
Connectors
Presentation Options
Layers...
Class Display Options
Configure Class Presentation Options...
Attribute Display Options
Operation Display Options
Enumeration Literal Display Options
Class Members Display Options
Association Display Options
Generalization Set Notation
Package Display Options
Information Item Display Option
Type
Show Template Info of Generalization, Realization and Binding Dependency
Suppress Implied "1" Multiplicity for Attribute and Association End
Background Color...
Grid
Auto Expand Borders
Show Package Header
Show Stereotypes
Show Tagged Values
Show Project Managements
Show Constraints
Always Show Model Element Indicators
Caption Placement
Shape Presentation Option
Highlight Glossary Terms
Show Stereotypes
Show From Role Name
Show To Role Name
Show From Role Visibility
Show To Role Visibility
Show From Multiplicity
Show To Multiplicity
Show Association End Property Strings
Show Multiplicity Constraints
Show Direction
Show Navigation Arrows
Configure Association Presentation Options...

CLASS DIAGRAM: UML PROPERTY

If multiplicity element is multivalued and specified as ordered, then the collection of values in an instantiation of this element is sequentially ordered. By default, collections are not ordered.

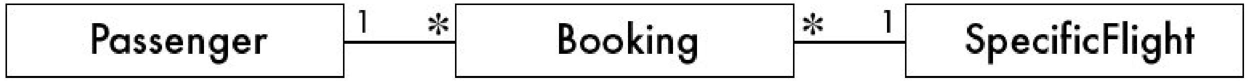
If multiplicity element is multivalued and specified as unique, then each value in the collection of values in an instantiation of this element must be unique. By default, each value in collection is unique.


Visual Paradigm Standard Edition(Queen Mary University of London)
Student
Course
*
enrolledTo
+students
+courses
*

CLASS DIAGRAM: MULTIPLICITY

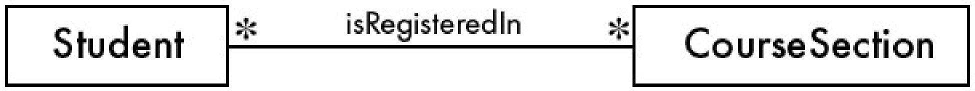
Discuss the implied associations in this diagram:

1. A Booking is always for exactly one Passenger
2. A Passenger can have any number of Bookings
3. A Booking is always for one SpecificFlight
4. A SpecificFlight can have any number of Bookings



No Booking without a Passenger
Passenger without booking!
Flight without booking
No booking without flight

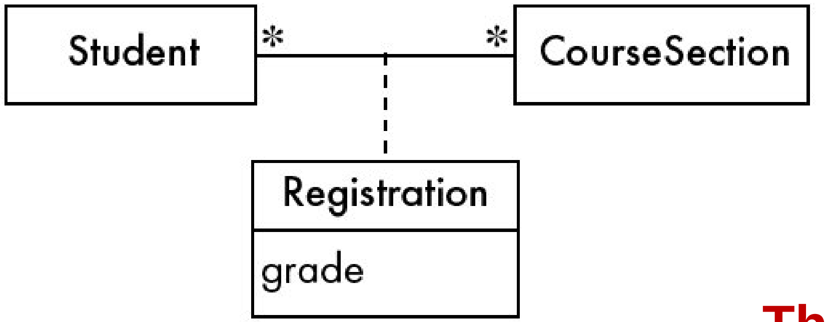
CLASS DIAGRAM: ASSOCIATION CLASSES

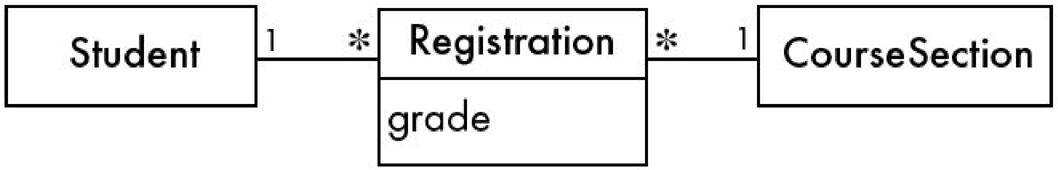


In which class should the student's grade be put?

Student class: a student could have only one grade

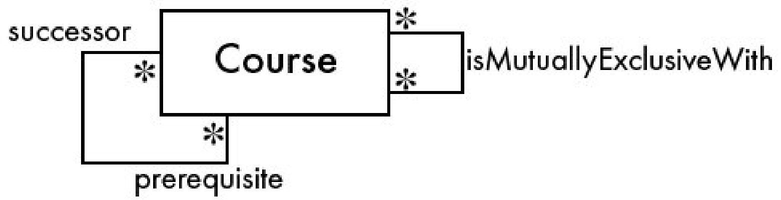
CourseSection class: a course section could have only one grade





These are exactly the same

CLASS DIAGRAM: REFLEXIVE ASSOCIATION



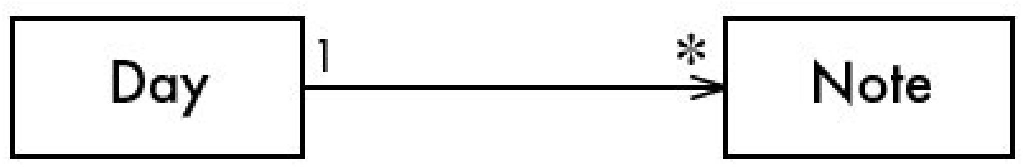
Reflexive association: Association to connect a class to itself

Asymmetric: Roles of each end is different

Symmetric: Same roles

It’s good practice to label an *asymmetric reflexive association using role names instead of an association name*.

CLASS DIAGRAM: DIRECTIONAL ASSOCIATIONS



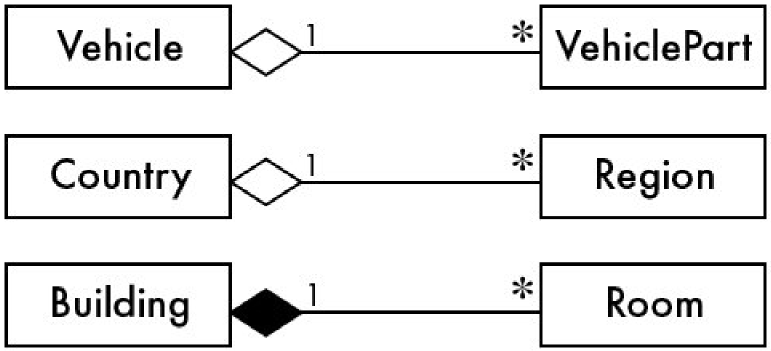
Day instances knows (expected) which *note instances they are linked to*.

Two types: bi-directional and uni-directional

Link is an instance of an association.

Associations and links are *bi-directional by default*.

