Modellistica Medica

Maria Grazia Pia

INFN Genova

Scuola di Specializzazione in Fisica Sanitaria
Genova
Anno Accademico 2002-2003

Lezione 7

UML

Behavioural diagrams

Behavioural Diagrams

Used to visualize, specify, construct, document dynamic aspects of system

- Use case diagram
- Sequence diagram
- Collaboration diagram
- Statechart diagram
- Activity diagram

Interaction diagrams

A pattern of interaction among instances is shown on an **interaction diagram**

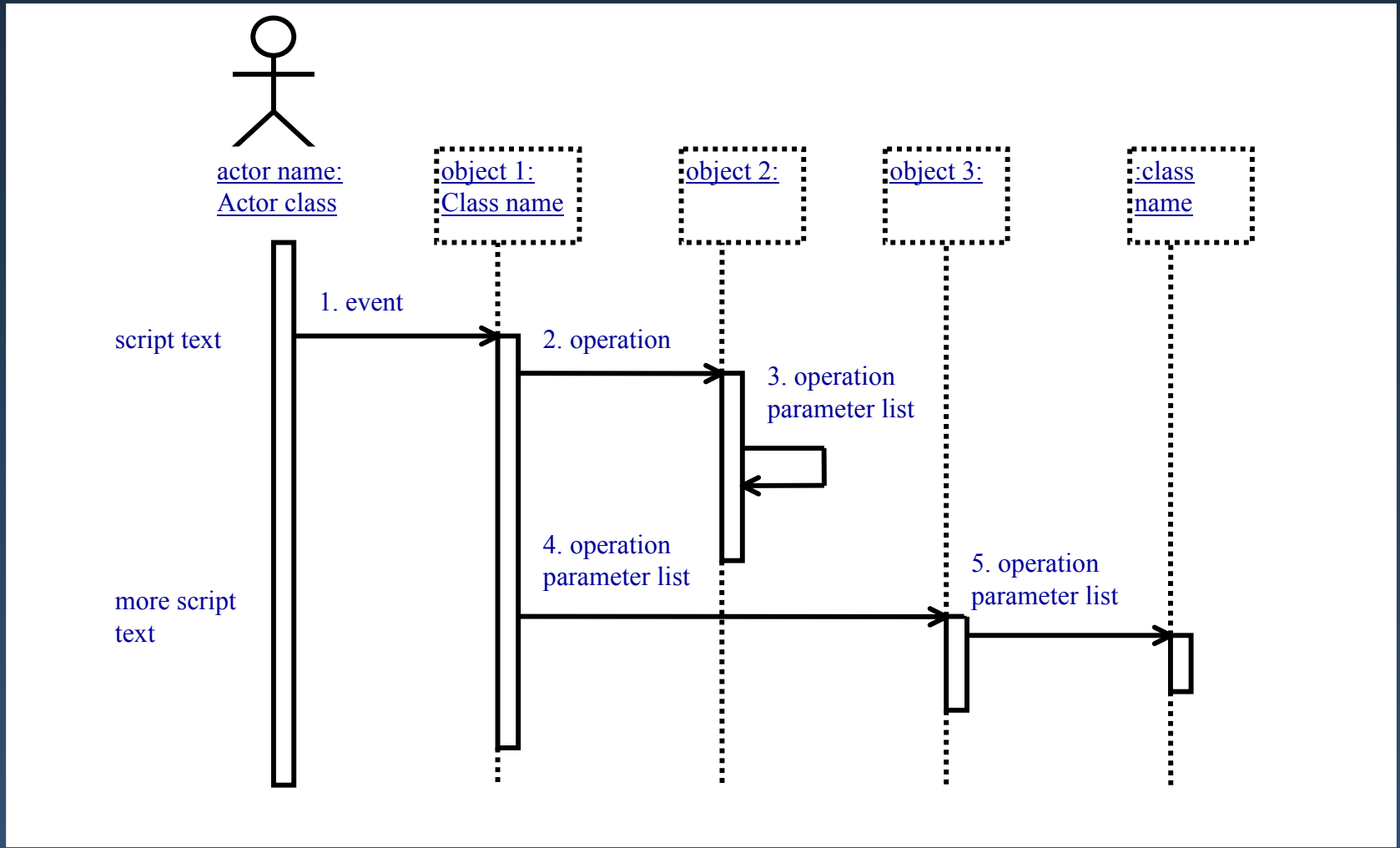
Interaction diagrams come in **two forms,** based on the **same underlying information**, but each form emphasizing a particular aspect of it

The two forms are sequence diagrams and collaboration diagrams.

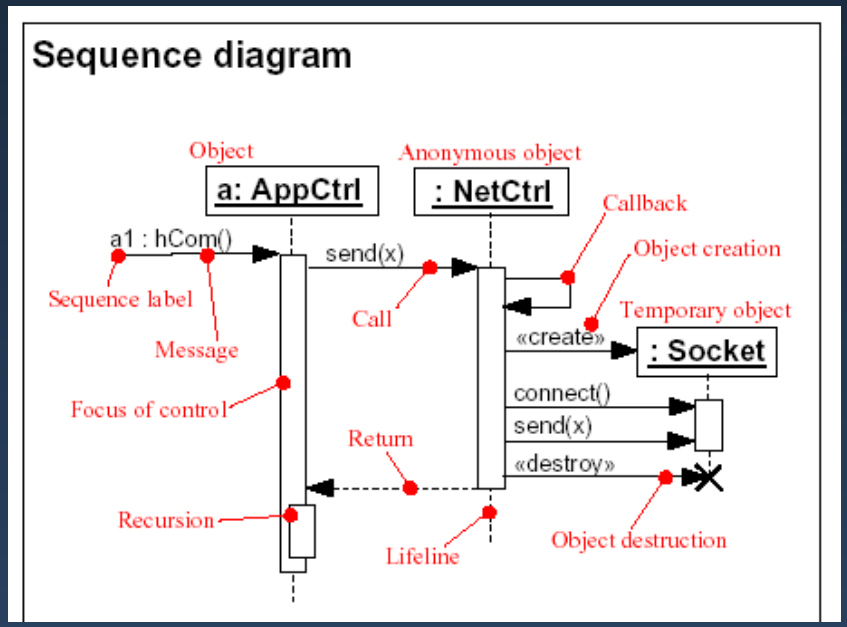
Sequence diagrams emphasize the time ordering of messages

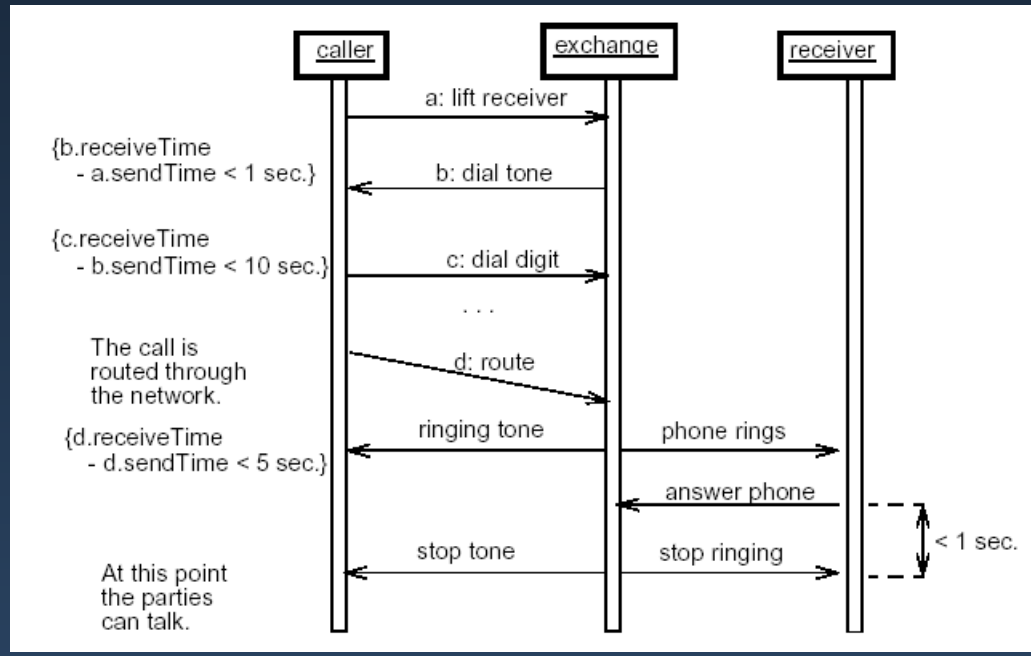
Collaboration diagrams show the relationships among instances and emphasize the organization of the objects that participate in the interaction

Sequence diagram



Sequence diagrams





A sequence diagram has two dimensions:

