UML FUNDAMENTALS

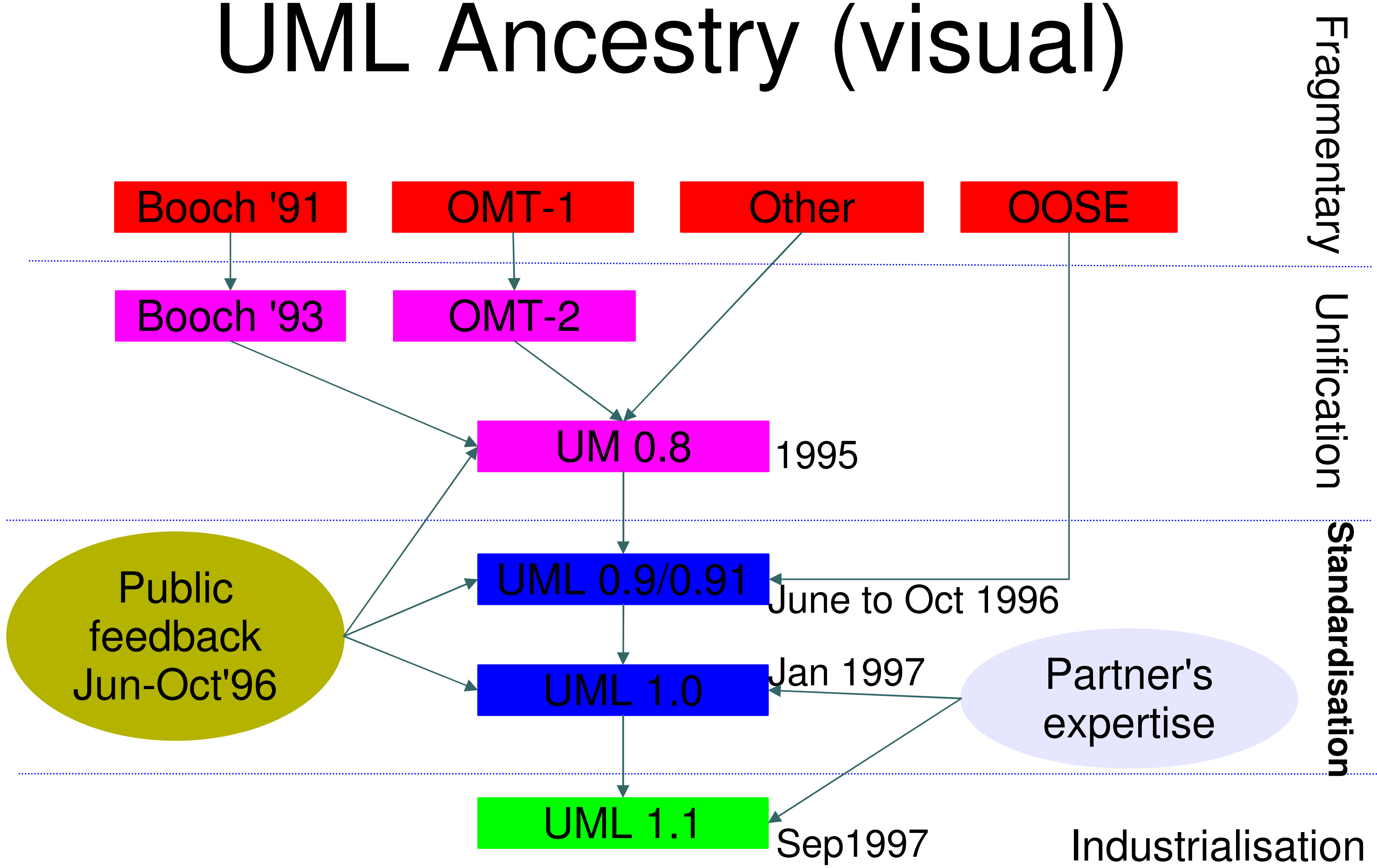
UML

Unified Modelling Language

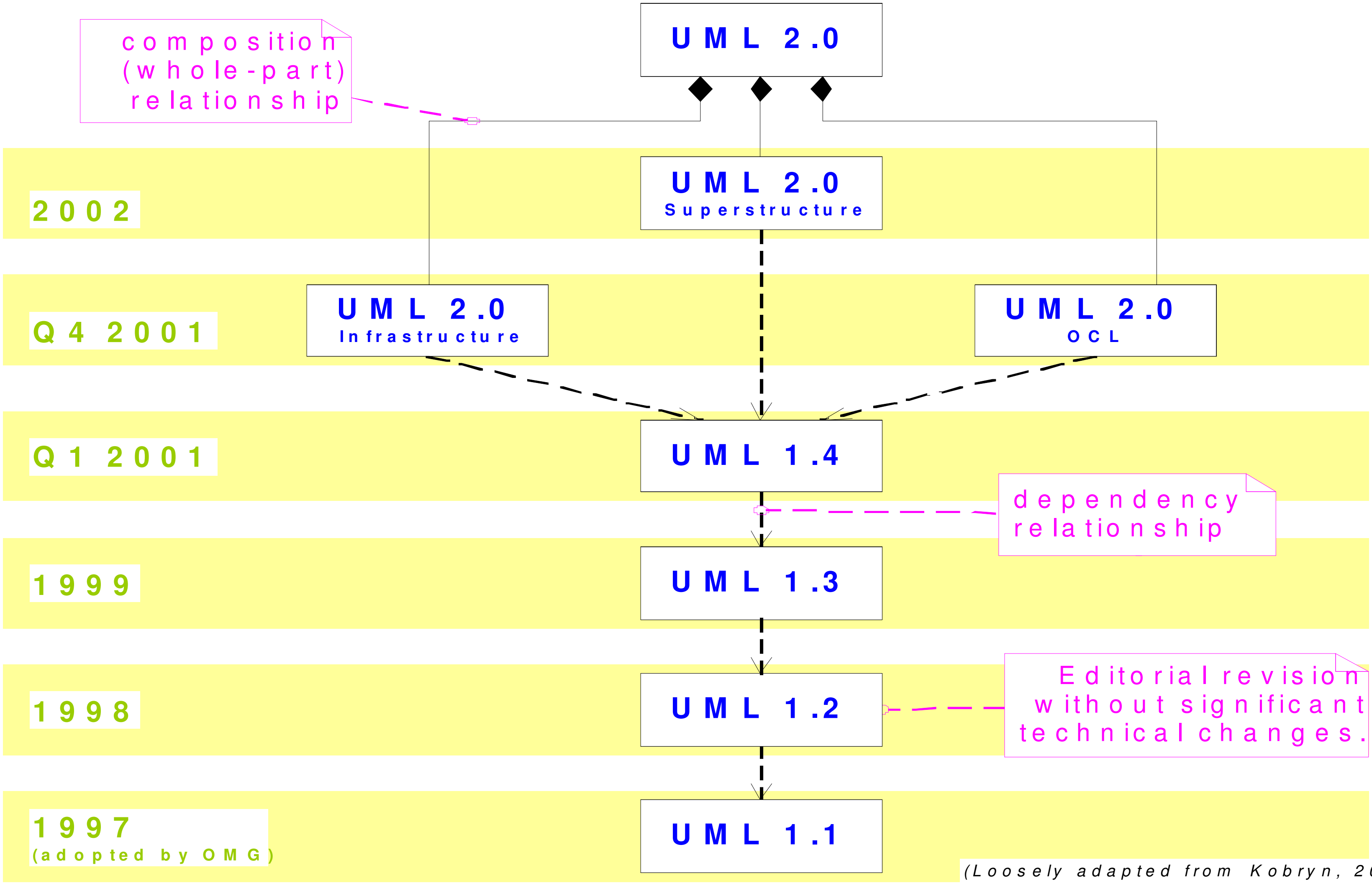
Visualising and documenting analysis and design effort.

- Unified because it …
  - Combines main preceding OO methods (Booch by *Grady Booch, OMT by Jim Rumbaugh and OOSE by Ivar Jacobson)*
- Modelling because it is …
  - Primarily used for visually modelling systems. Many system views are supported by appropriate models
- Language because …
  - It offers a syntax through which to express modelled knowledge

UML Ancestry (visual)



Further (latest) UML Evolution



(Loosely adapted from Kobryn, 2001)

UML Partners

The list is quite an impressive one:

- Hewlett-Packard
- IBM
- Microsoft
- Oracle
- i-Logix
- Intelli Corp.
- MCI Systemhouse
- ObjectTime
- Unisys
- Sterling Software
- Rational Software
- ICON computing
- Platinum Technology
- *and others…*

*and so…*What is UML?

Based on the previous three slides…

- A language for capturing and expressing knowledge
- A tool for system discovery and development
- A tool for visual development modelling
- A set of well-founded guidelines
- A milestone generator
- A popular (therefore supported) tool

and…What UML is not!