CLASS DIAGRAM: AGGREGATION

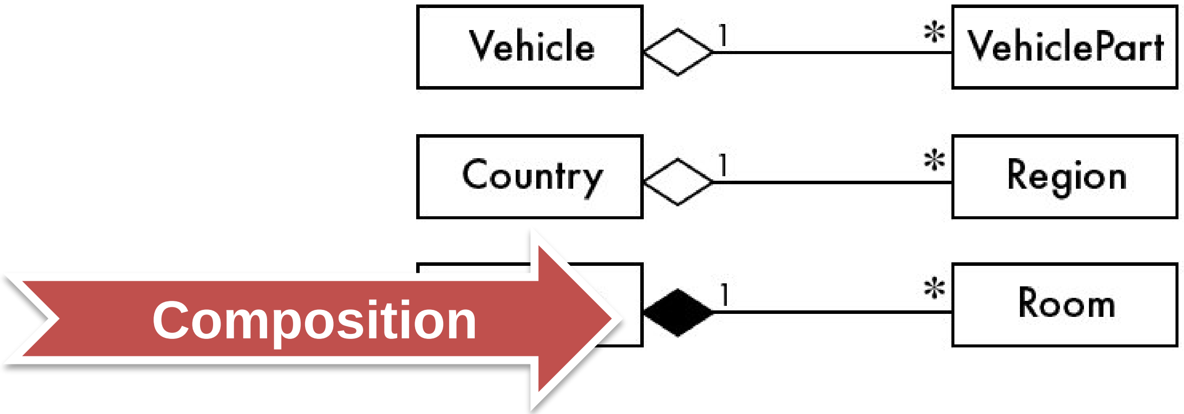


Mark an association as an aggregation if

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.

CLASS DIAGRAM: COMPOSITION



A composition is a strong kind of aggregation in which if the aggregate is destroyed, then the parts are destroyed as well.

1. In an aggregation, the child class instance can outlive its parent class.
2. In an composition, the child class's instance lifecycle is dependent on the parent class's instance lifecycle.

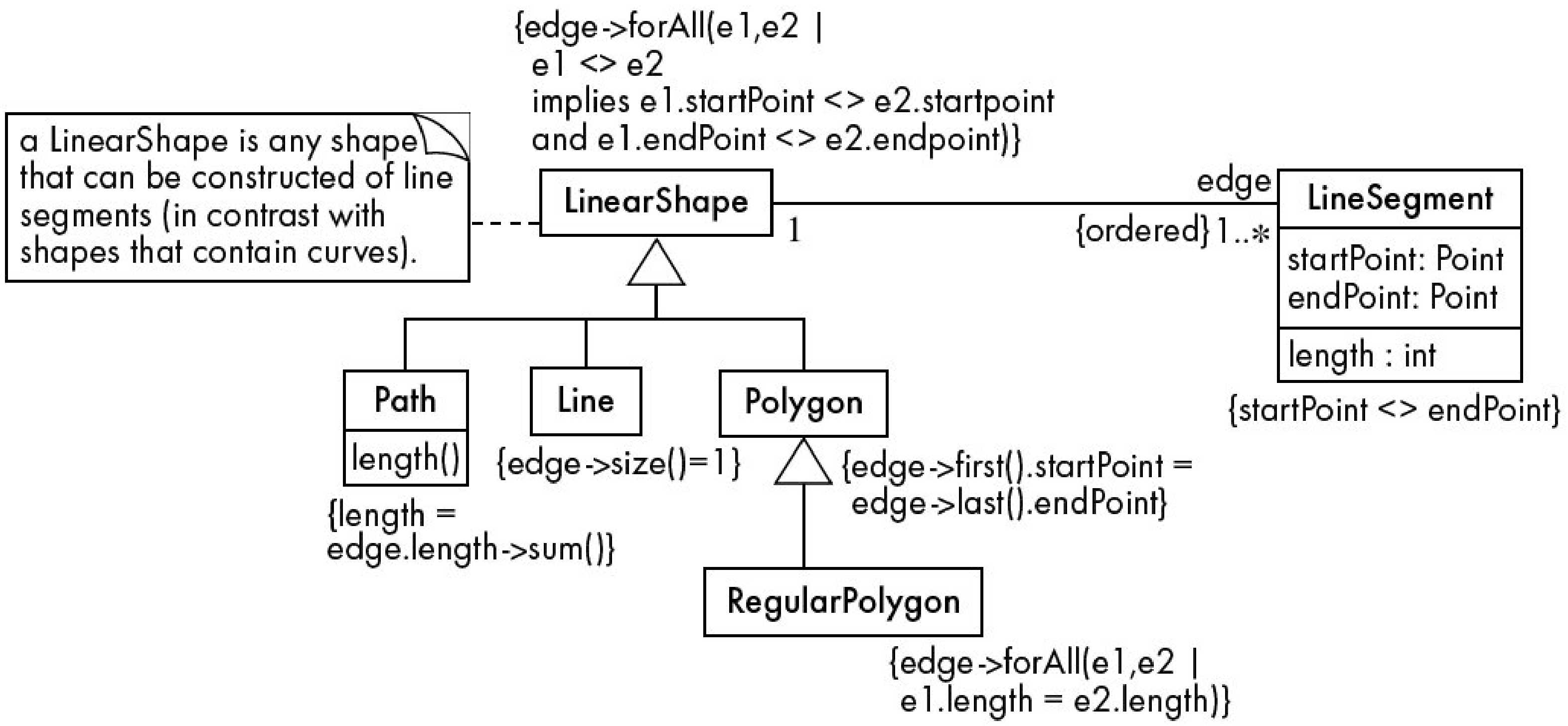
Parts of a composition *can never have a life of their own!*

CLASS DIAGRAM: COMPOSITION

Benefit of aggregation: *improved encapsulation!*

A system considers computers as an aggregation of parts allow performing operations such as sell, send, delete on computer rather than individual parts.

CLASS DIAGRAM: CONSTRAINTS AND NOTES


{edge->forAll(e1,e2 |
e1 <> e2
implies e1.startPoint <> e2.startpoint
and e1.endPoint <> e2.endpoint)}
a LinearShape is any shape that can be constructed of line segments (in contrast with shapes that contain curves).
LinearShape
1
edge
{ordered} 1..*
LineSegment
startPoint: Point
endPoint: Point
length : int
{startPoint <> endPoint}
Path
length()
{length = edge.length->sum()}
Line
{edge->size()=1}
Polygon
{edge->first().startPoint = edge->last().endPoint}
RegularPolygon
{edge->forAll(e1,e2 | e1.length = e2.length)}

Read the rest of Chapter 5!