1) the **vertical** dimension represents **time**

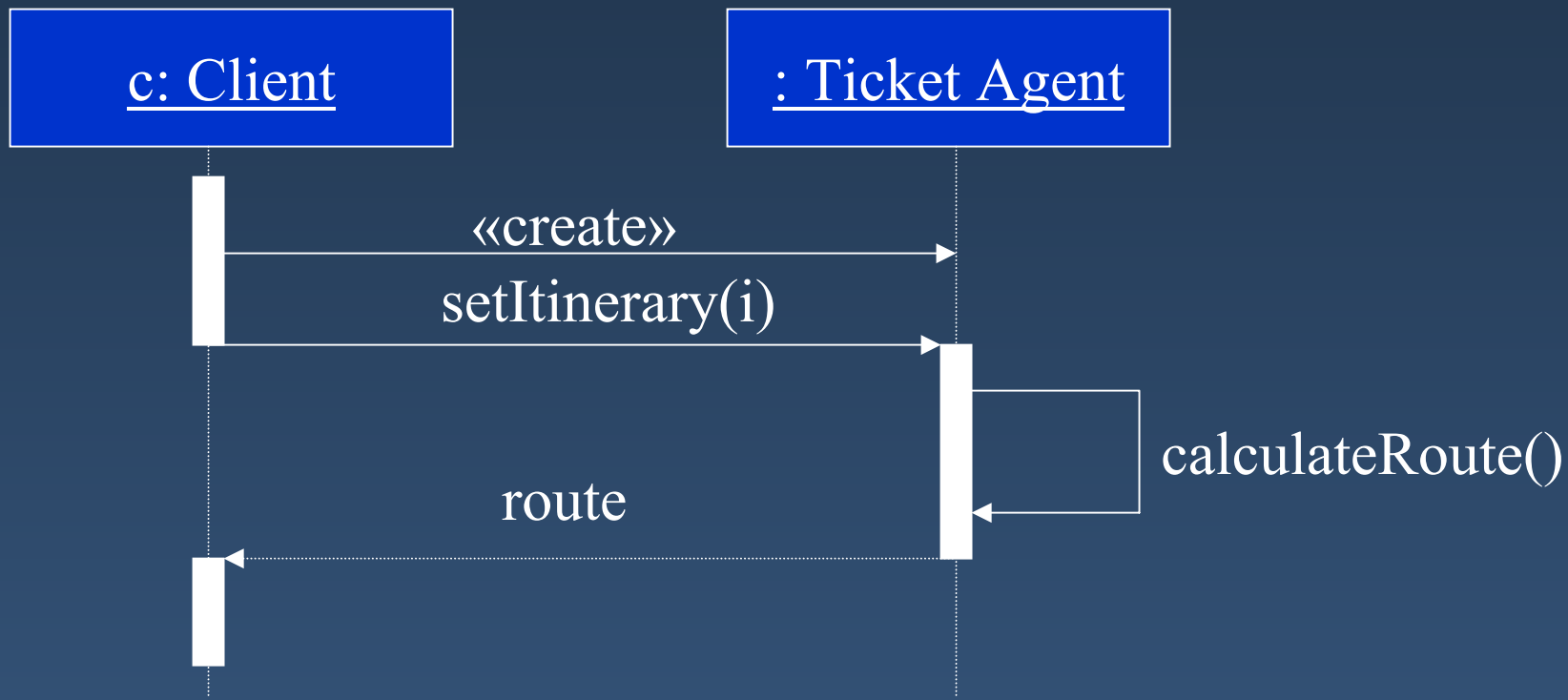
2) the **horizontal** dimension represents **different objects**

- *Normally time proceeds down the page*
- *Usually only time sequences are important*
- *There is no significance to the horizontal ordering of the objects*
- *Objects can be grouped into "swimlanes" on a diagram*

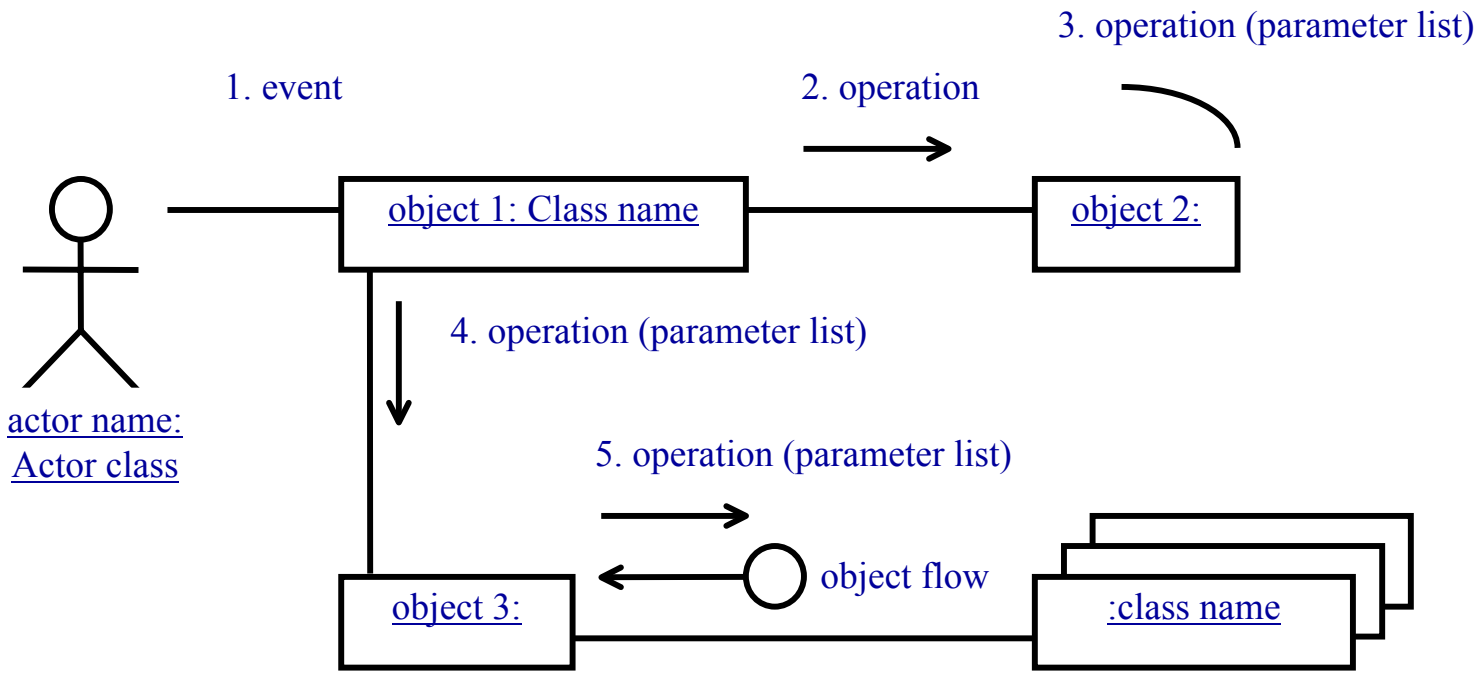
Sequence Diagram Notation

A **lifeline** is a vertical dashed line that represents the lifetime of an object

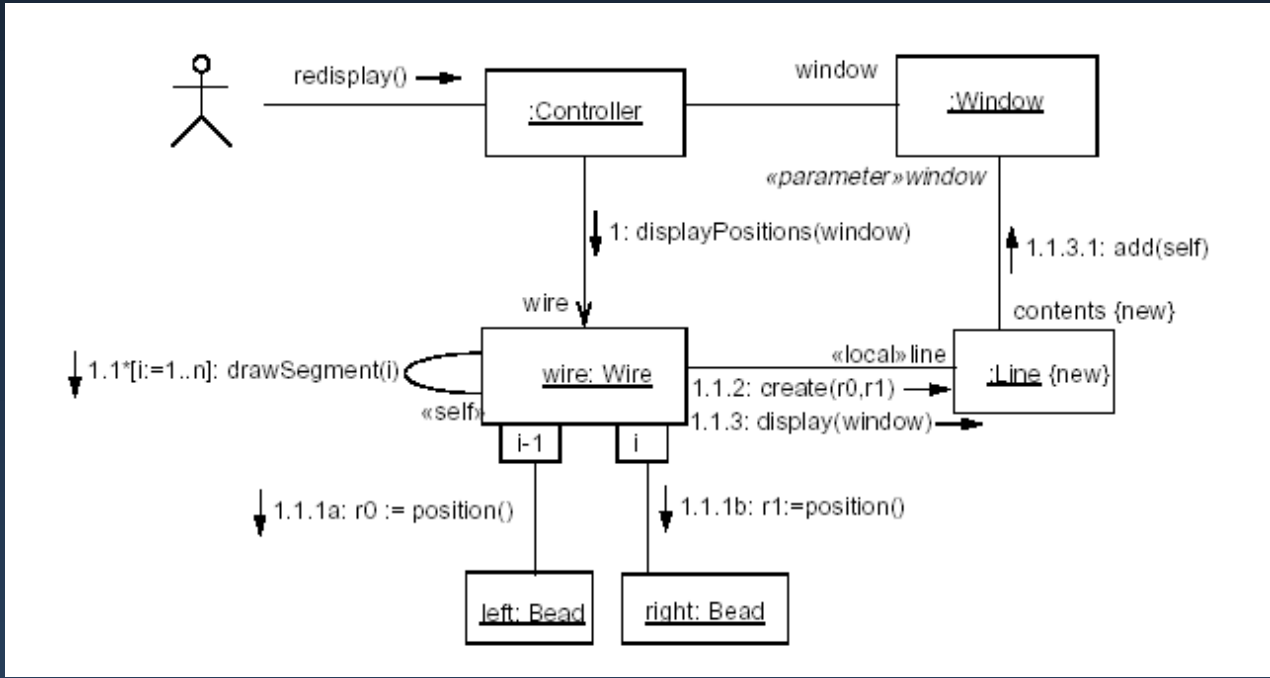
A **focus of control** is a tall, thin rectangle that shows the period of time during which an object is performing an action