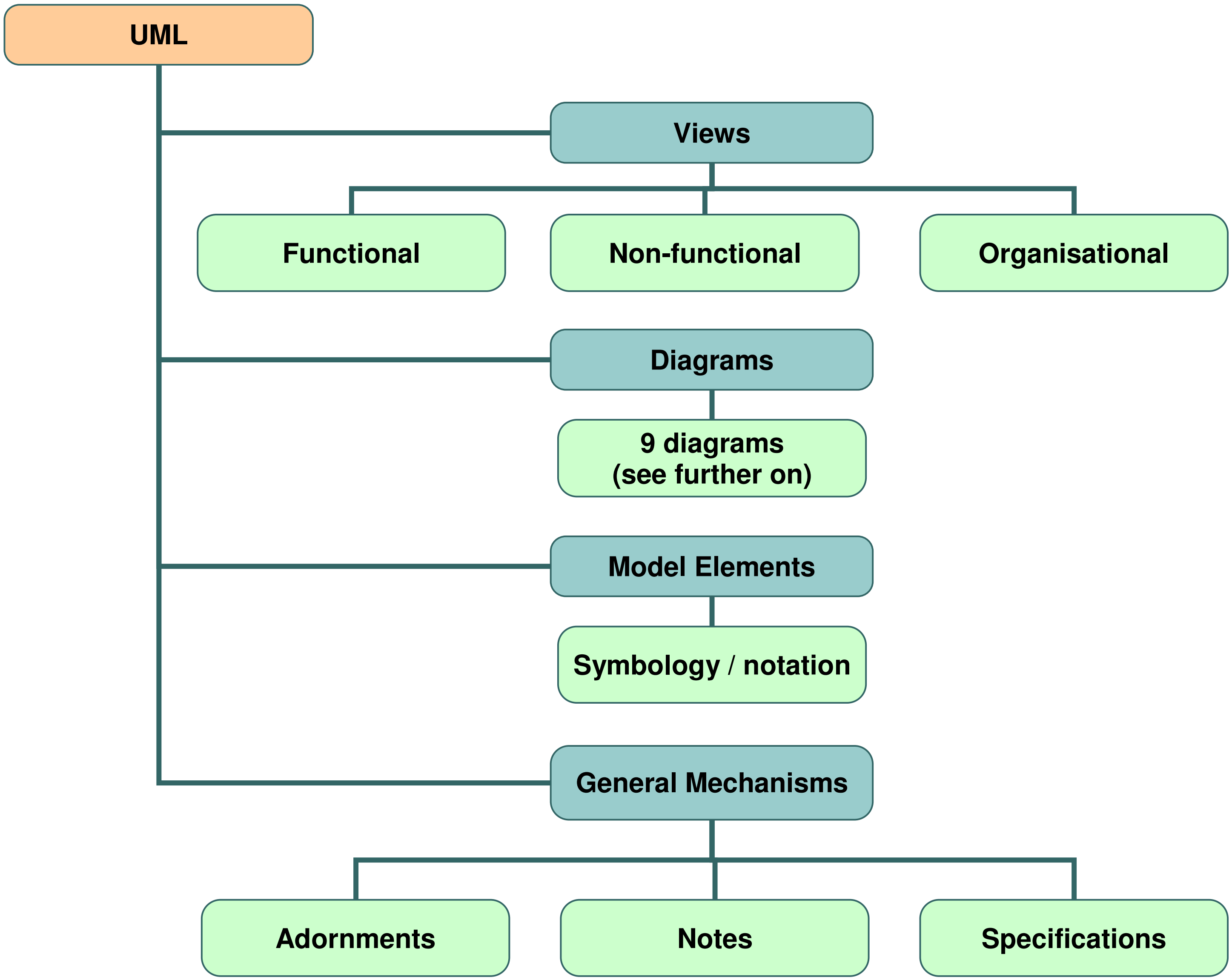
- A visual programming language or environment
- A database specification tool
- A development process (i.e. an SDLC)
- A panacea
- A quality guarantee

What UML can do for you

Help you to:

- Better think out and document your system before implementing it
- “forecast” your system
- Determine islands of reusability
- Lower development costs
- Plan and analyse your logic (system behaviour)
- Make the right decisions at an early stage (before committed to code)
- Better deploy the system for efficient memory and processor usage
- Easier maintenance/modification on well documented systems
- Lower maintenance costs
- Establish a communication standard
- Minimise “lead-in” costs

UML components



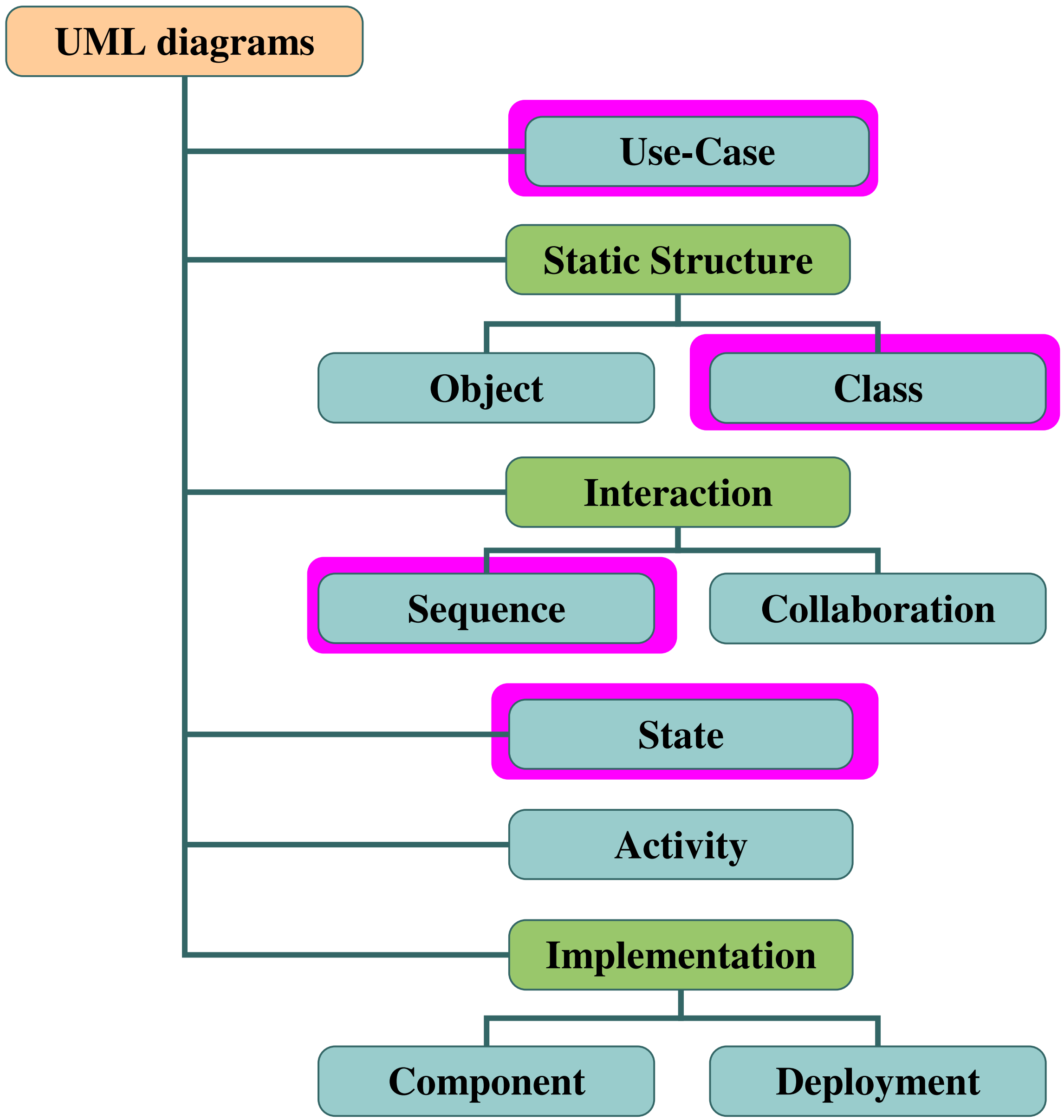
The Case “for” Diagrams

- Aesthetic
- Descriptive
- Compressive
- Simple
- Understandable
- Universal
- Formalise-able / Standardise-able

The Case “against” Diagrams

- Not inherent knowledge
- Easily cluttered
- Require some training
- Not necessarily revealing
- Must be liked to be accepted and used
- Effort to draw

UML diagrams



UML Diagrams (comparative slide)

- **Use-Case** *(relation of actors to system functions)*
- **Class** *(static class structure)*
- **Object** *(same as class - only using class instances – i.e. objects)*
- **State** *(states of objects in a particular class)*
- **Sequence** *(Object message passing structure)*
- **Collaboration** *(same as sequence but also shows context - i.e. objects and their relationships)*
- **Activity** *(sequential flow of activities i.e. action states)*
- **Component** *(code structure)*
- **Deployment** *(mapping of software to hardware)*

UML Diagram Philosophy

Any UML diagram:

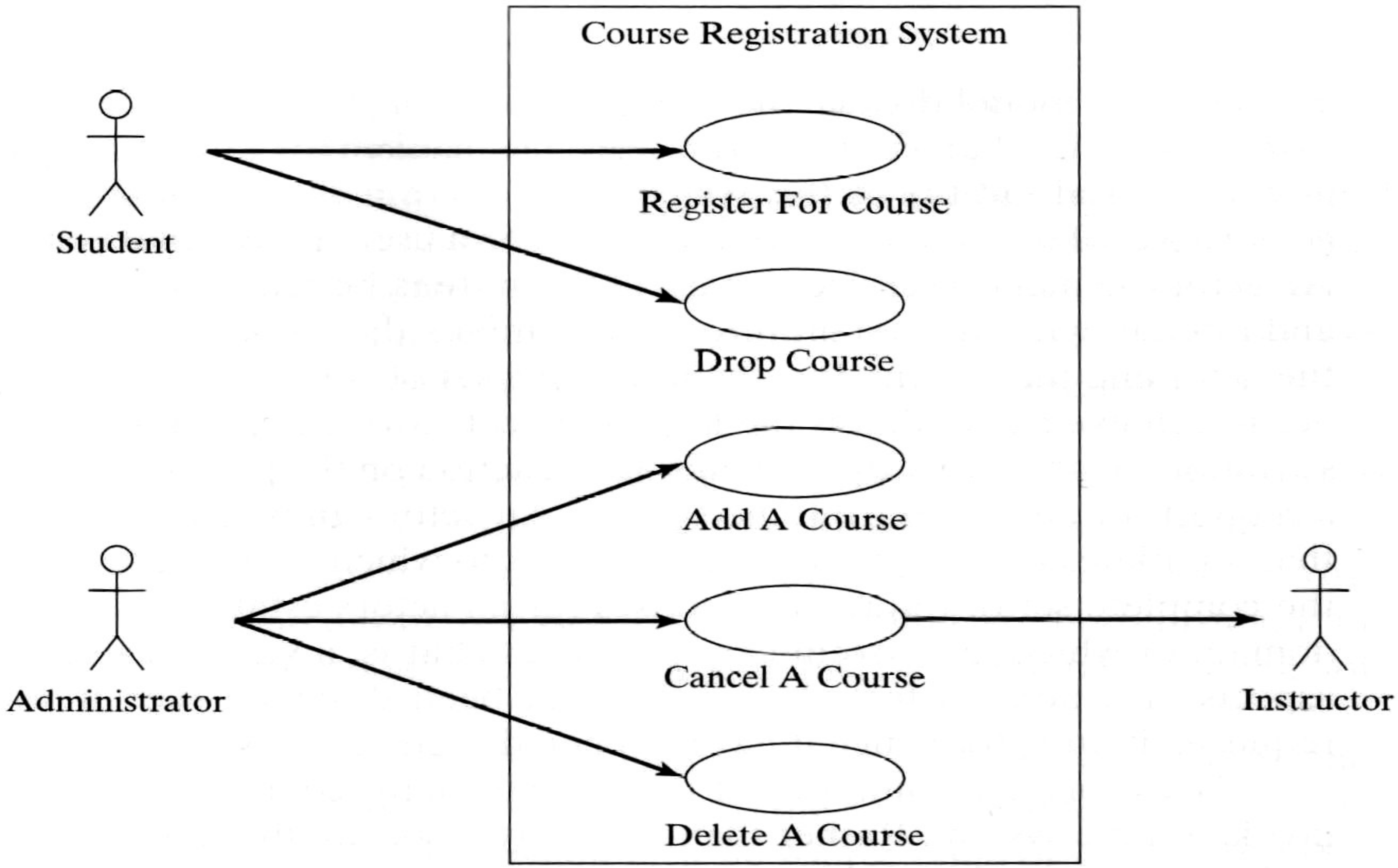
- Depicts concepts
  - as symbols
- Depicts relationships between concepts
  - as directed or undirected arcs (lines)
- Depicts names
  - as labels within or next to symbols and lines