THE DYNAMIC MODEL

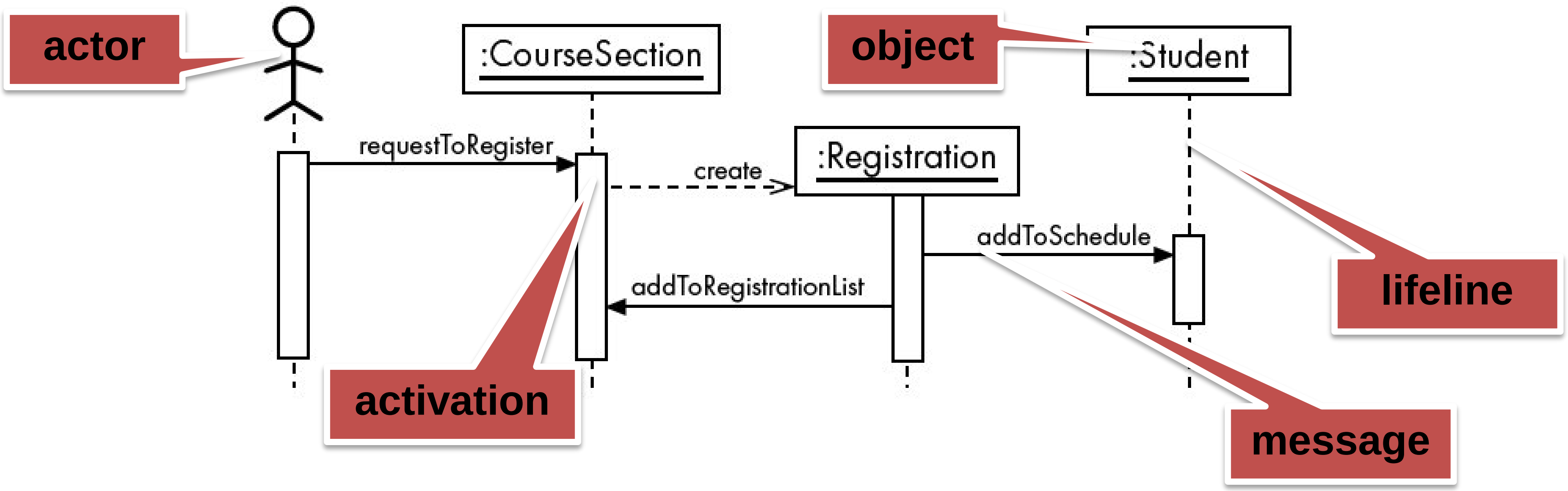
- What an object does
- With whom does the object collaborate
- What happens to an object

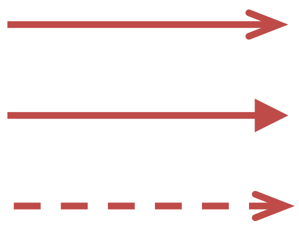
THE DYNAMIC MODEL

Two types:

- Interaction diagram models system execution
  - Sequence diagram
  - Communication diagram
- State and activity diagrams model system behaviour