Collaboration diagram



Collaboration diagrams

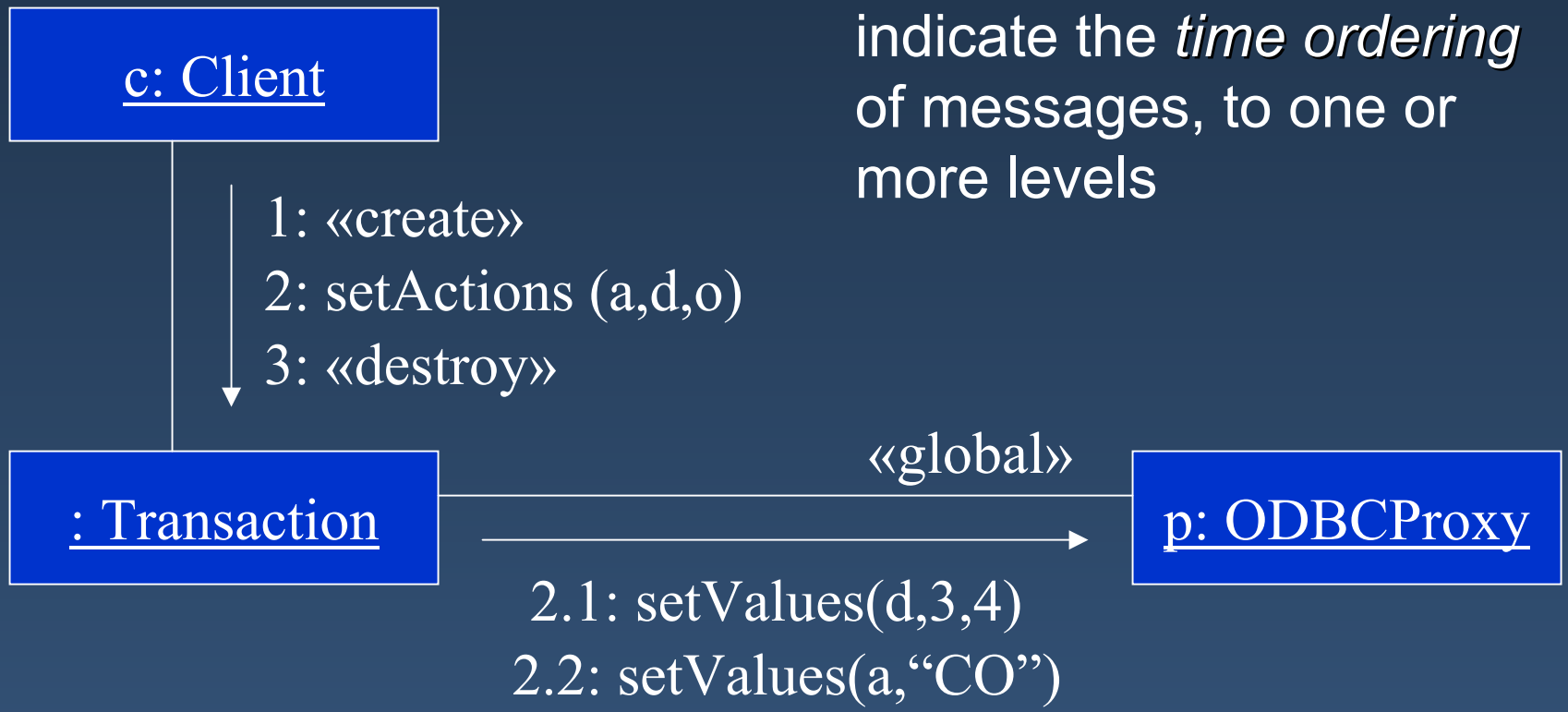


A collaboration diagram shows an **interaction organized around the roles** in the interaction and their links to each other.

Unlike a sequence diagram, a collaboration diagram shows the relationships among the objects playing the different roles.

On the other hand, a collaboration diagram does not show time as a separate dimension, so the sequence of interactions and the concurrent threads must be determined using sequence numbers.

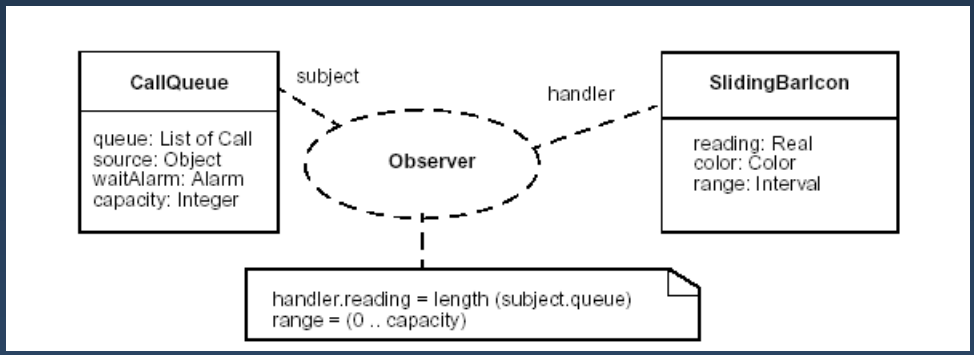
Collaboration Diagram Notation

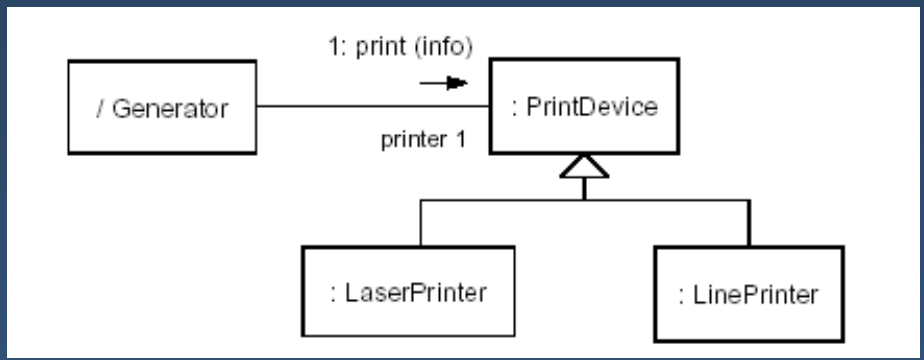


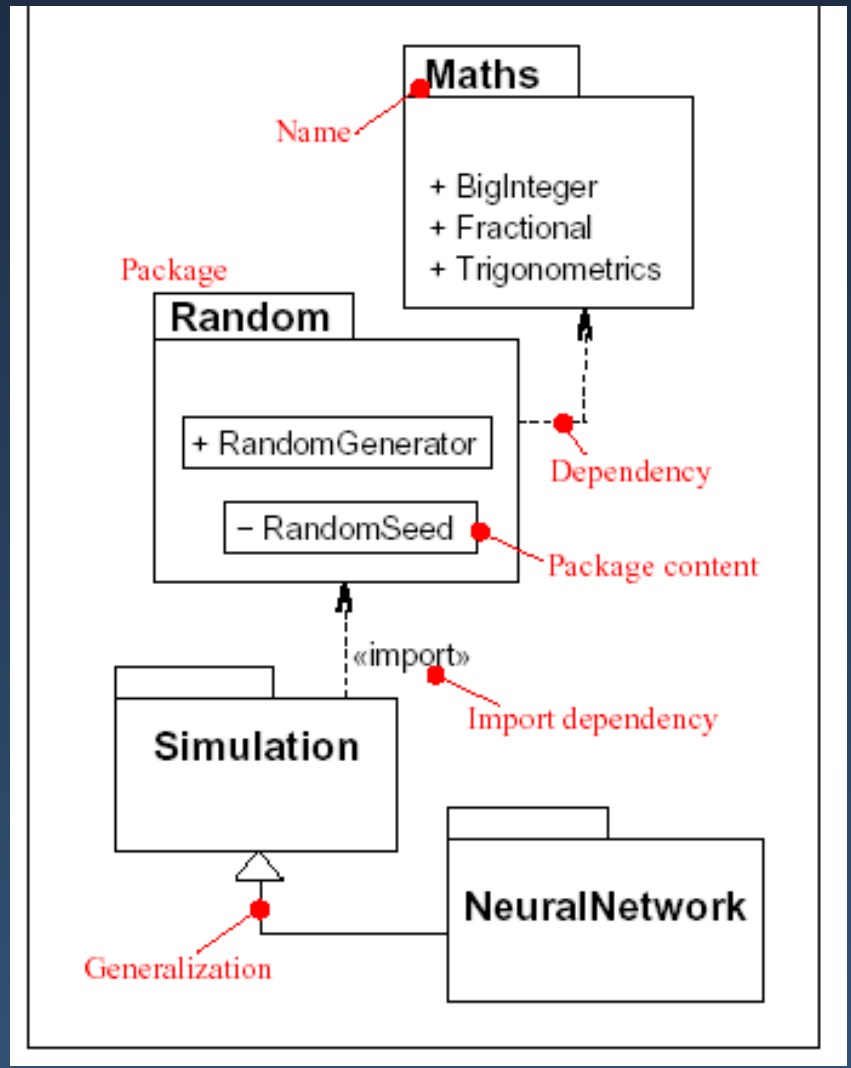
Sequence numbers indicate the *time ordering* of messages, to one or more levels

Collaboration

A collaboration **realizes** a classifier, such as a class or a use case, or an operation.