The Main 4 UML Diagrams

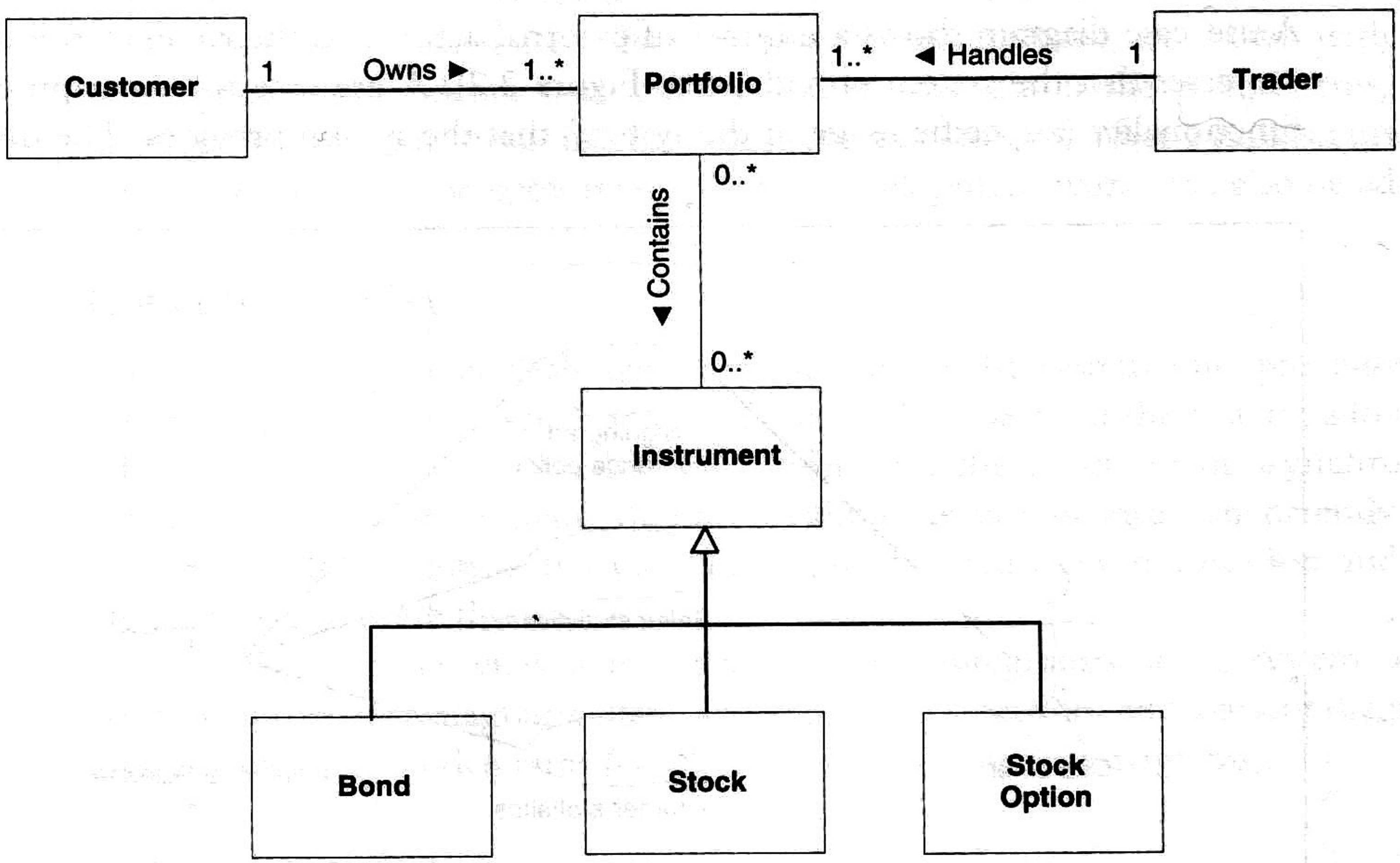
- Use-Case
- Class
- Sequence
- State

Examples are depicted on the following slides.