SEQUENCE DIAGRAM



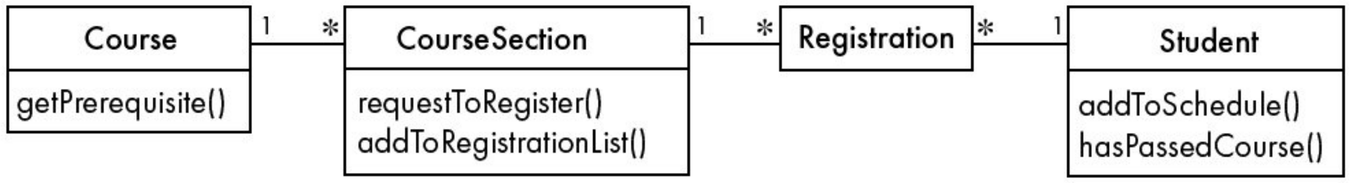


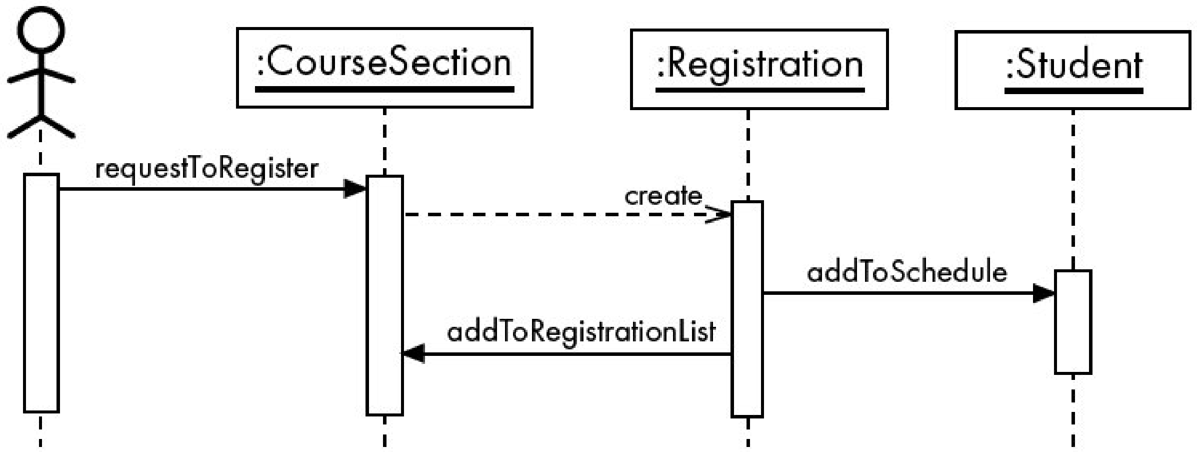
Asynchronous message

Synchronous message

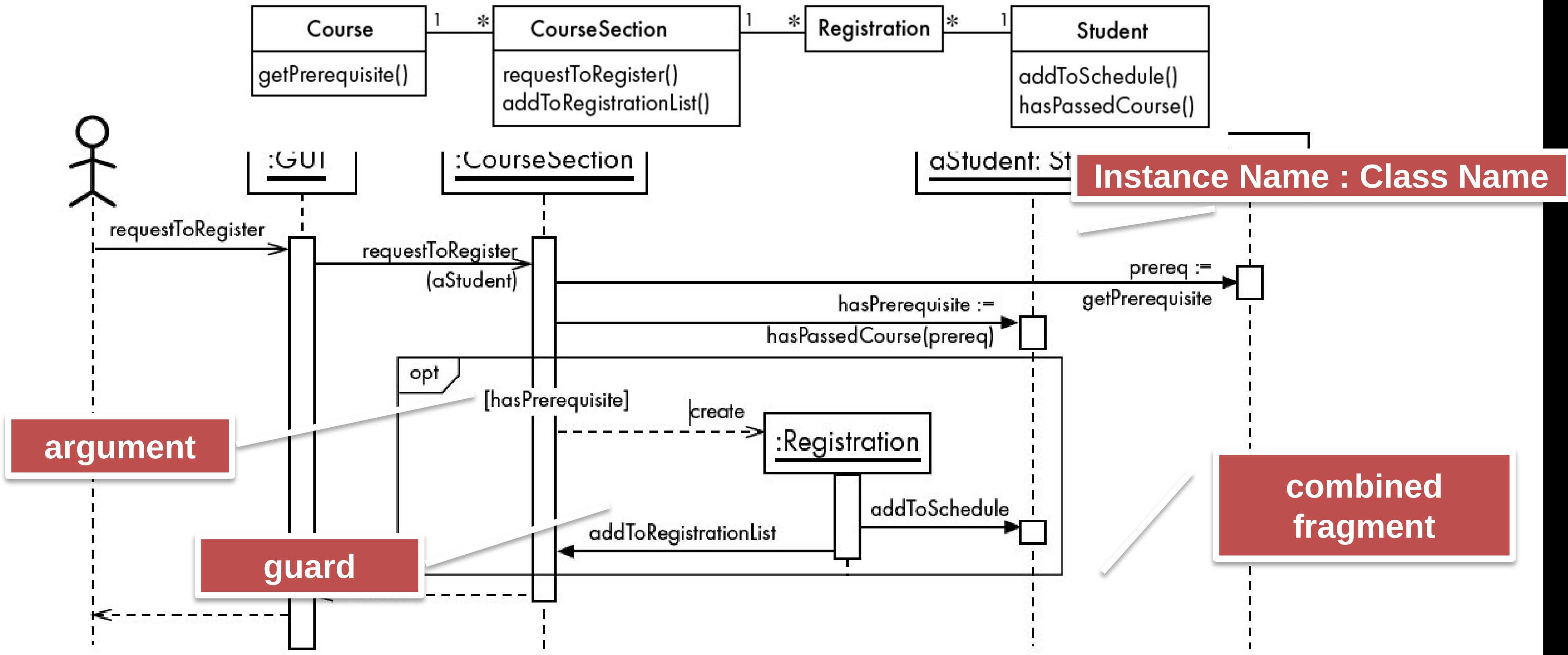
Response message

SEQUENCE DIAGRAM

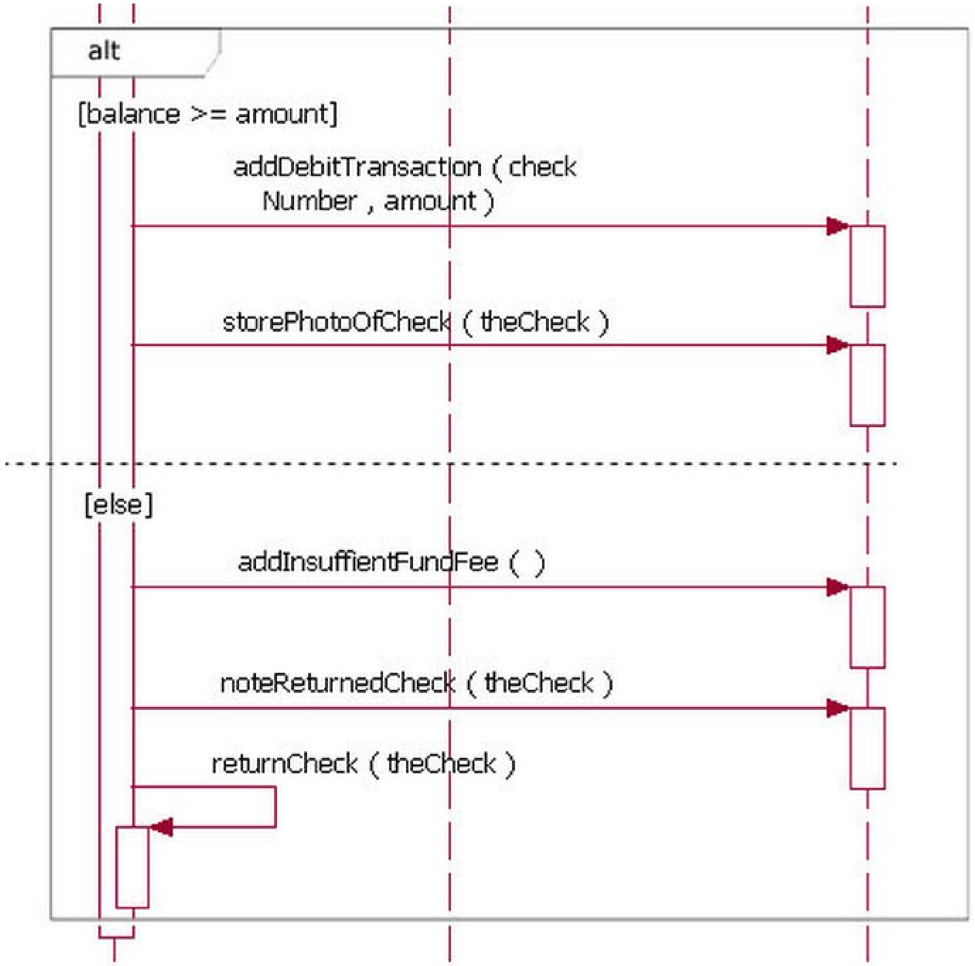




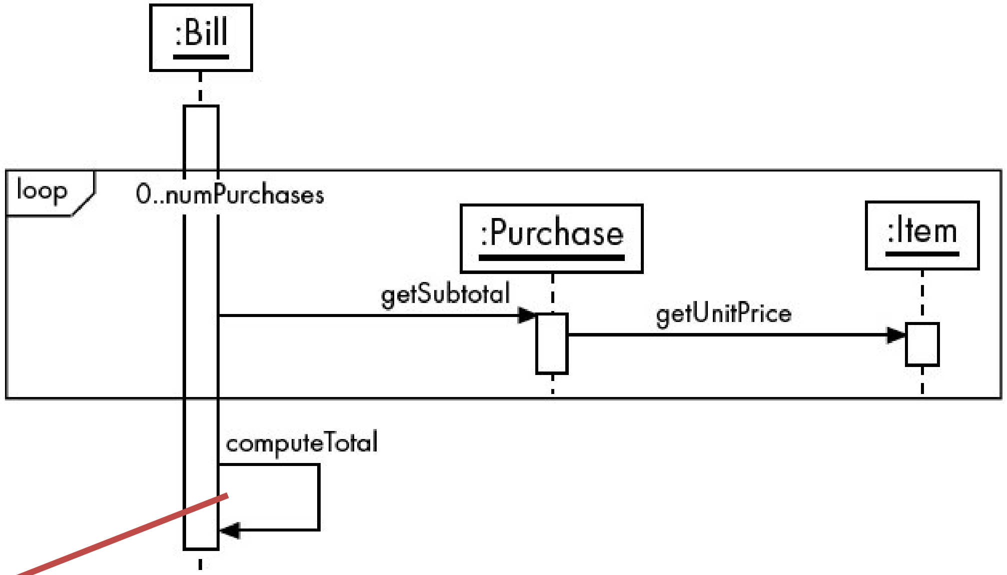
SEQUENCE DIAGRAM

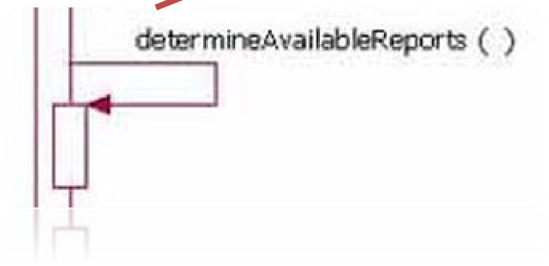


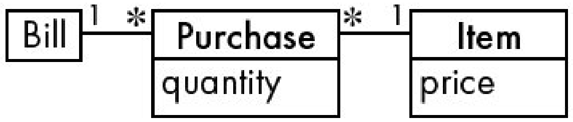
SEQUENCE DIAGRAM


alt
[balance >= amount]
addDebitTransaction (check Number , amount)
storePhotoOfCheck (theCheck)
[else]
addInsuffientFundFee ( )
noteReturnedCheck (theCheck)
returnCheck (theCheck)