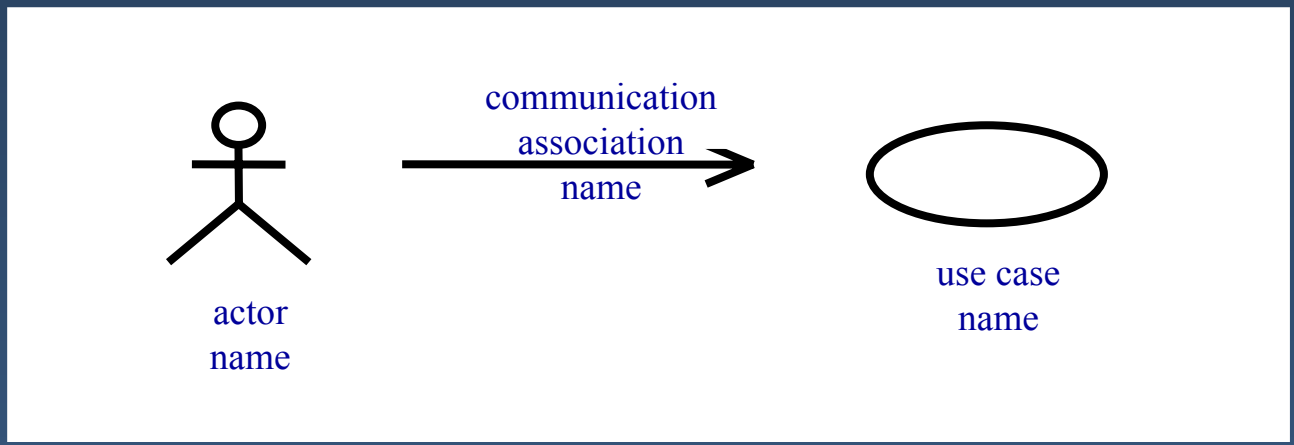




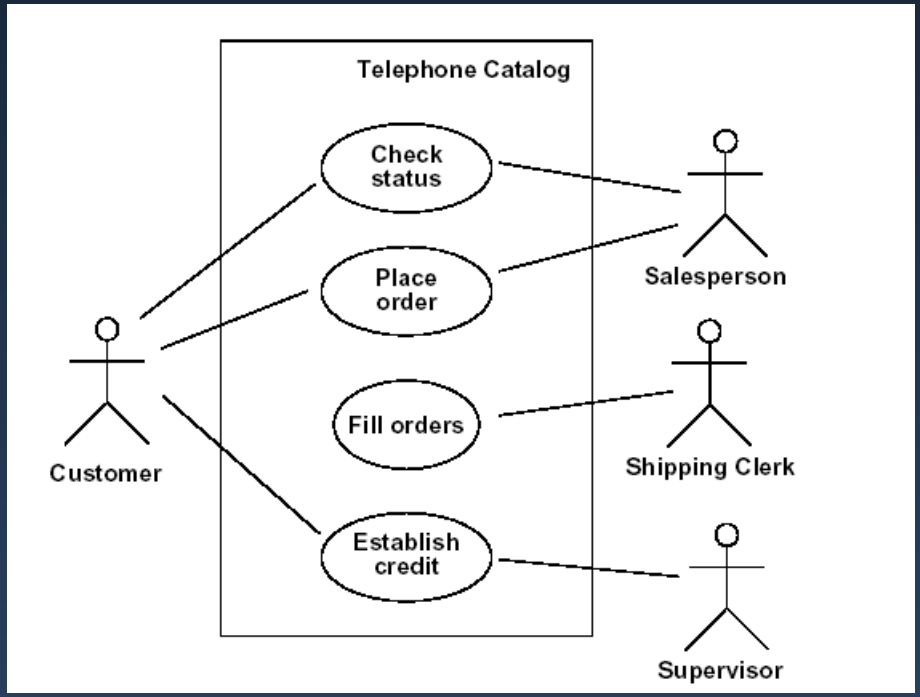
Use case, actor

A **use case** is a **coherent unit of functionality** provided by a system, a subsystem, or a class as manifested by sequences of messages exchanged among the system and one or more outside **interactors** (called **actors**) together with actions performed by the system

An **actor** is a **coherent set of roles** that human and/or non-human users of use cases play when interacting with those use cases



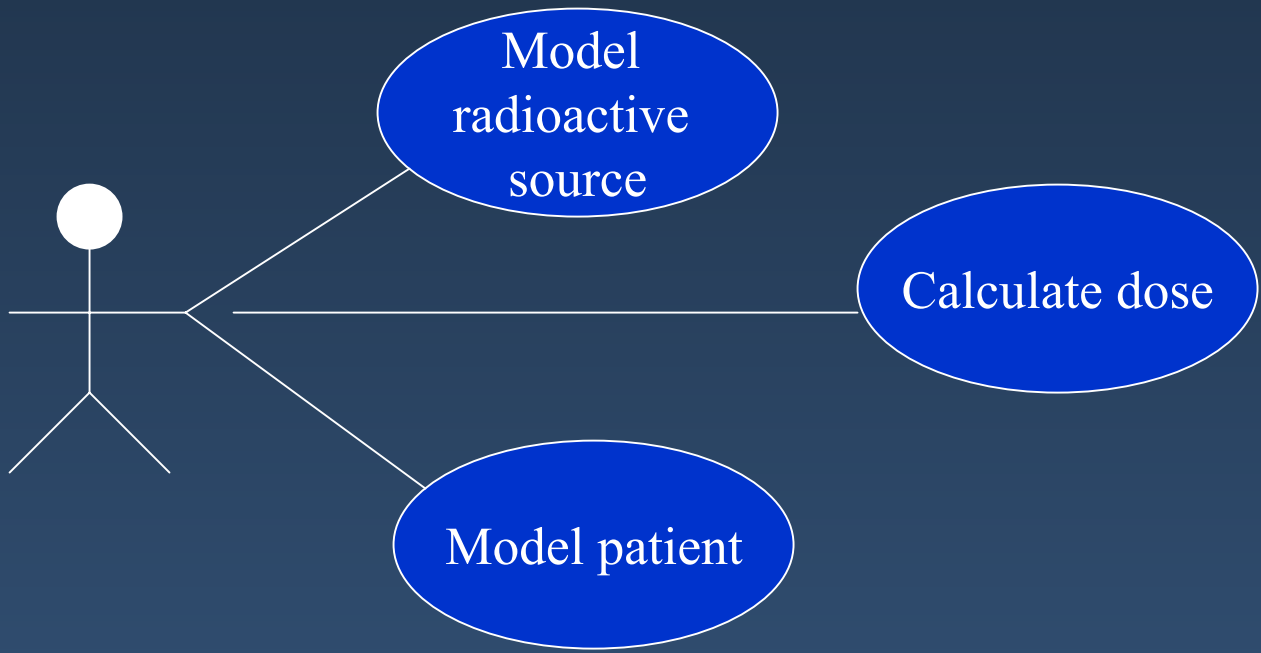
Use case diagram



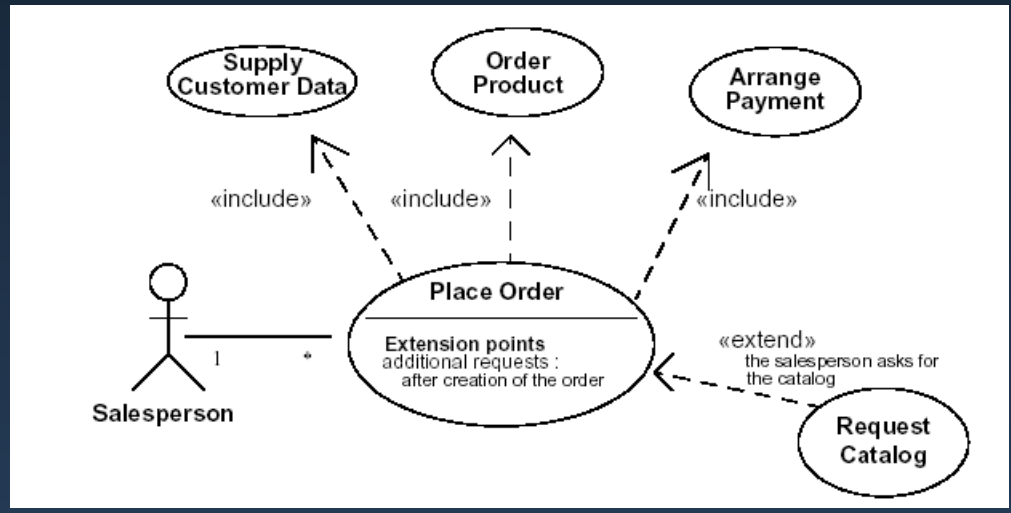
Use case diagrams show actor and use case together with their relationships.

The use cases represent functionality of a system or a classifier, like a subsystem or a class, as manifested to external interactors with the system or the classifier.

Simple Use Case Diagram



Use case relationships



Association – The participation of an actor in a use case, i.e. instances of the actor and instances of the use case communicate with each other. This is the only relationship between actors and use cases.

Extend – An extend relationship from use case A to use case B indicates that an instance of use case B may be extended (subject to specific conditions specified in the extension) by the behavior specified by A.

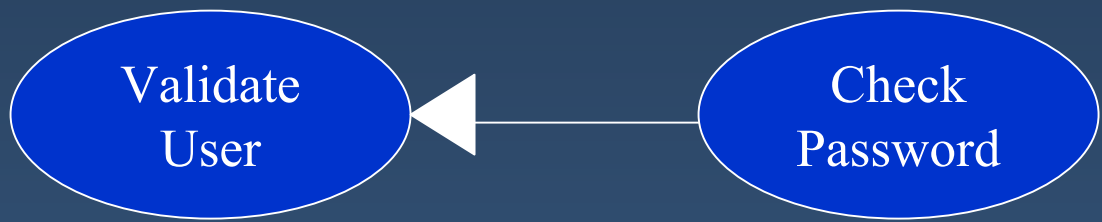
Generalization – A generalization from use case A to use case B indicates that A is a specialization of B.

Include – An include relationship from use case A to use case B indicates that an instance of the use case A will also include the behavior as specified by B.

Use Case Generalization

One can generalize use cases just like one generalize classes:

- the child use case inherits the behavior and meaning of the parent use case
- can add to or override that behavior



Include

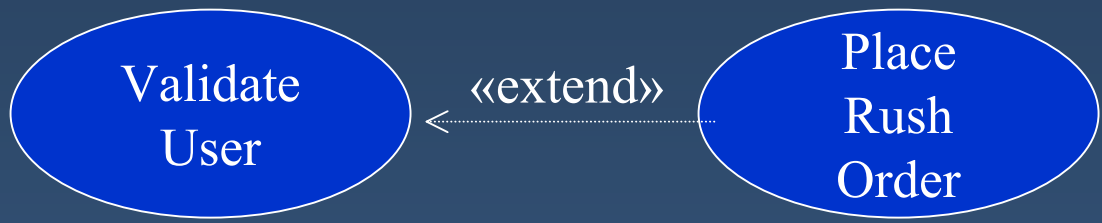
The «include» stereotype indicates that one use case “includes” the contents of another use case

This allows to factor out frequent common behavior

Extend

The «extend» stereotype indicates that one use case is “extended” by another use case.