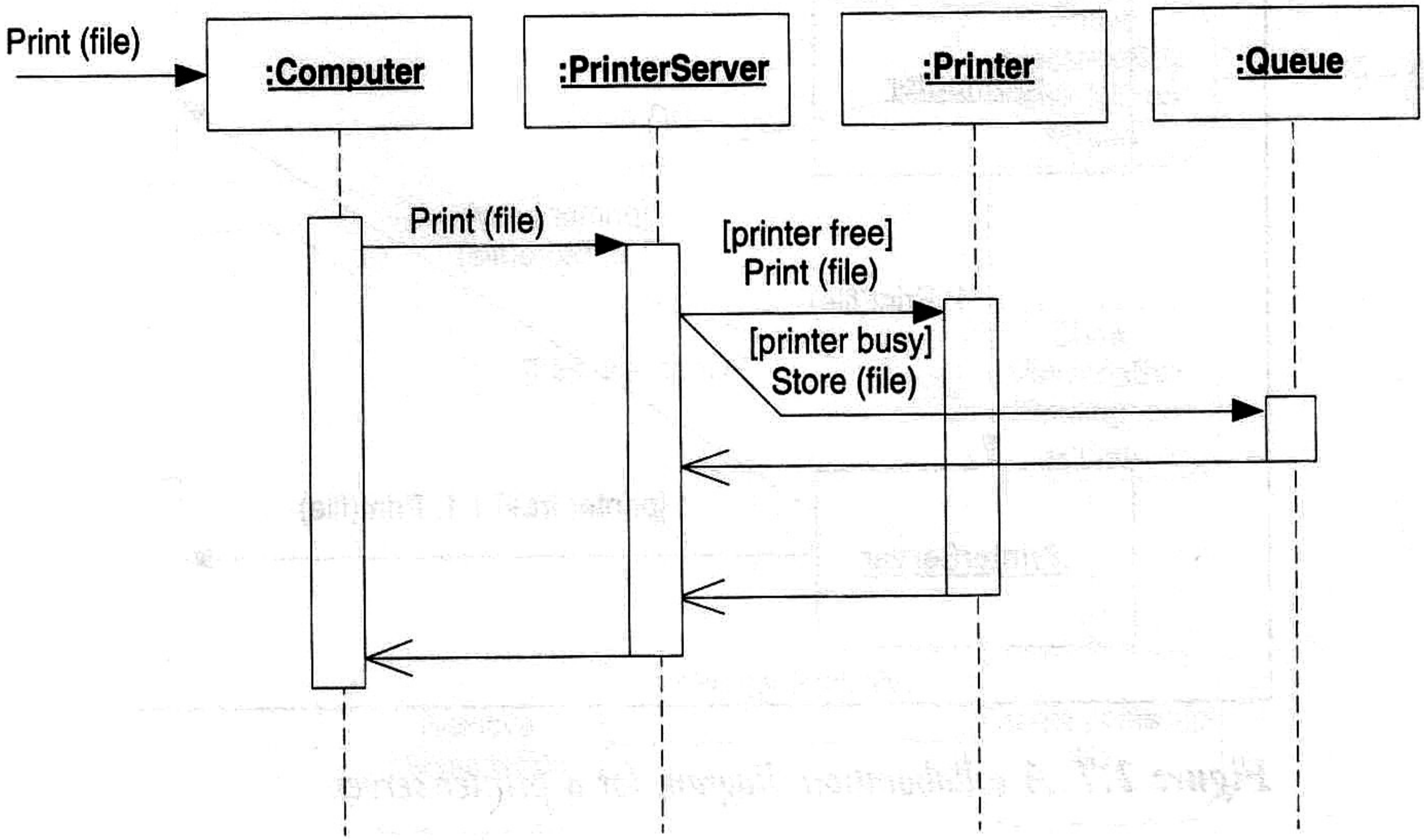
The Use-Case Diagram



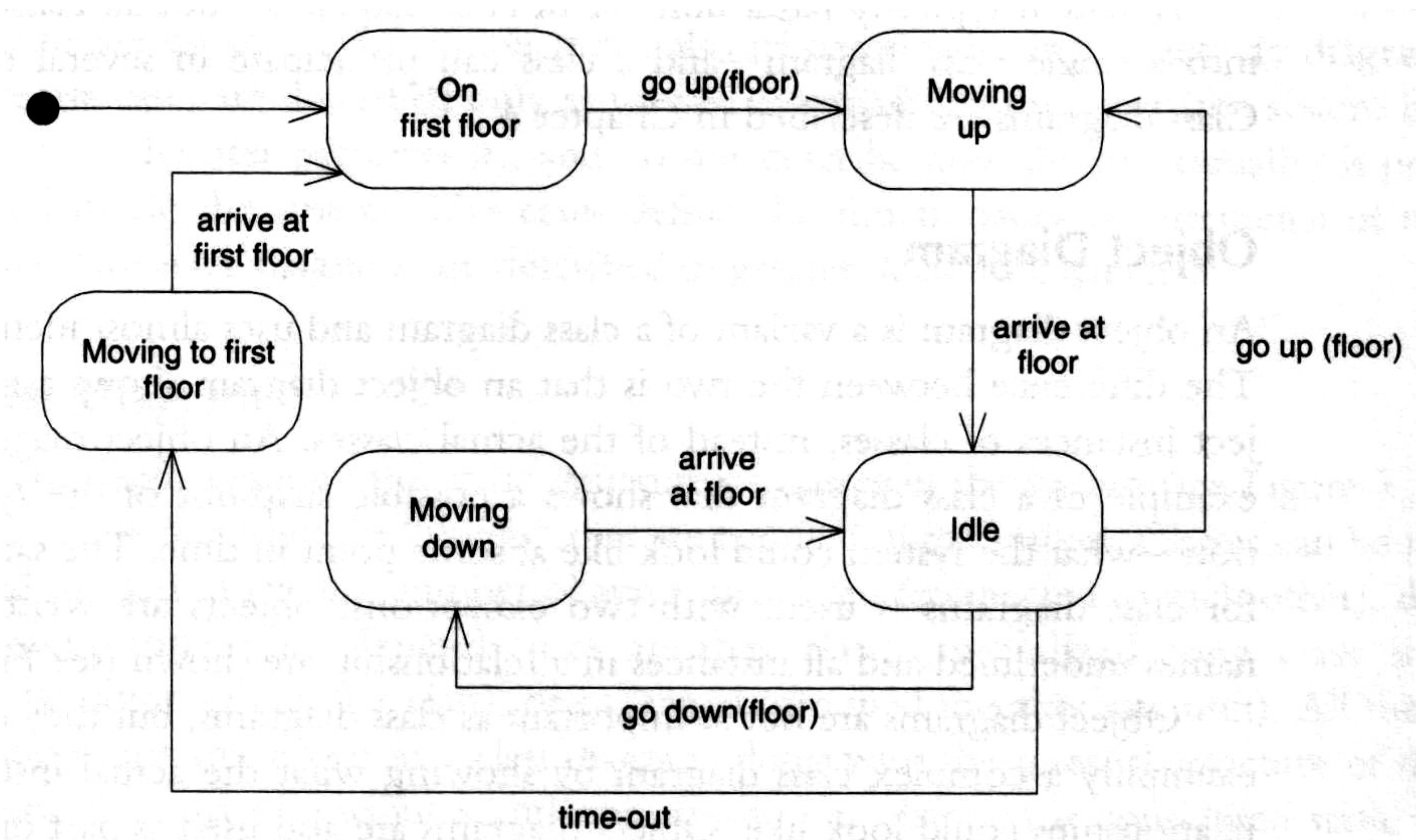
The Class Diagram



The Sequence Diagram



The State Diagram

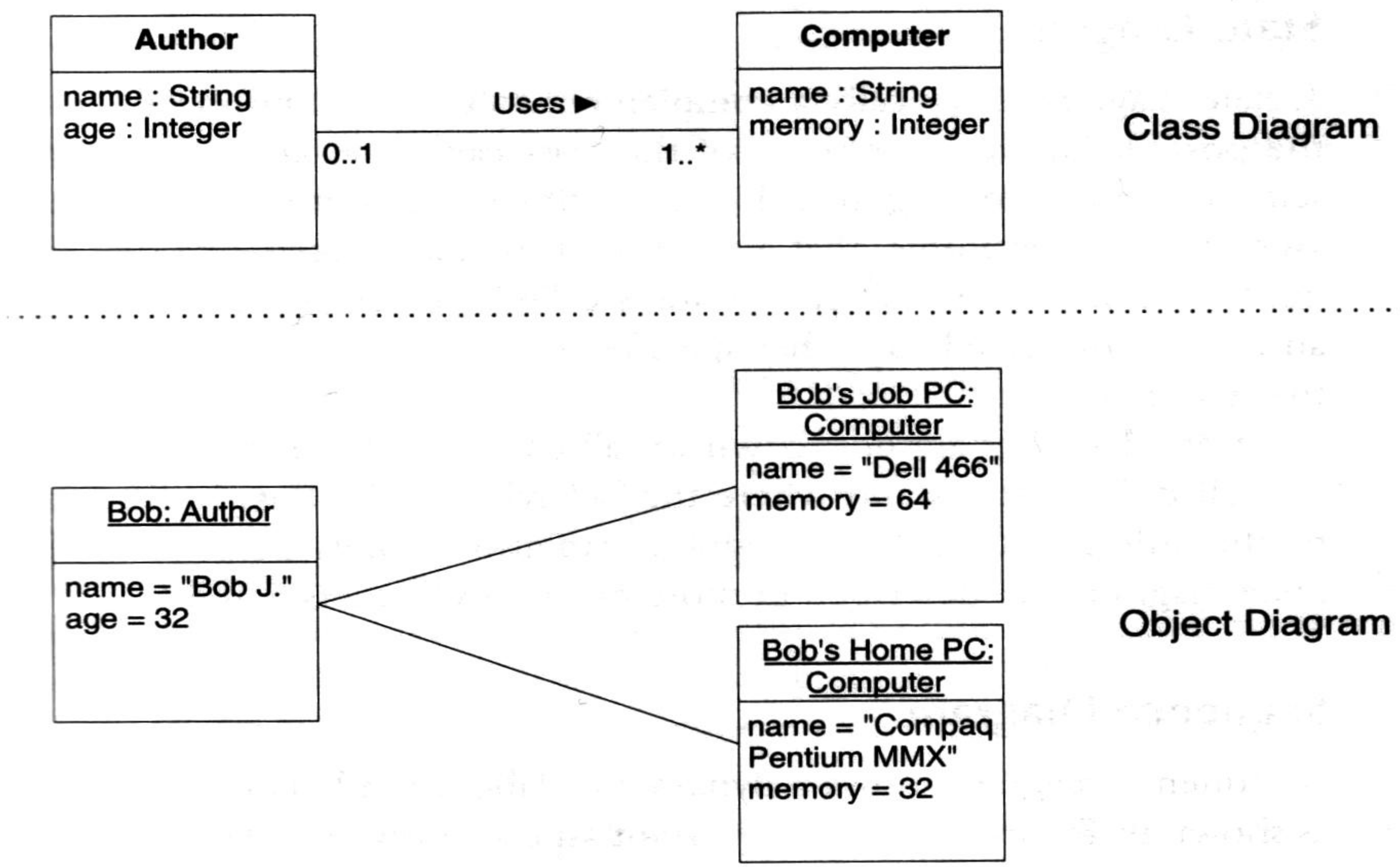


The Other 5 UML Diagrams

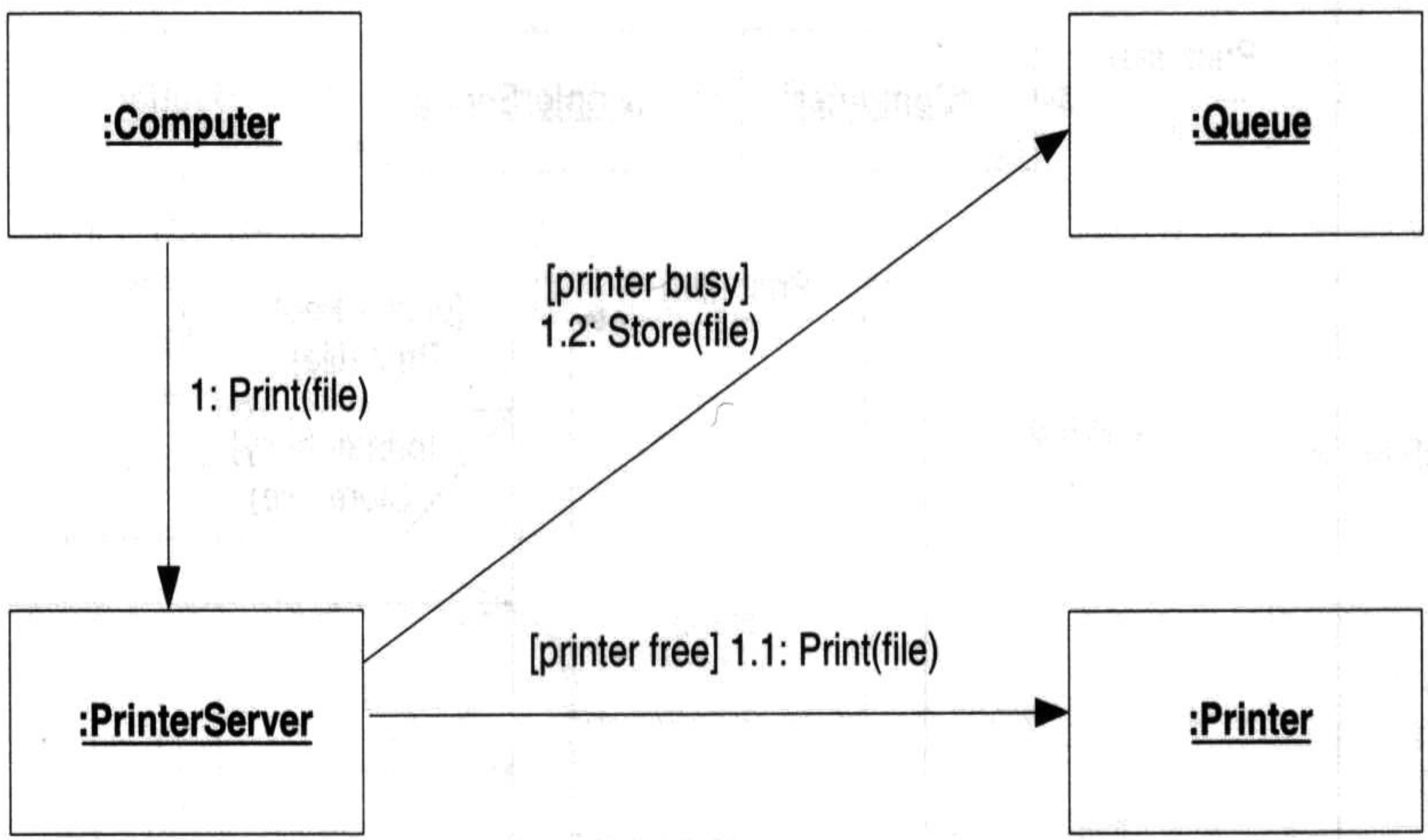
- Object
- Collaboration
- Activity
- Component
- Deployment

Examples are depicted on the following slides.