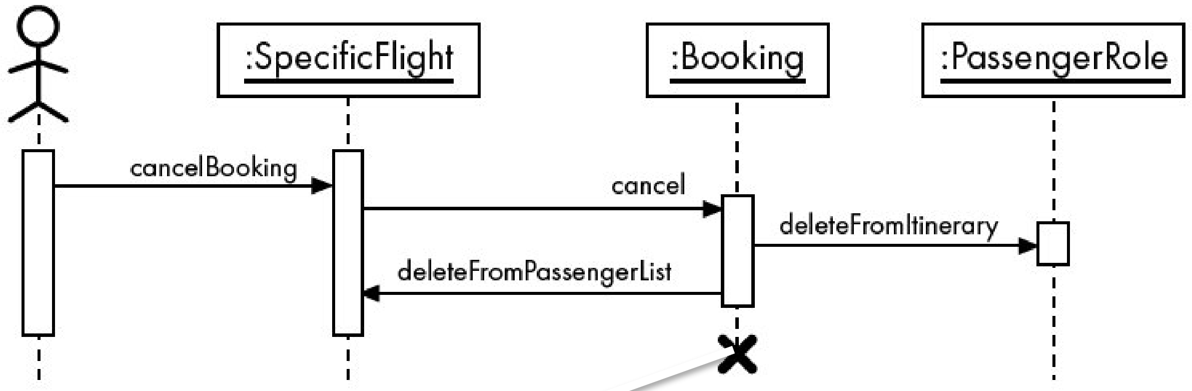
SEQUENCE DIAGRAM





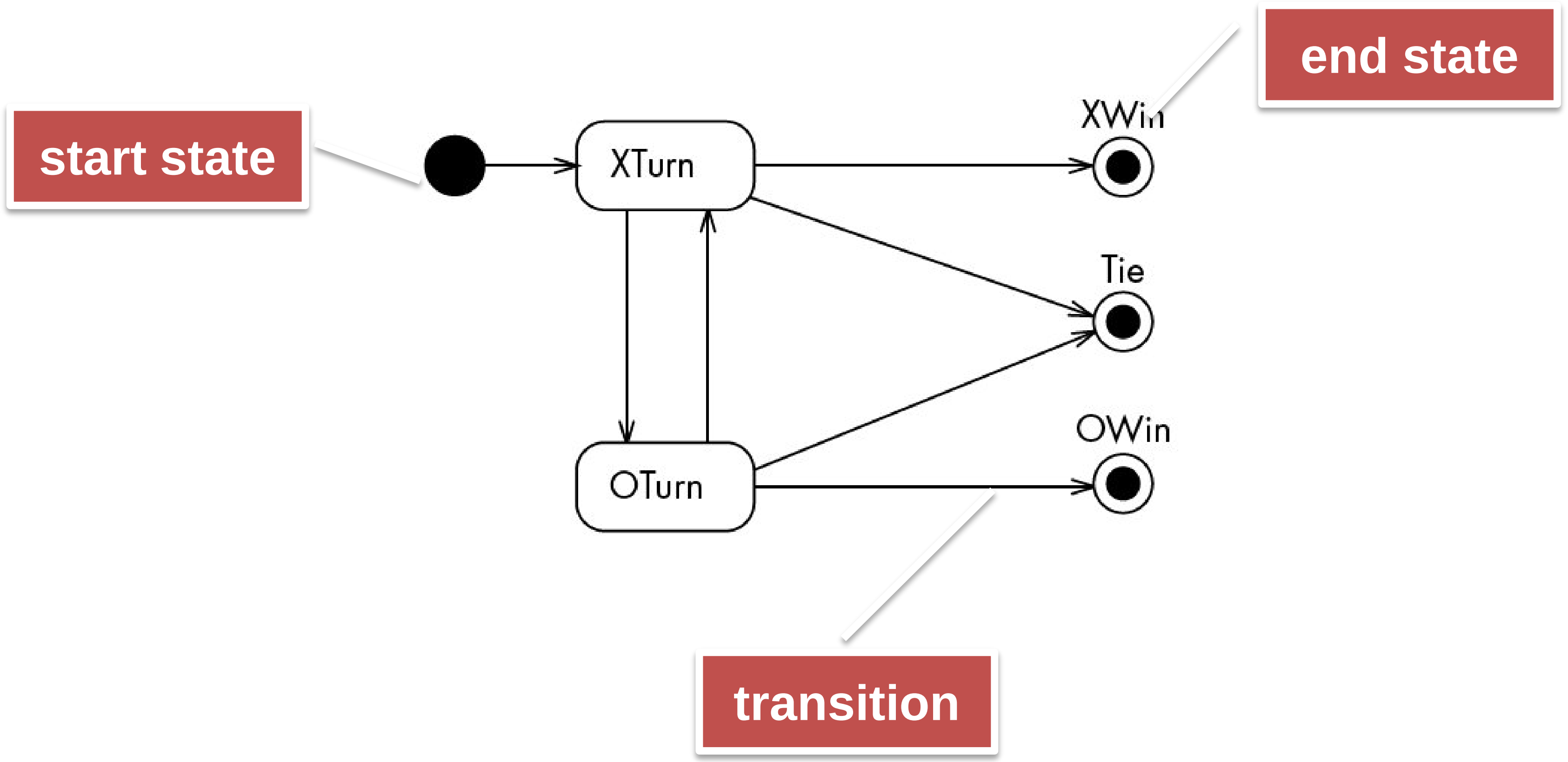


SEQUENCE DIAGRAM



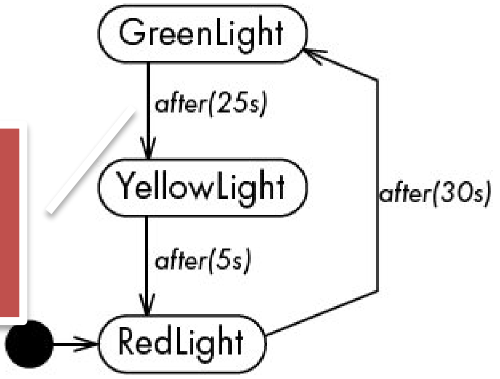
delete

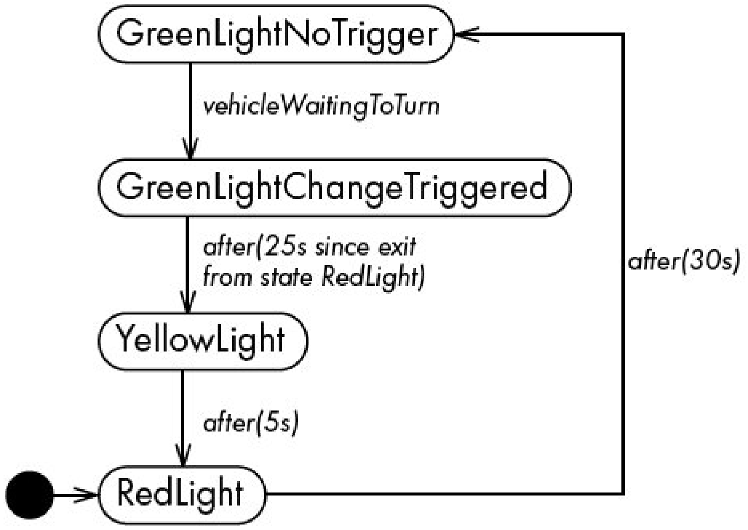
STATE (MACHINE) DIAGRAM



Queen Mary University of London

STATE DIAGRAM



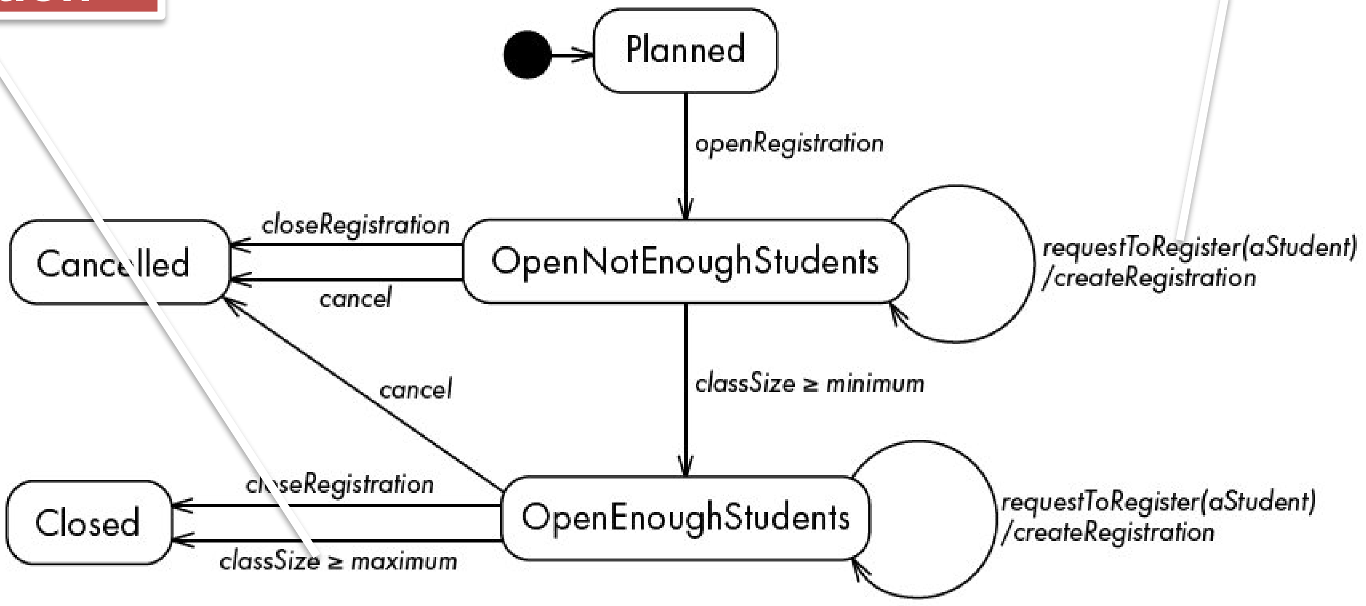


Elapsed-time transition

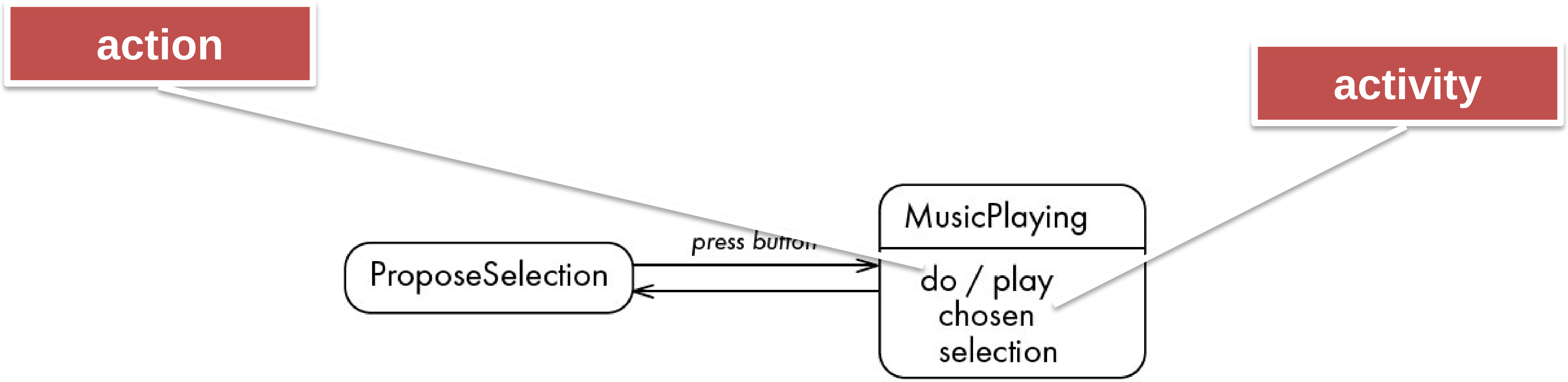
STATE DIAGRAM

Transition with a condition

Transition with a condition



STATE DIAGRAM