This enables one to factor out infrequent common behavior

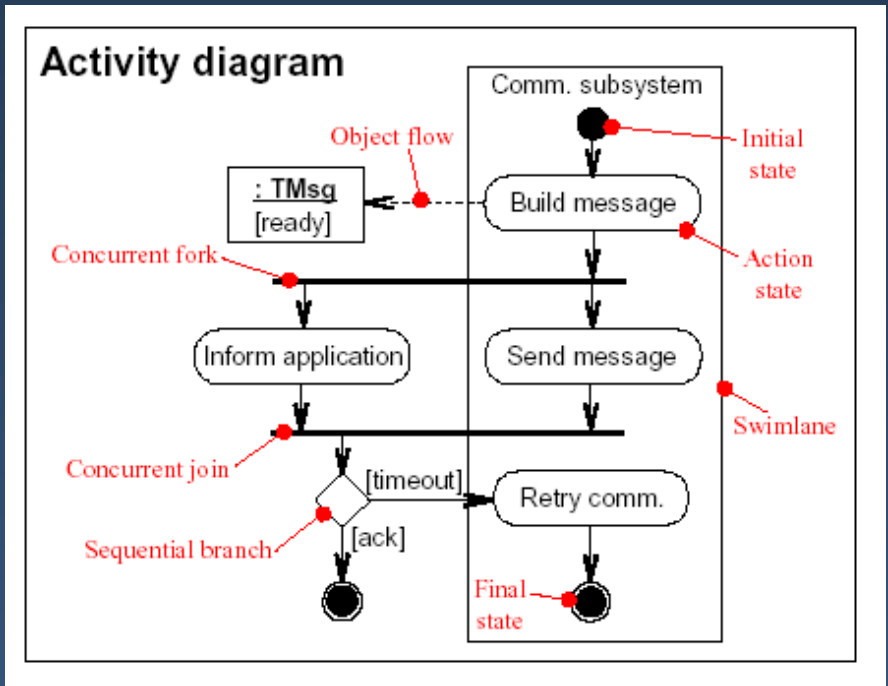


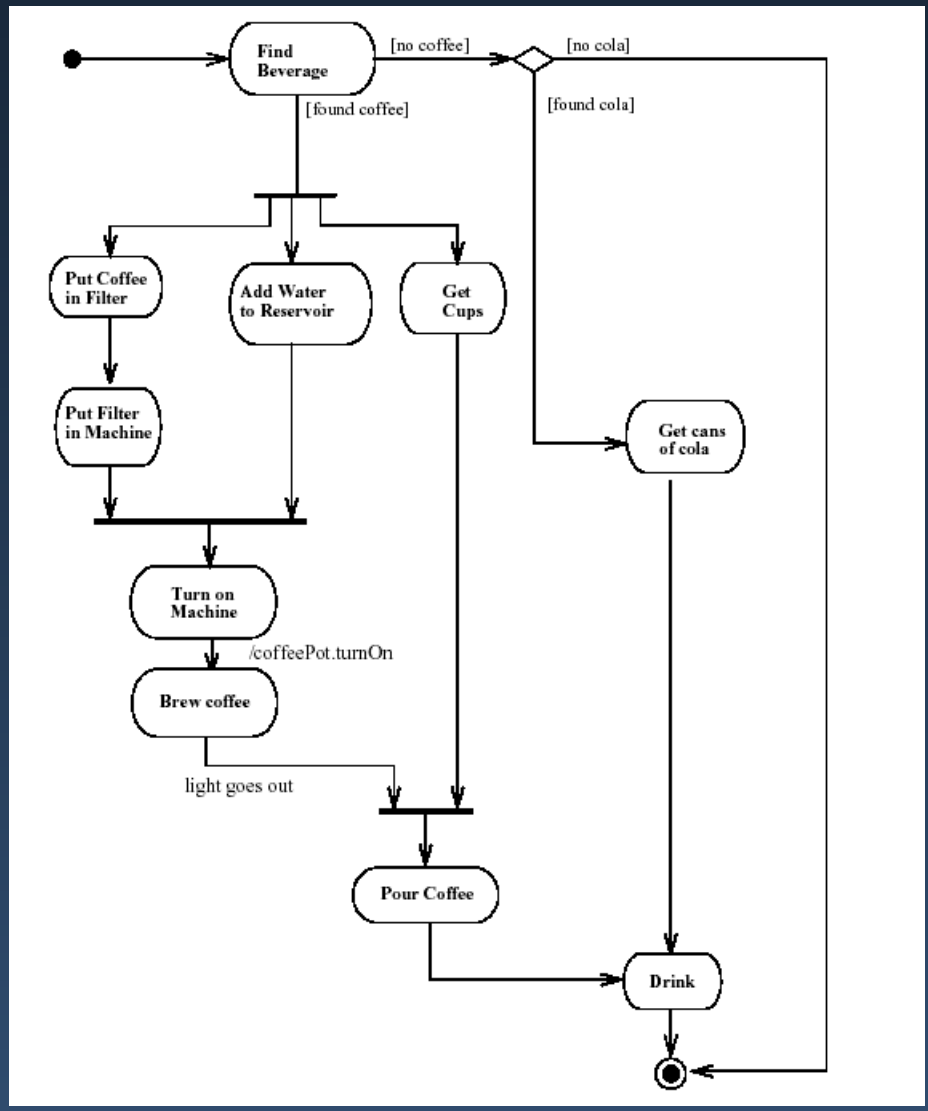
Why Activity Diagrams?

- An activity diagram is useful for modeling workflows and the details of operations
  - also used to model software process workflows
- Resembles a flowchart
- While an interaction diagram looks at the objects that pass messages, an activity diagram looks at the operations that are passed among objects

Activity diagram

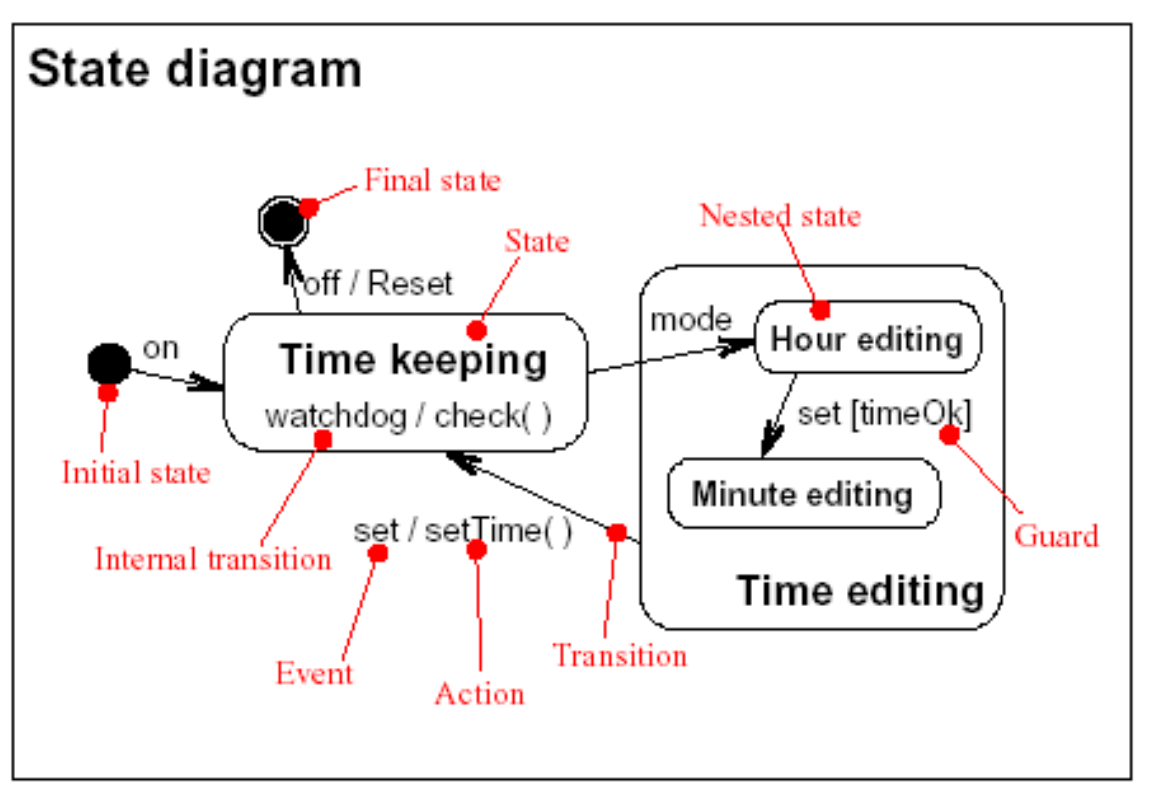
The purpose of this diagram is to focus on **flows driven by internal processing** (as opposed to external events).