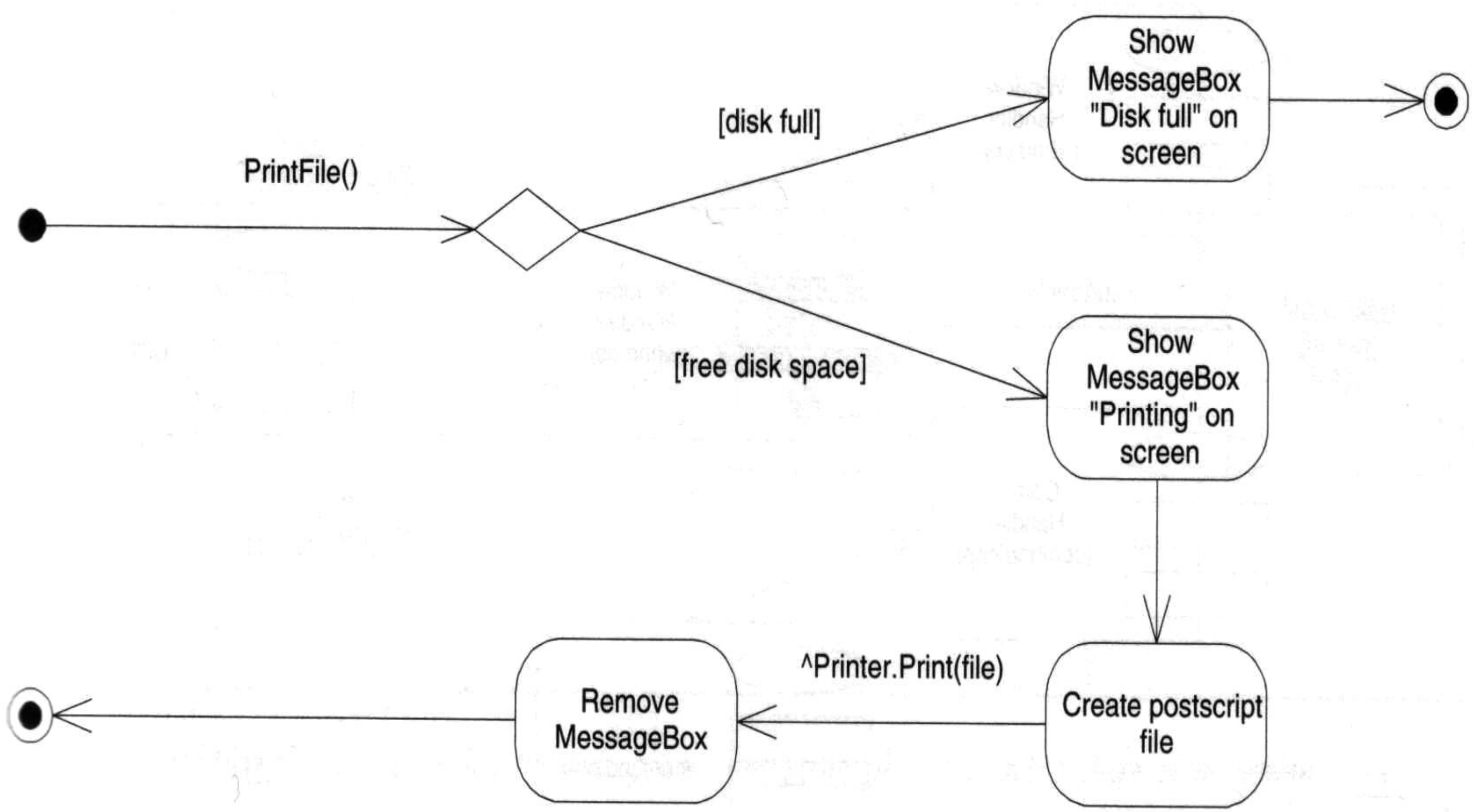
The Object Diagram



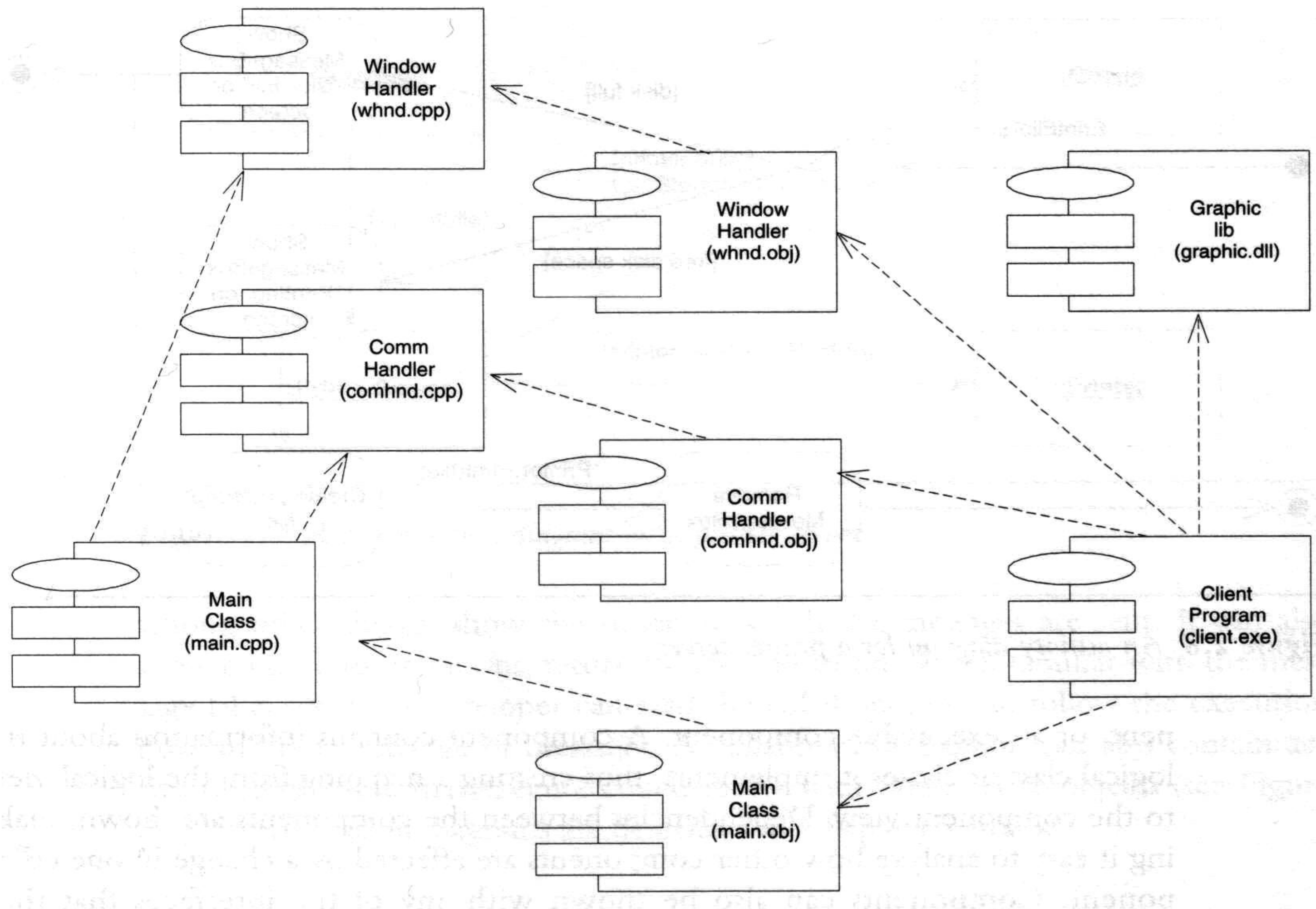
The Collaboration Diagram



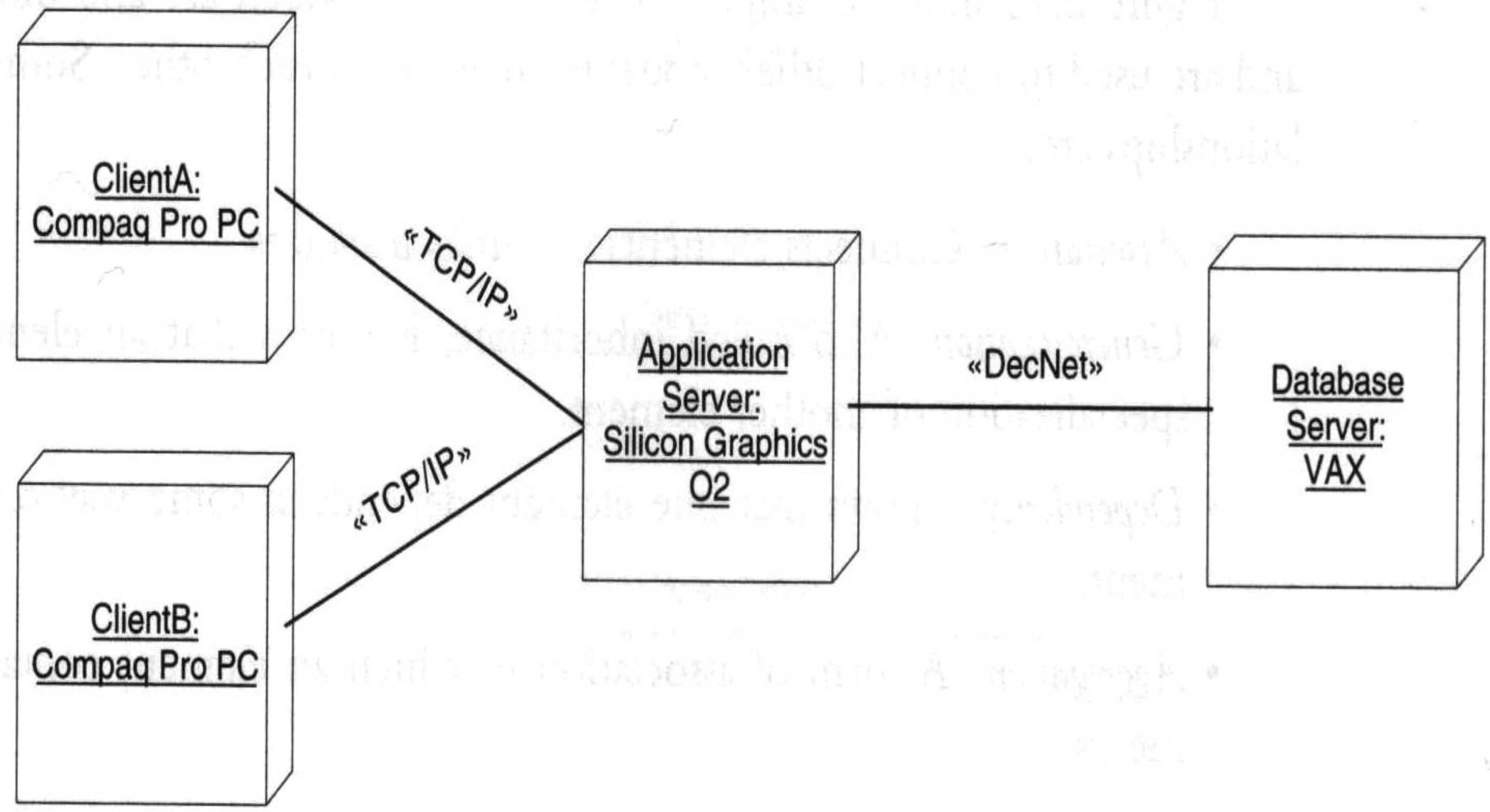
The Activity Diagram



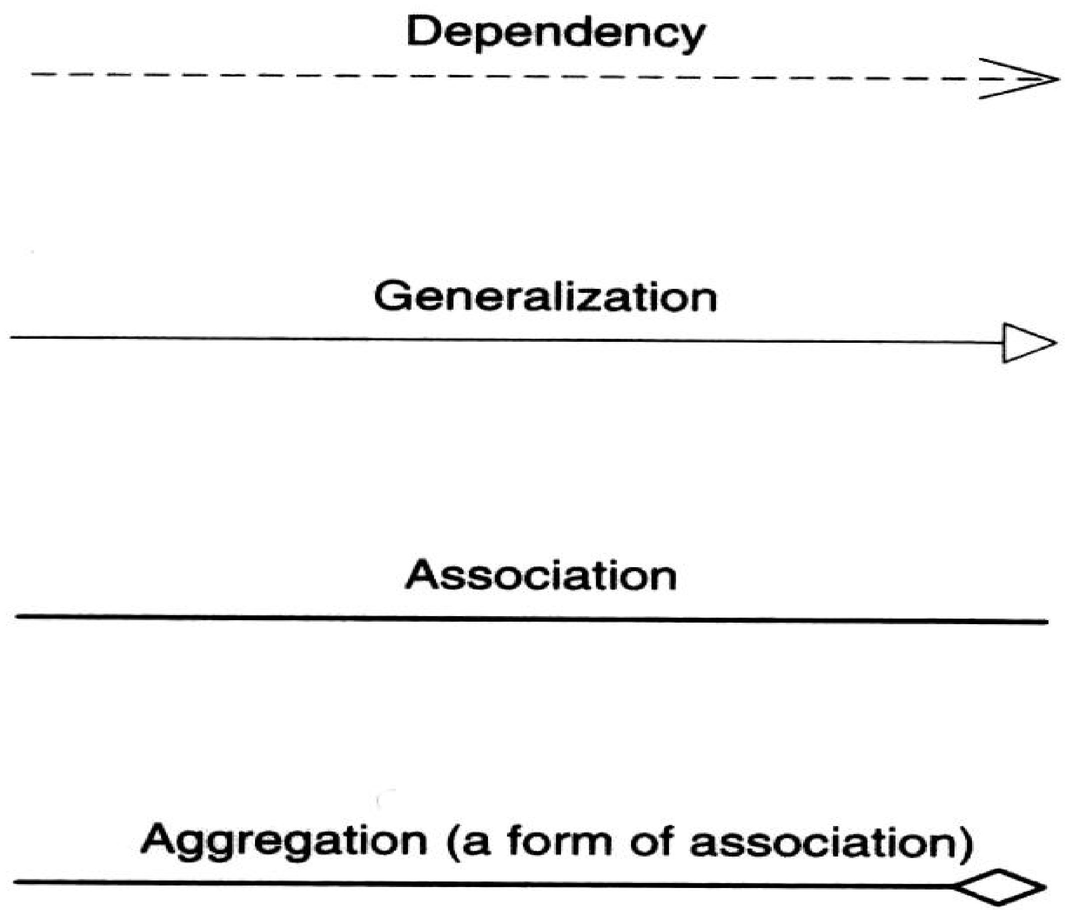
The Component Diagram



The Deployment Diagram



UML Relationships



Some Points to Ponder

1. How would you justify the use of UML in any IS project to programming personnel?
2. What is the difference between static and dynamic UML diagrams?
3. Why does UML attempt to model systems with a heavy emphasis on graphic notation?
4. Why does UML not restrict itself to one type of diagram?
5. Is UML restrictive to system development? Justify your reply.
6. A common misconception is that systems built using UML are quality guaranteed. Discuss this issue and present (write down) your reasoning.

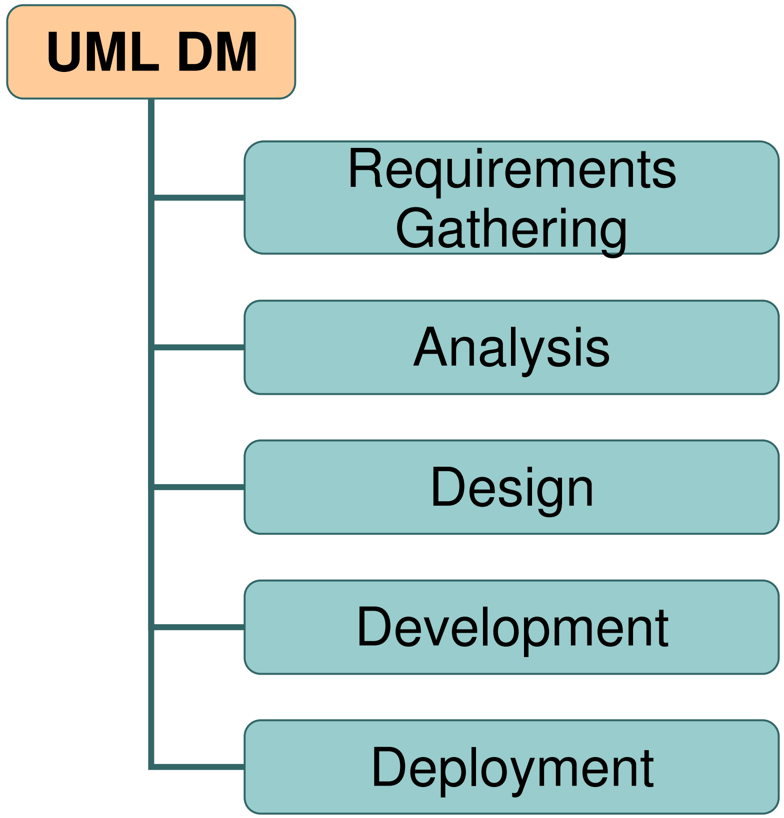
Workshop Activity -7-

Try textually describing the software controlling production of a large-scale pre-packed food company (weather/calendar/market controlled).

Once done, try describing the exact same system diagrammatically.

Draw some conclusions from your work.

UML Development Model



The next slides will outline the above phases

Requirements Gathering (1/2)

Determine what the client wants

- **Business process elicitation**
  - *Tool:* Interview; Questionnaire; Observation; Experience
  - *Product:* UML Activity diagram
- **Domain analysis**
  - *Tool:* Interview; Noun-Verb extraction; Observation
  - *Product:* UML HL Class diagram; textual supporting data

Requirements Gathering (2/2)

- **System context determination**
  - *Tool:* Observation; Process walk-through
  - *Product:* UML Deployment diagram
- **System requirements elicitation**
  - *Tool:* JAD moderated session
  - *Product:* Refined HL Class Diagram; UML Package diagram
- **Phase outcome presentation**
  - *Tool:* n/a
  - *Product:* n/a

Analysis (1/2)

Unfold a detailed understanding of the problem

- Determine system usage (Actor determination)
  - *Tool:* JAD session; User interrogation
  - *Product:* Use-Case diagram/s
- Expand Use-Cases (Fleshing-out)
  - *Tool:* Further user interrogation
  - *Product:* Textual supplement to Use-Case diagrams
- Refine Class diagram
  - *Tool:* JAD session; Observation (of JAD)
  - *Product:* Refined Class diagram (inc. associations, cardinalities/modalities, generalisations, etc.)