Two types of computations:

- **Actions**
- **Activities**

STATE DIAGRAM

Action takes place effectively instantaneously

When the system takes a particular transition.

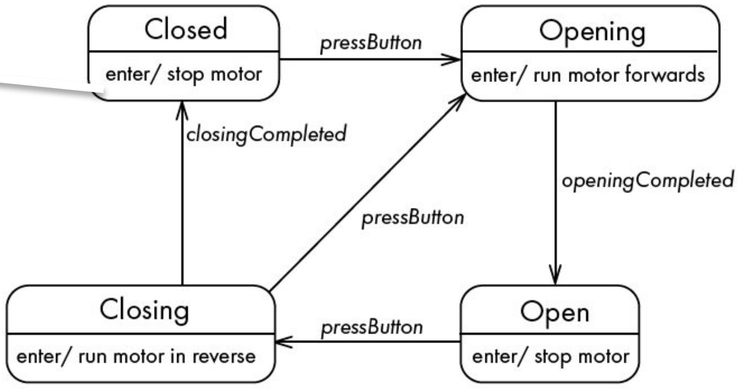
Upon entry into a particular state, no matter which transition causes entry into that state.

Upon exit from a particular state, no matter which transition is being taken.

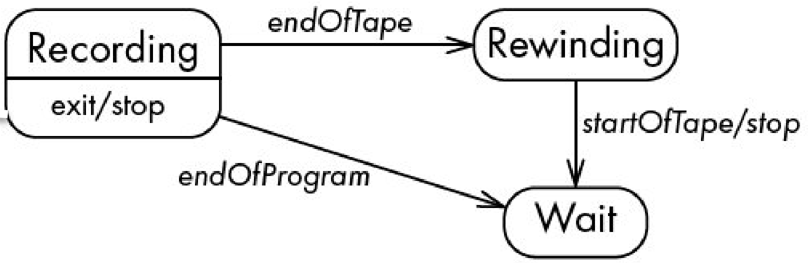
UML have a number of built-in actions such as *send a message*, *create* and *destroy*.