Use activity diagrams in situations where all or most of the events represent the completion of internally-generated actions *(that is, procedural flow of control)*

State diagram



Implementation diagrams

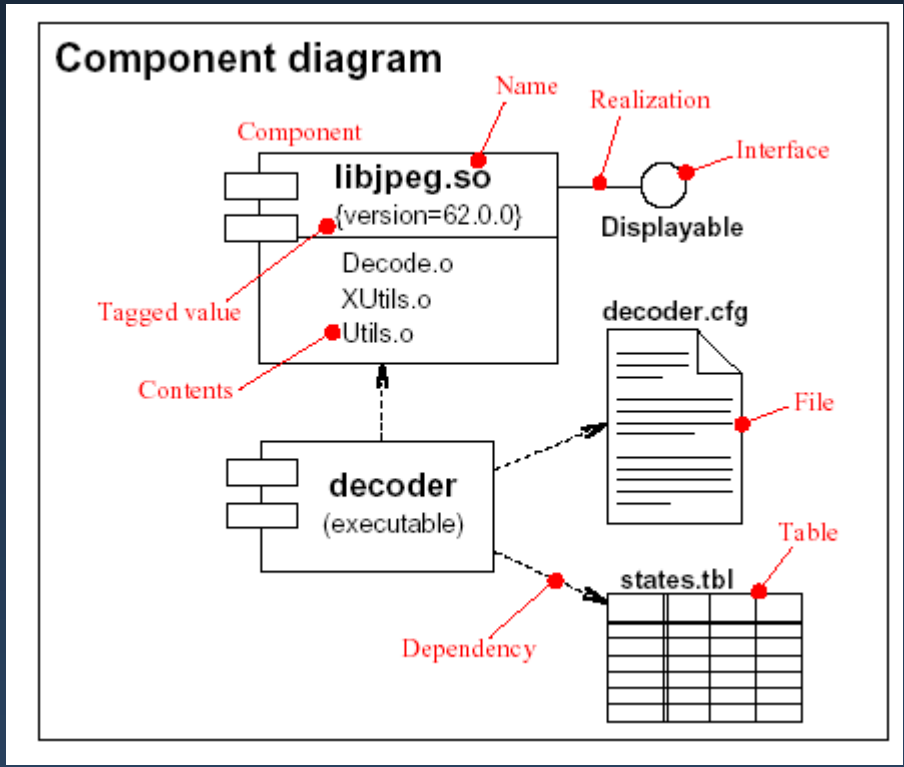
Implementation diagrams show aspects of implementation, including source code structure and run-time implementation structure.

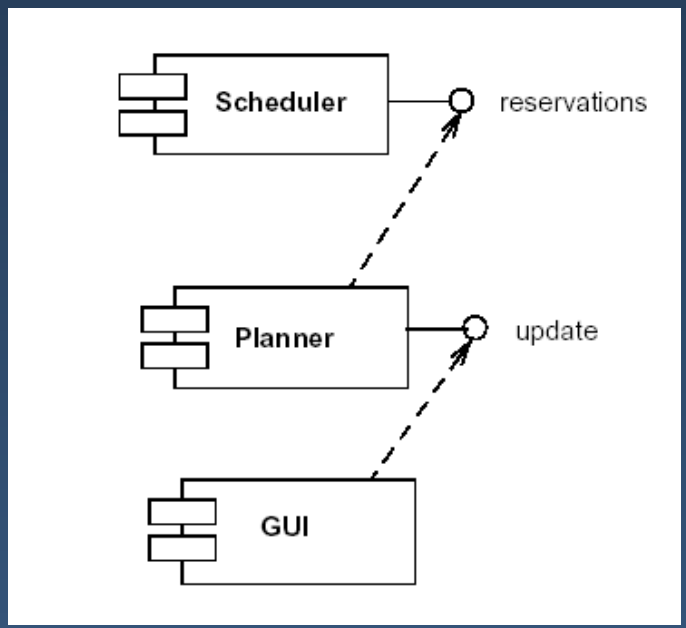
They come in two forms:

1) component diagrams show the structure of the code itself
2) deployment diagrams show the structure of the run-time system.

Component diagram

- A **component diagram** shows the dependencies among software components, including
  - source code components
  - binary code components
  - executable components

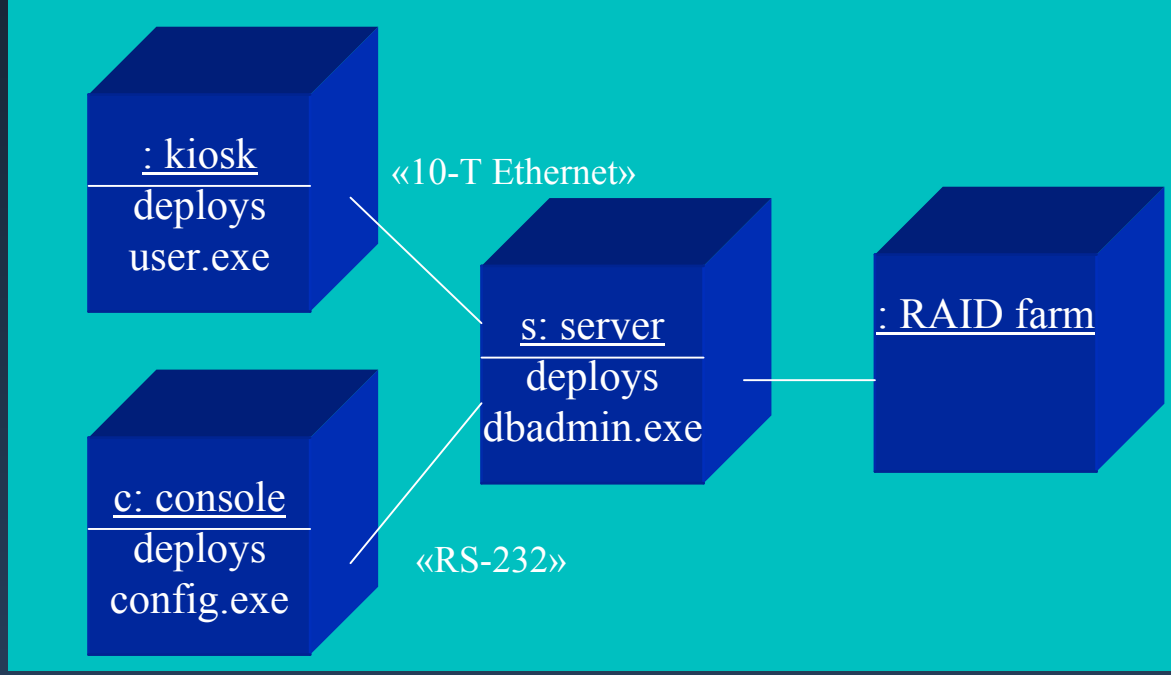


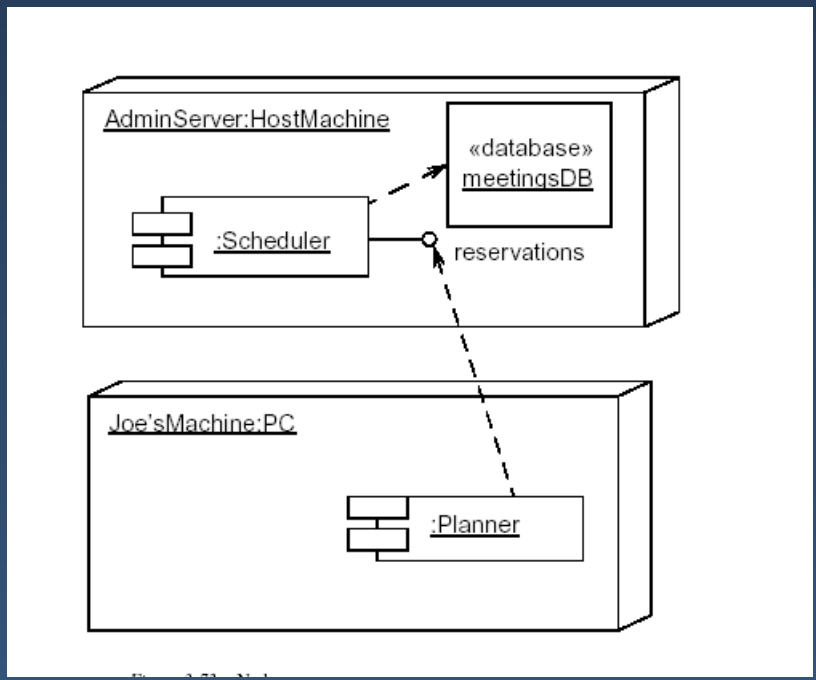


A **component** is a physical, replaceable part of a system that conforms to, and provides the realization of, a set of interfaces

A **software module** may be represented as a component type

Deployment diagrams



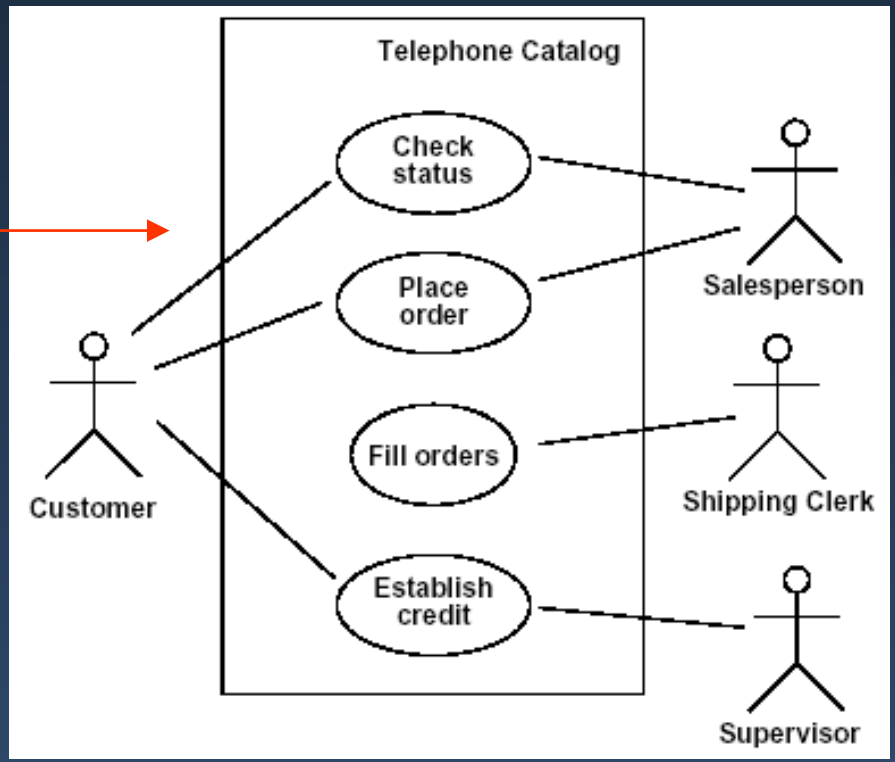


Deployment diagrams show the configuration of run-time processing elements and the software components, processes, and objects that live on them