Analysis (2/2)

- State analysis
  - *Tool:* Thought
  - *Product:* State diagram
- Interaction analysis
  - *Tool:* Internal experience (Use-Case + refined Class + State)
  - *Product:* Sequence and Collaboration diagrams
- System integration analysis
  - *Tool:* Observation; walk-through, interrogation
  - *Product:* Detailed deployment diagram

Design (1/2)

Analysis ↔ Design (until design is completed)

- **Develop and refine object diagrams**
  - *Tool:* Operation analysis
  - *Product:* Activity diagrams
- **Develop component diagrams**
  - *Tool:* Programmers; system component (and interaction) visualisation
  - *Product:* Component diagrams
- **Start thinking about deployment**
  - *Tool:* Analysis of component structure and their integration (from previous); external system co-operation
  - *Product:* Part of deployment diagram from analysis

Design (2/2)

- **Prototype development (GUI)**
  - *Tool:* JAD session previous and new; previous use-case diagrams
  - *Product:* Screen prototypes and shots
- **Testing of design**
  - Peer-developed test cases based on existing use-case diagrams
  - *Product:* Test cases (scripts)
- **Design documentation structure**
  - *Tool:* Designer input and document configuration tools
  - *Product:* Document structure

Development (1/2)

Programmers' realm – should proceed swiftly if right effort was initially invested

- **Coding**
  - *Tool:* Programmers using class, object, activity and component diagrams
  - *Product:* Code
- **Testing**
  - *Tool:* Back-and-forth from the coding activity using the test cases designed in the design phase
  - *Product:* Test results

Development (2/2)

- **Implement, connect and test UIs**
  - *Tool:* UI development environment; programmer interaction
  - *Product:* Full working system
- **Finalise documentation**
  - *Tool:* programmer input; inspection
  - *Product:* Full system documentation

Deployment

Software to hardware mapping and consideration of interacting systems

- **Backup and recovery strategy**
  - *Tool:* Initial requirements; system nature considerations
  - *Product:* Crash recovery plan
- **Installation**
  - *Tool:* n/a
  - *Product:* Deployed system
- **Installation testing**
  - *Tool:* Implementation of test sequences including backup and recovery
  - *Product:* Final (actual) test results

Workshop Activity -8-

Internally within each team, divide up into "client/analyst", "object engineer", and "programmer/documentation specialist" and try developing a <u>basic</u> DVD rental control software system using basic UML notation so far covered and according to UML DM. Request trainer inspection after every phase. Coding need only be carried out at a highly abstract structured text or pseudo-code level.

Use-Case Diagrams (UCDs) (1/2)

- A use-case is…
  - a simplification of (a part of) a business process model
  - a ***set of activities*** within a system
  - presented from the point of view of the associated ***actors*** (i.e. those actors interacting with the system)
  - leading to an externally ***visible result***
- What is the model supposed to do?
  - offer a simplified and limited notation
  - allow other diagrams used to support (realise) it
  - combinatorial with prototypes as complementary information
  - not intended to model functional decomposition

Use-Case Diagrams (UCDs) (2/2)

Components: use-cases and actors

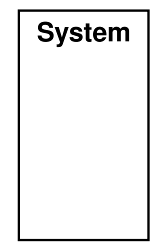
- a use-case <u>must always</u> deliver a value to an actor
- the aggregate of all use-cases is the system's complete functionality

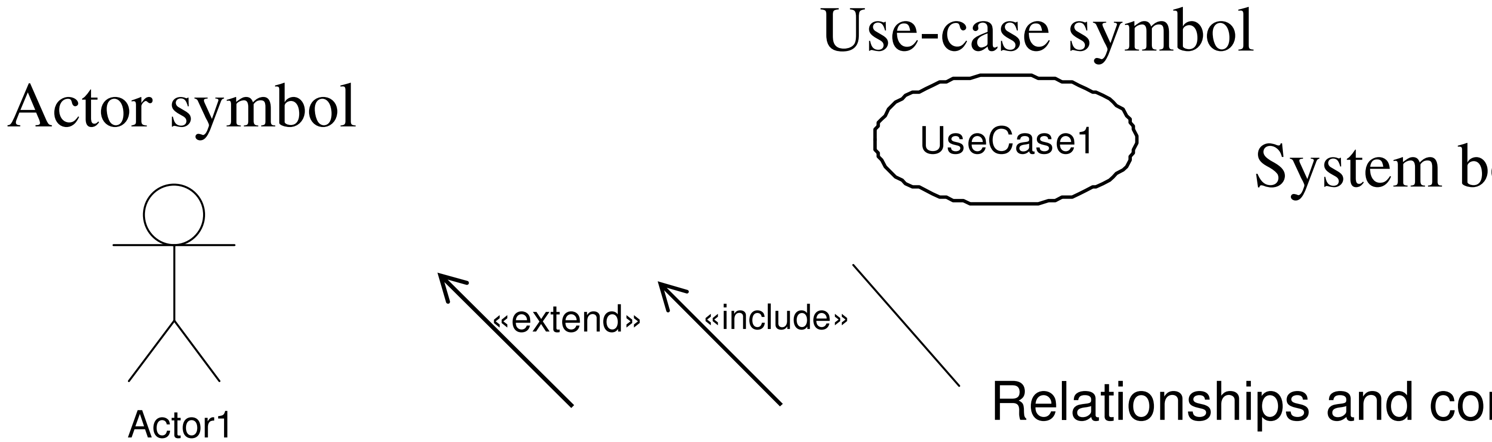
Goals:

- consolidate system functional requirements
- provide a development synchronisation point
- provide a basis for system testing
- provide a basic function-class/operation map

UCD Components

- The use case itself is drawn as an oval.
- The actors are drawn as little stick figures.
- The actors are connected to the use case with lines.





UML Actors

- An actor
  - Is a class that forms a ***system boundary***
  - participates in a use-case
  - is not within our responsibility as systems analyst/s and/or designer/s
- Examples are
  - end-users (roles)
  - external systems (co-operations)
  - time related events (events)
  - external, passive objects (entities)

UML Actor Classification

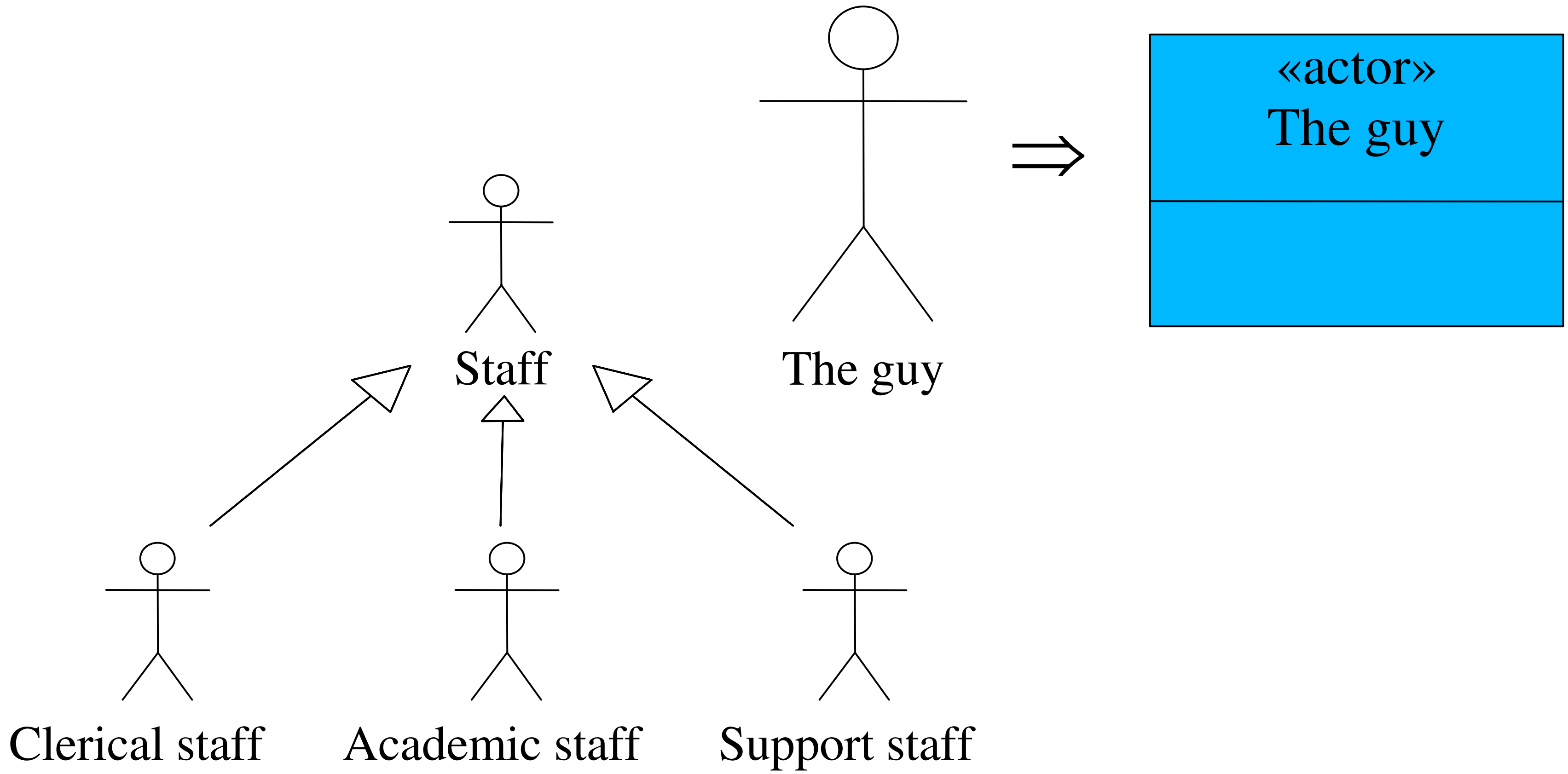
- A primary actor uses the system's primary functions (e.g. a bank cashier);
- A secondary actor uses the system's secondary functions (e.g. a bank manager, system administrator);
- An active actor initiates a use-case;
- A passive actor only participates in one or more use-cases.

Identifying UML Actors

Ask yourself the following questions:

- Who are the system’s primary users?
- Who requires system support for daily tasks?
- Who are the system’s secondary users?
- What hardware does the system handle?
- Which other (if any) systems interact with the system in question?
- Do any entities interacting with the system perform multiple roles as actors?
- Which other entities (human or otherwise) might have an interest in the system's output?