STATE DIAGRAM

enter action



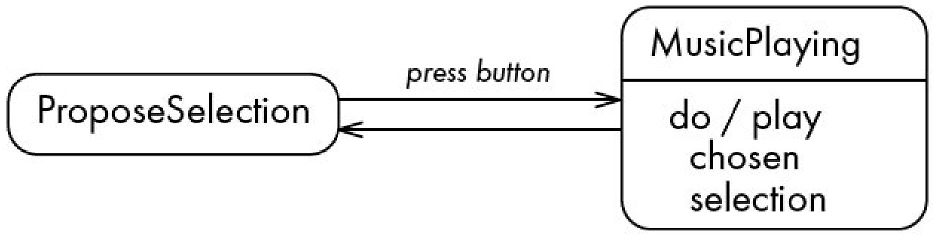
exit action



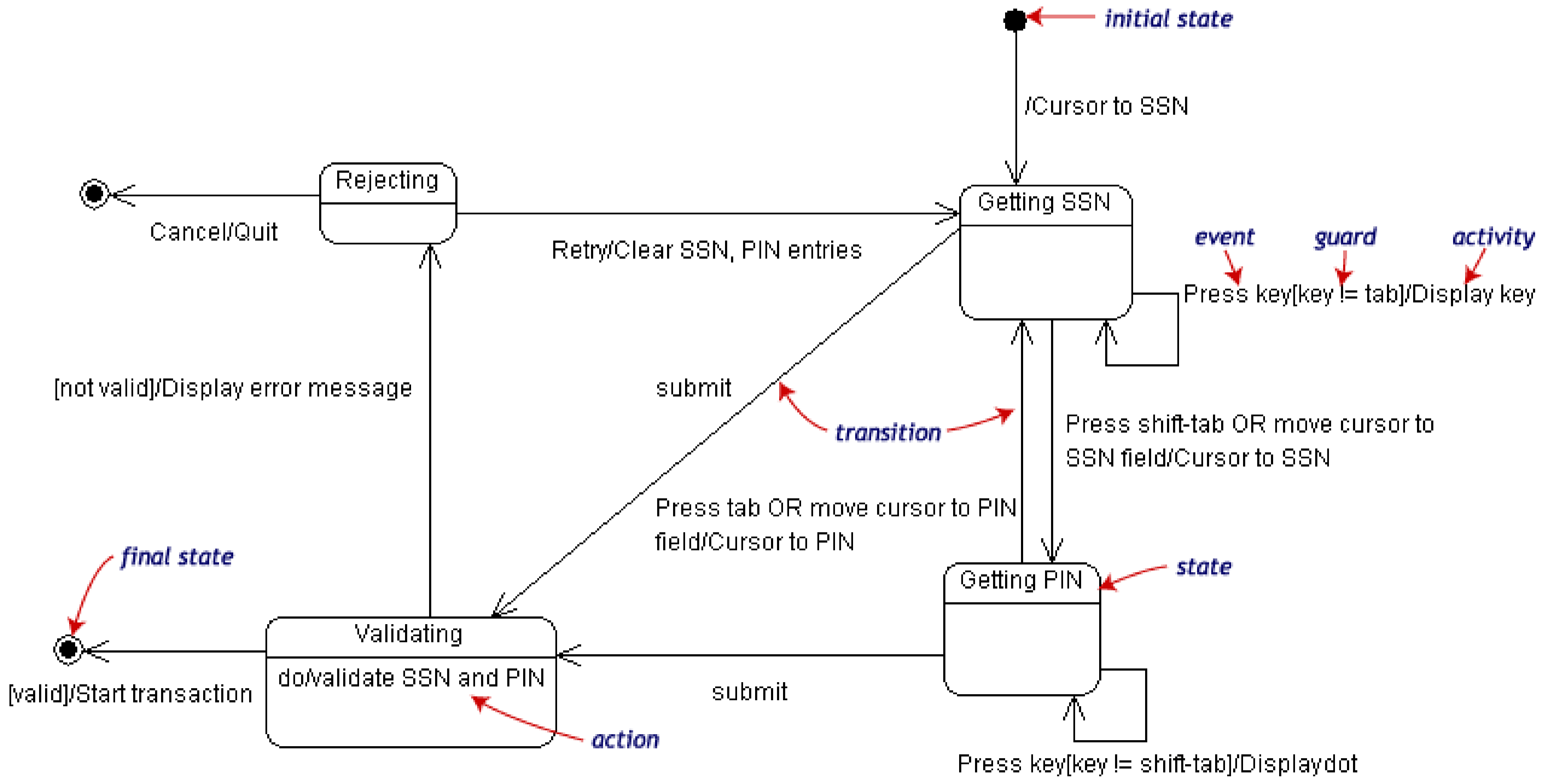
STATE DIAGRAM

An Activity:

- Often occurs over a period of time and takes place while the system is in a state.
- Out of the state in response to completion of the activity.
- If some other transition is triggered first, then the system has to terminate the activity as it leaves the state.

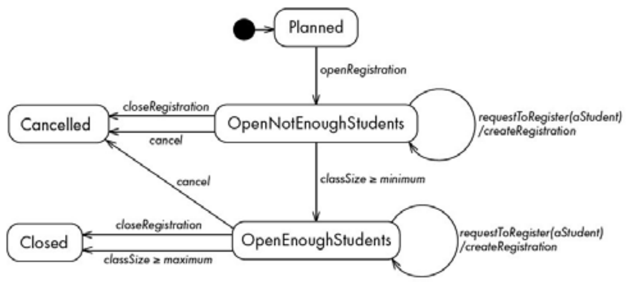


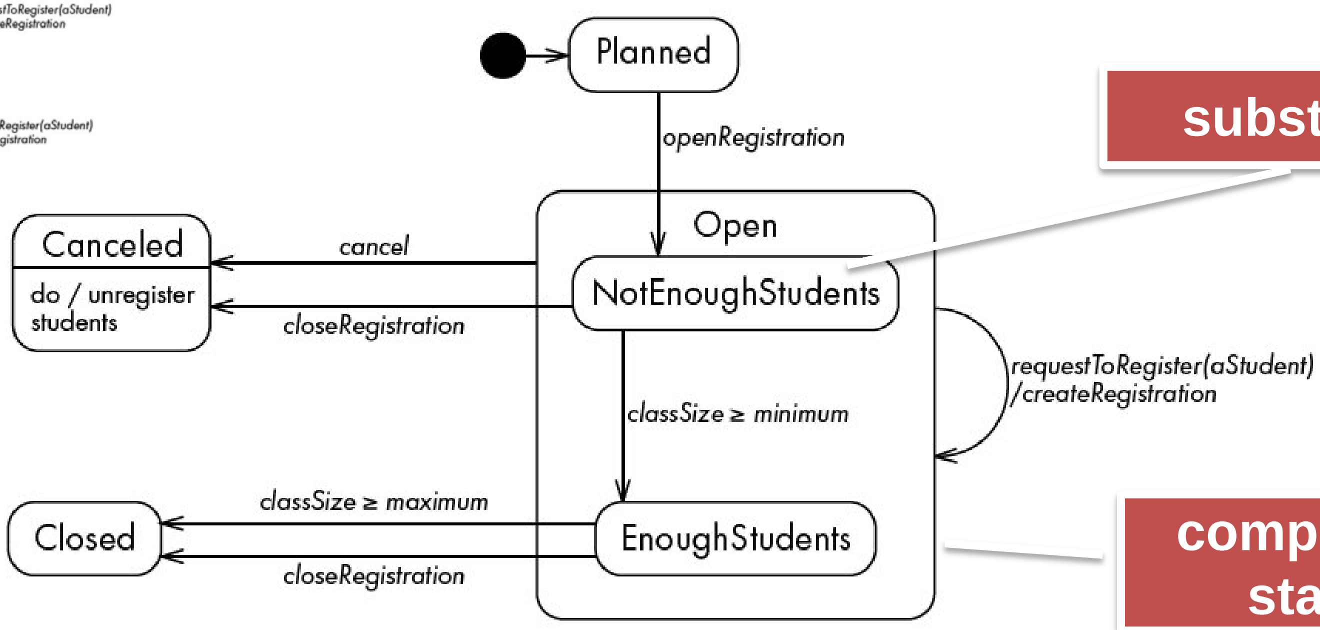
STATECHART IN TOGETHER


initial state
/Cursor to SSN
Rejecting
Getting SSN
Cancel/Quit
Retry/Clear SSN, PIN entries
event
guard
activity
Press key[key != tab]/Display key
[not valid]/Display error message
submit
transition
Press shift-tab OR move cursor to SSN field/Cursor to SSN
Press tab OR move cursor to PIN field/Cursor to PIN
final state
Getting PIN
state
Validating
do/validate SSN and PIN
[valid]/Start transaction
submit
action
Press key[key != shift-tab]/Displaydot