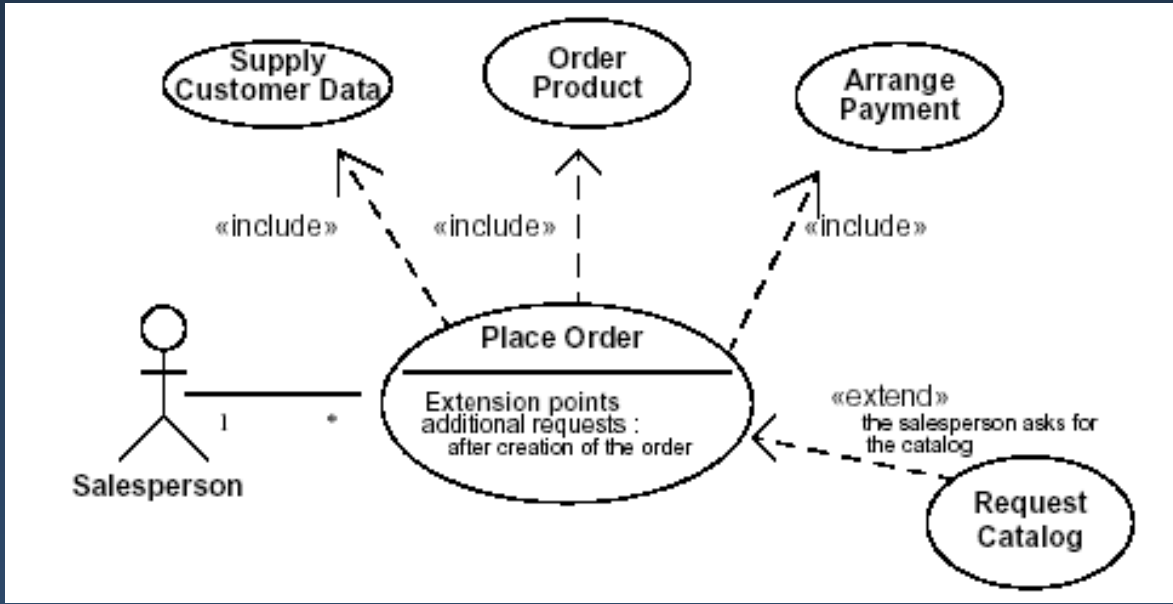
Software component instances represent run-time manifestations of code units

Basics

What is?

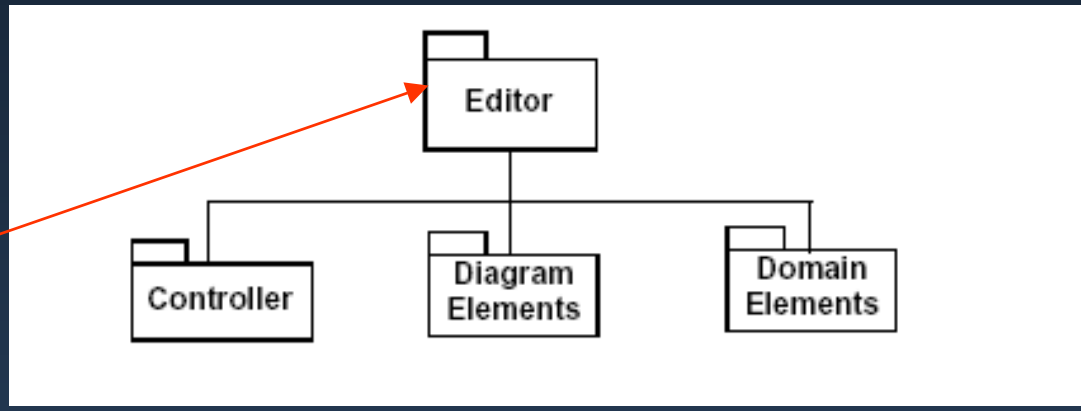


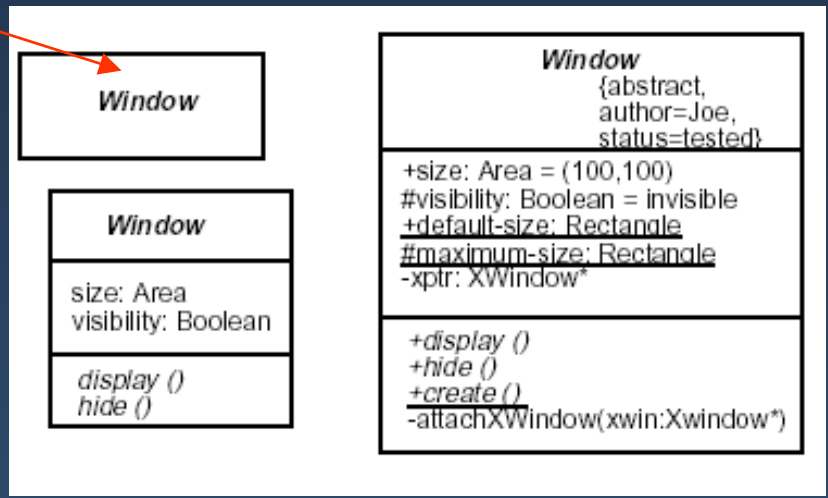
Please explain



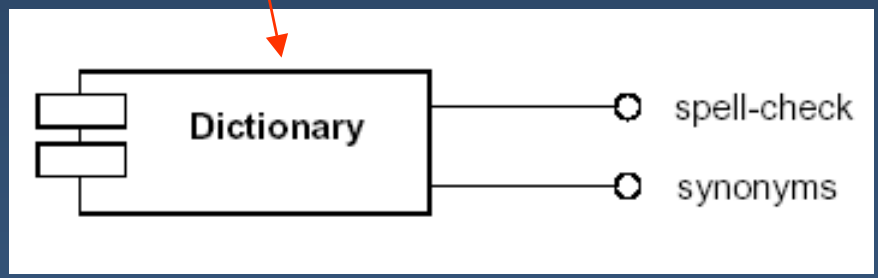
UML

What is?



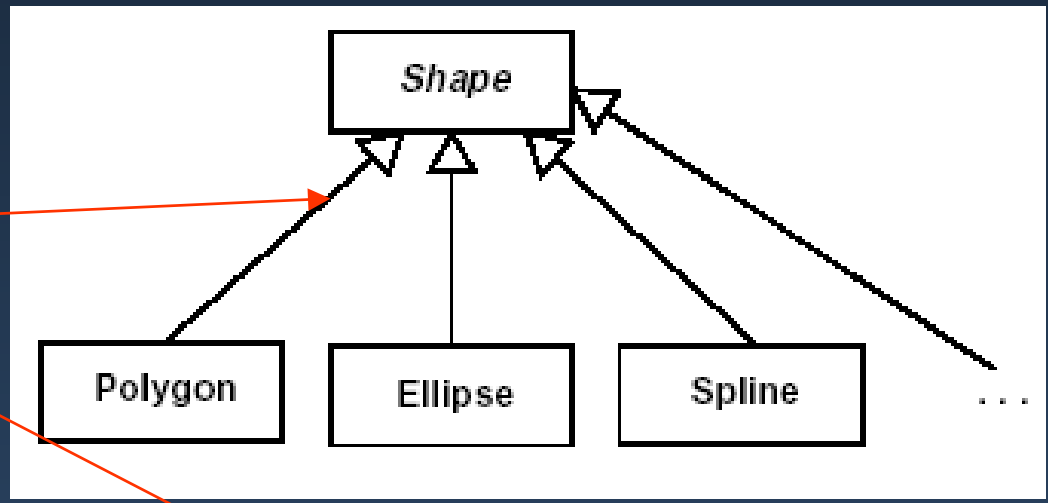


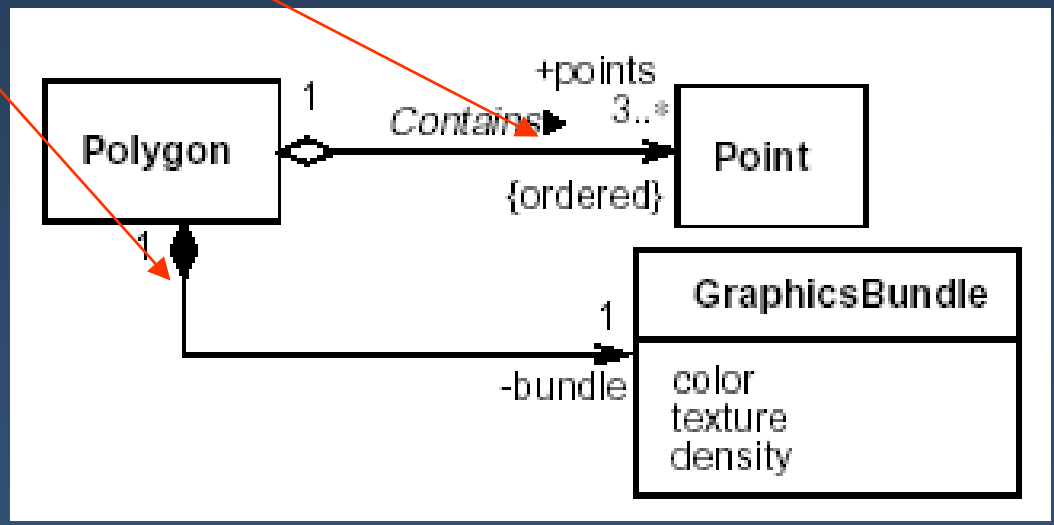
What do +,- # mean?



Please explain

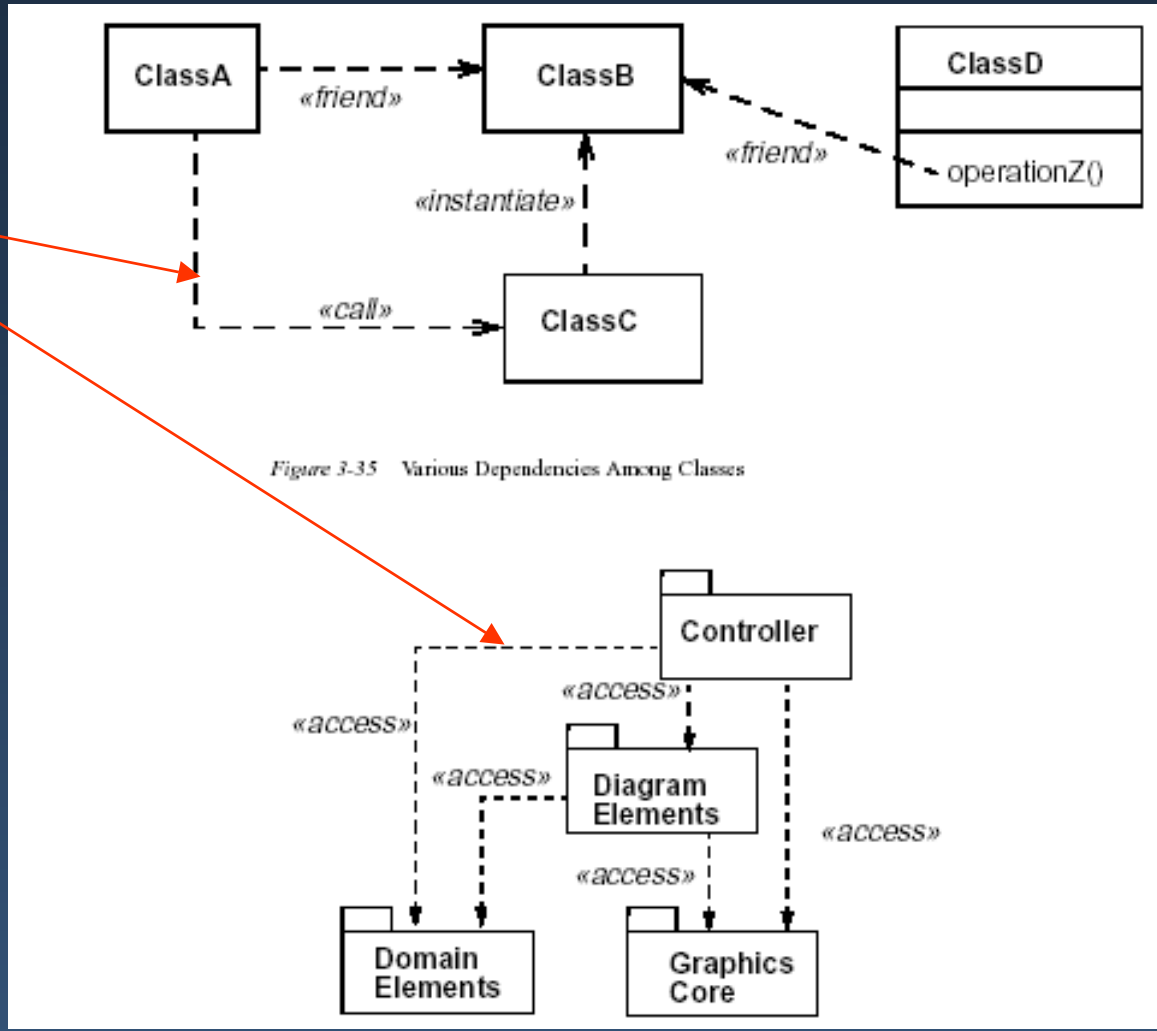
What is?





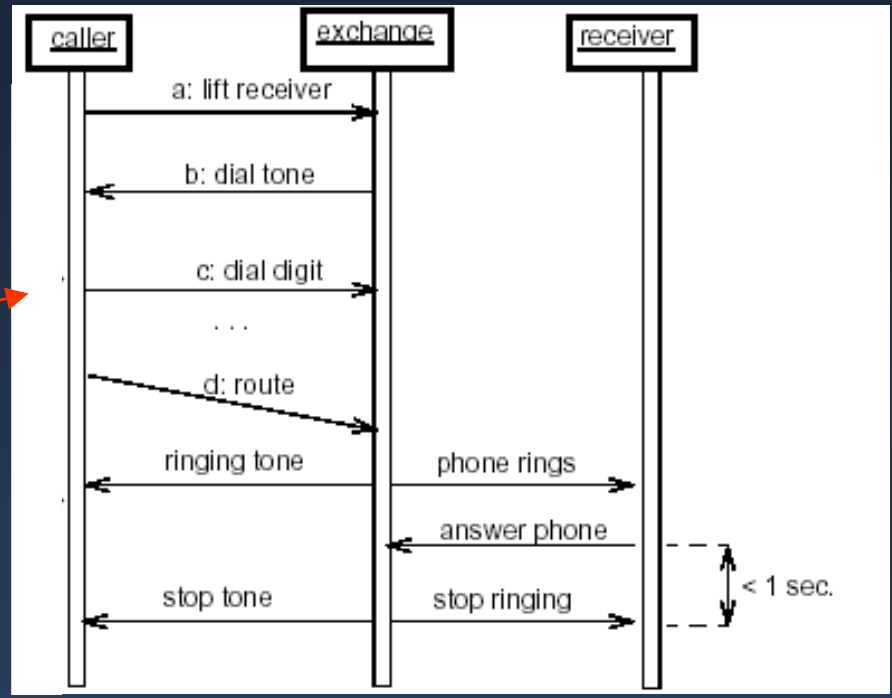
UML

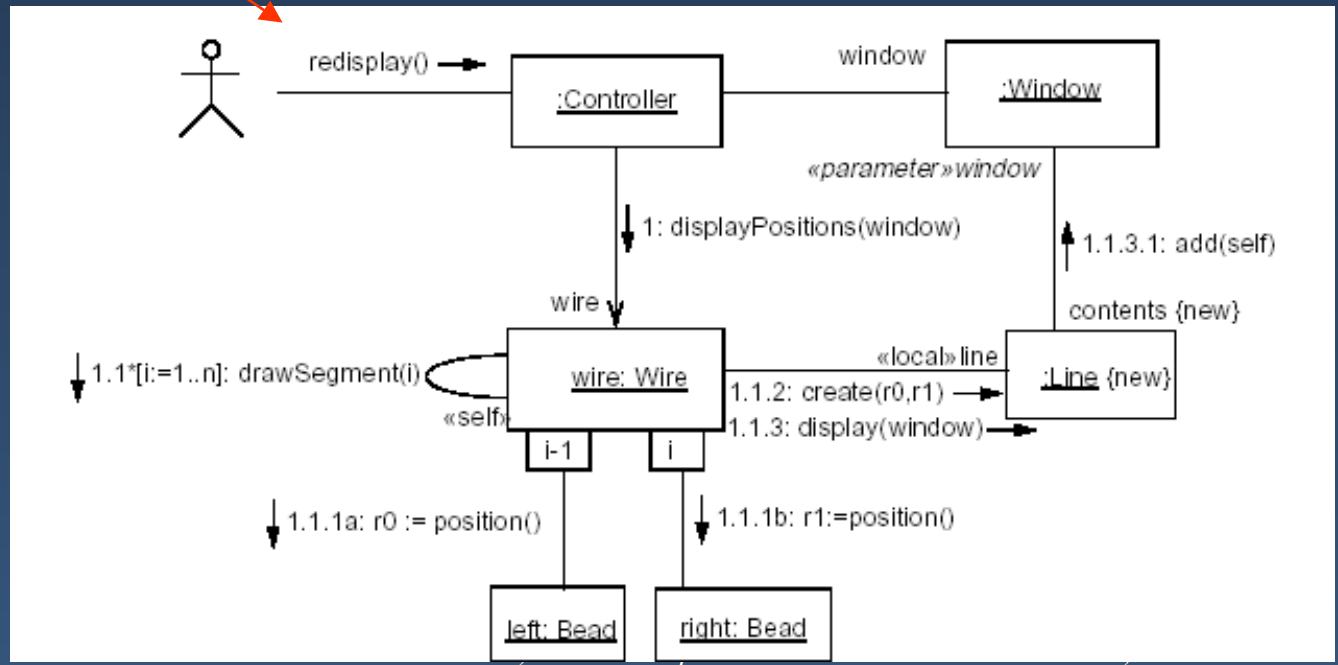
What is?


ClassA
«friend»
ClassB
ClassD
operationZ()
«friend»
«instantiate»
«call»
ClassC
Figure 3-35 Various Dependencies Among Classes
Controller
«access»
«access»
«access»
Diagram Elements
«access»
«access»
Domain Elements
Graphics Core

UML

What is?


caller
exchange
receiver
a: lift receiver
b: dial tone
c: dial digit
. . .
d: route
ringing tone
phone rings
answer phone
< 1 sec.
stop tone
stop ringing


redisplay()
:Controller
window
:Window
«parameter»window
1: displayPositions(window)
1.1.3.1: add(self)
wire
contents {new}
1.1*[i:=1..n]: drawSegment(i)
«self»
wire: Wire
«local» line
:Line {new}
1.1.2: create(r0,r1)
1.1.3: display(window)
i-1
i
1.1.1a: r0 := position()
1.1.1b: r1:=position()
left: Bead
right: Bead