STATE DIAGRAM: NESTED STATES

Queen Mary
University of London





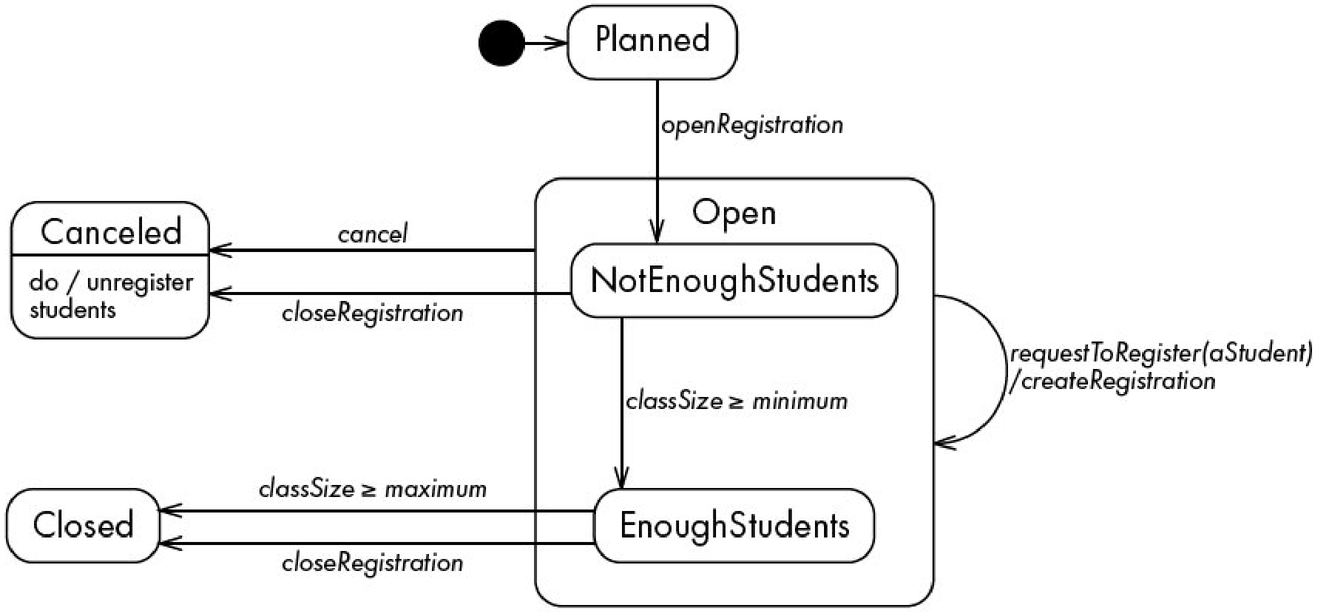
substate

composite state

Substate/nested state: States inside another state (inner diagram).

Composite state: States that contain multiple states

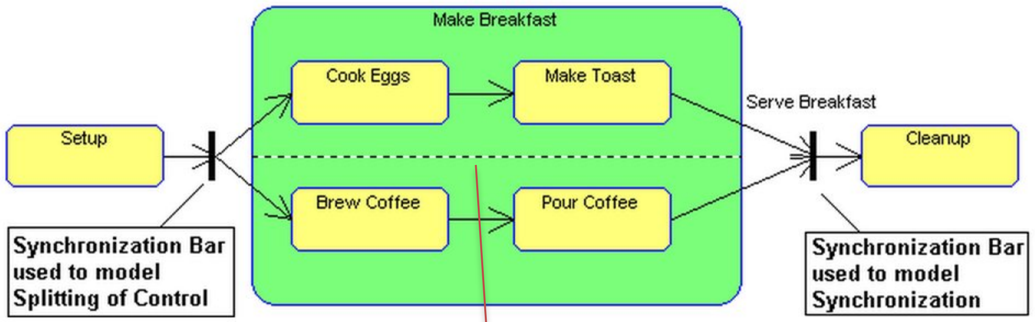
STATE DIAGRAM: NESTED STATES



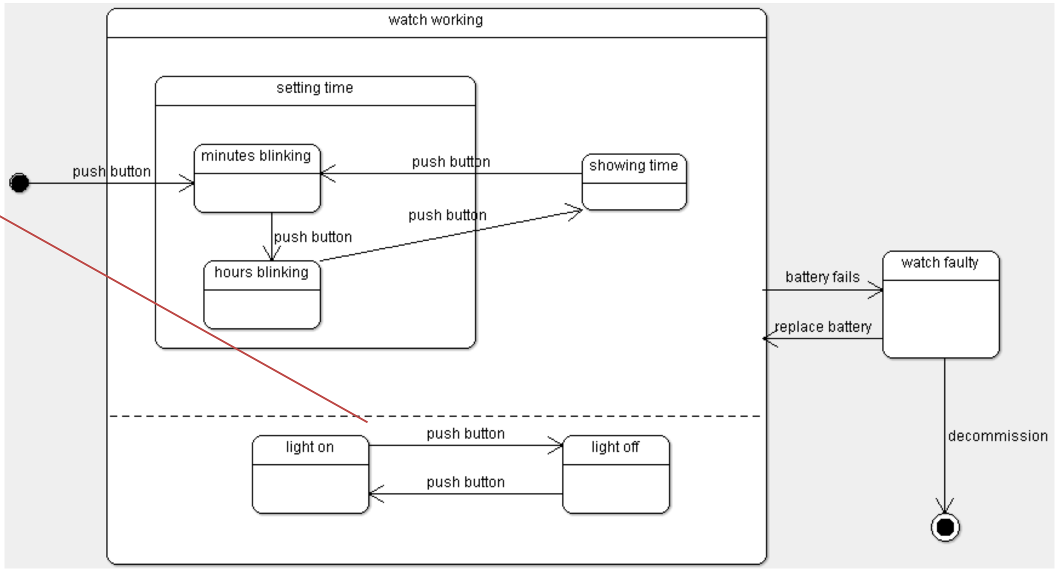
Sequential (Mutually Exclusive Disjoint) Substates:

Modeled element can be in one of the substates at a time, cannot be in two substates concurrently.

STATE DIAGRAM: NESTED STATES


Make Breakfast
Cook Eggs
Make Toast
Brew Coffee
Pour Coffee
Setup
Serve Breakfast
Cleanup
Synchronization Bar used to model Splitting of Control
Synchronization Bar used to model Synchronization

Concurrent state



DYNAMIC MODELLING: DIFFICULTIES AND RISKS

Dynamic modeling a large system is difficult:

- There are a very large number of possible paths a system can take
- It is hard to choose which classes to allocate to each behavior

SUMMARY

- We represent class relations using association, multiplicity, aggregation and composition
- Interaction diagrams (only sequence) show the order in which several objects communicate with each other.
- State diagrams show the states in which objects/systems can be found, as well as what causes them to change state.