UML Actor Notation and Generalisation Examples



UML Use-Cases (UCs not UC Diagrams UCDs)

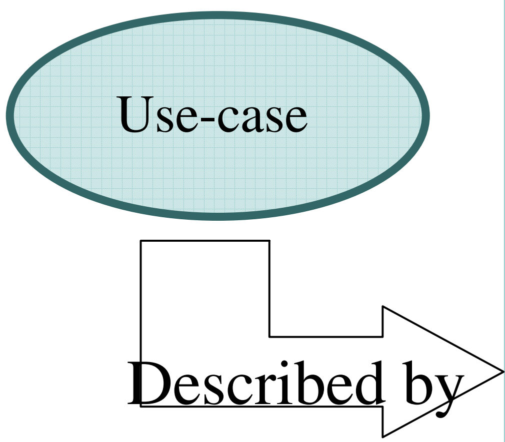
Definition: *"A set of sequences of actions a system performs that yield an observable result of value to a particular actor.“*

Use-case characteristics:

- Always initiated by an actor (voluntarily or
- involuntarily);
- Must provide discernible value to an actor;
- Must form a complete conceptual function.

(conceptual completion is when the end observable value is produced)

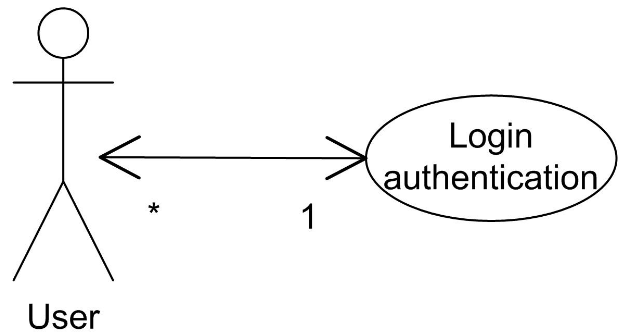
UC Description Criteria



Use-Case Number (ID) and Name

- actors
- pre- and post-conditions
- invariants
- non-functional requirements
- Behaviour modelled as:
  - activity diagram/s
  - decomposition in smaller UC diagrams
- error-handling and exceptions
- Rules modelled as:
  - activity diagram/s
- services
- examples, prototypes, etc.
- open questions and contacts
- other diagrams

UC Description Example



Example on the next slide

Example on the slide after the next

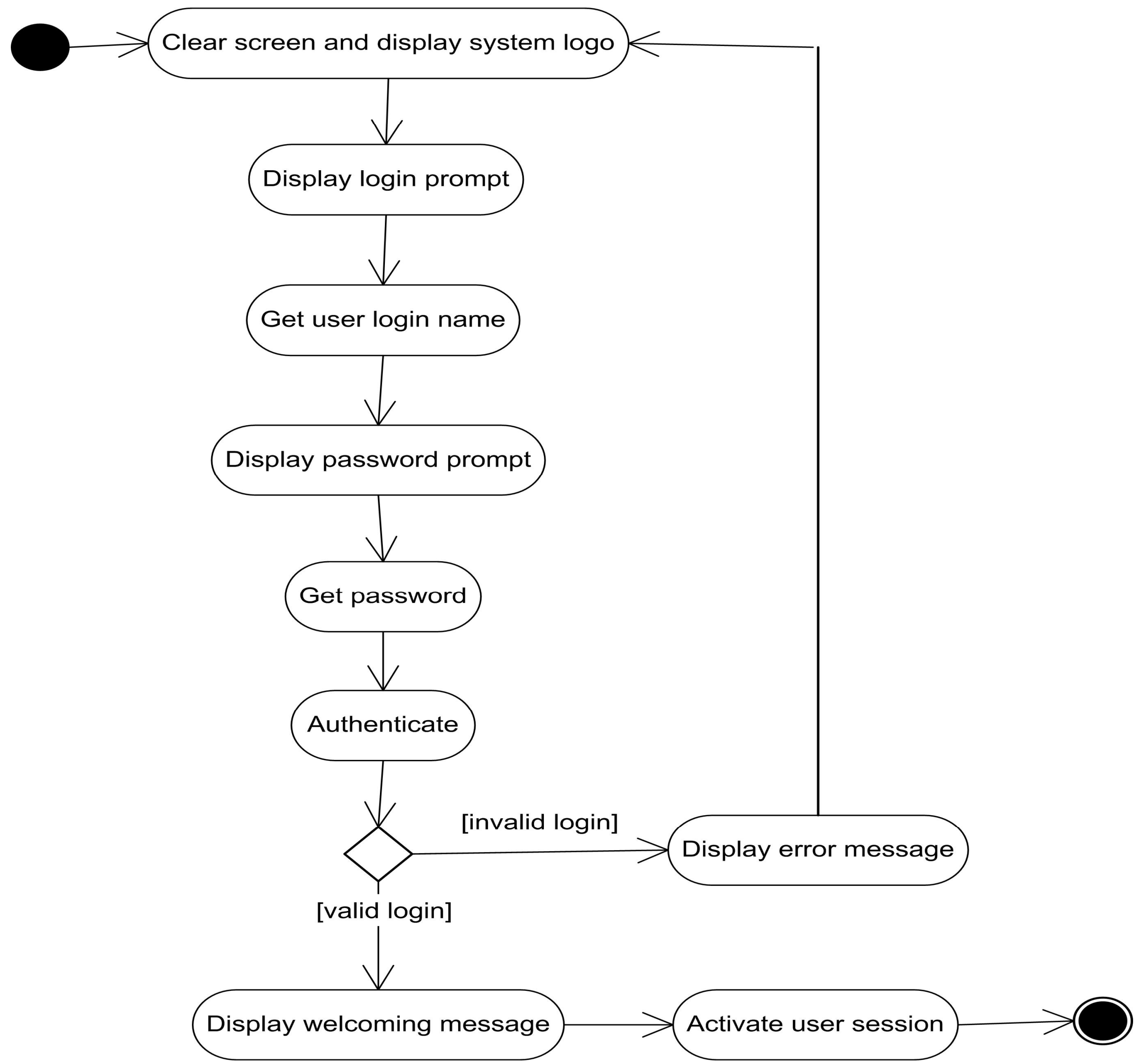
Example two slides further on

E.g. Collaboration diagram (tackled later on)

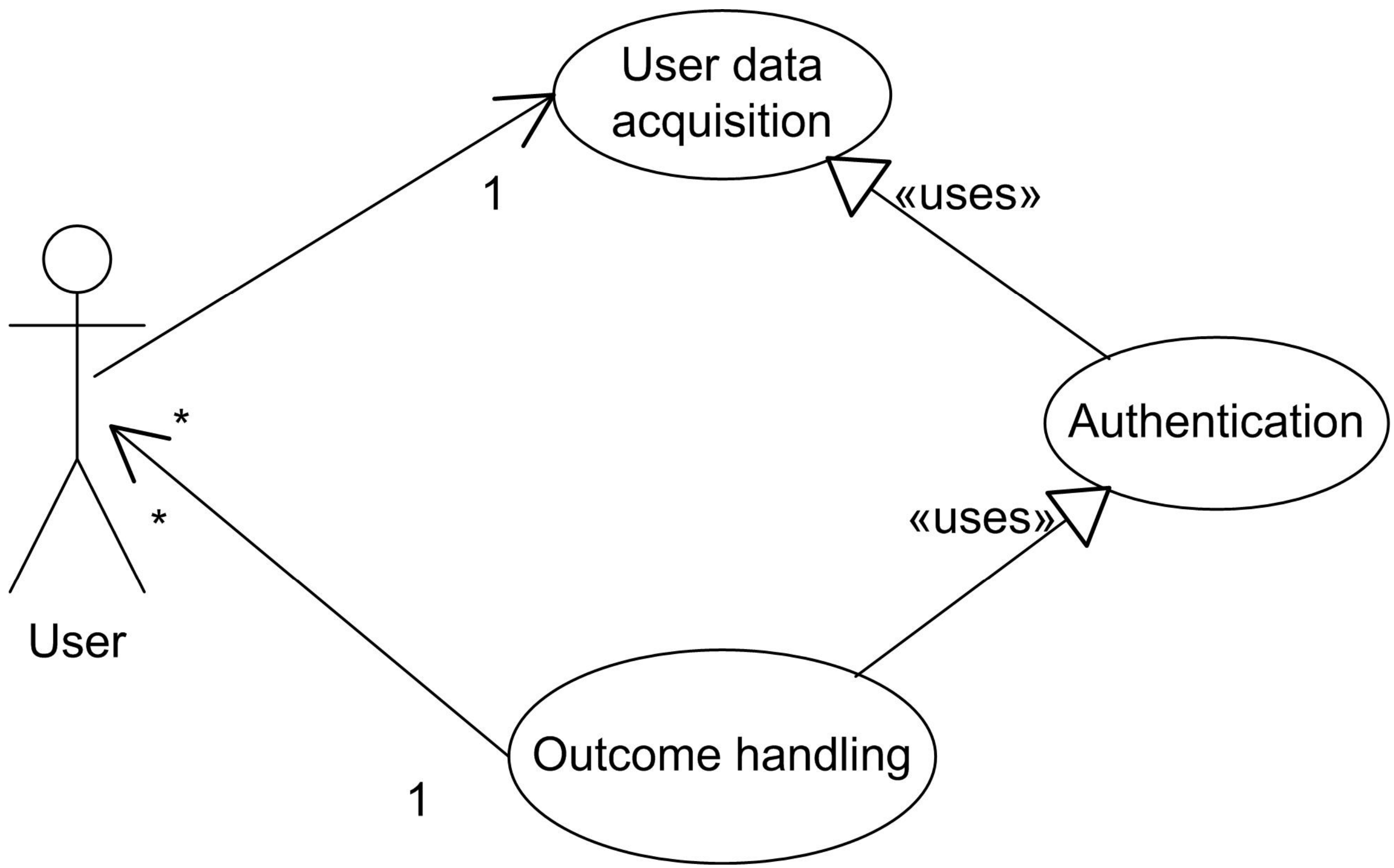
UC: Login authentication

- User
- Disable access - Enable access
- Logged in user = valid user
- Login delay; line security
- Behaviour modelled as:
  - activity diagram/s
  - decomposition in smaller UC diagrams
- Invalid login name; interrupt entry
- Rules modelled as:
  - activity diagram/s
- Log, pass prompts; authenticate
- examples, prototypes, etc.
- open questions and contacts
- other diagrams (realisations)

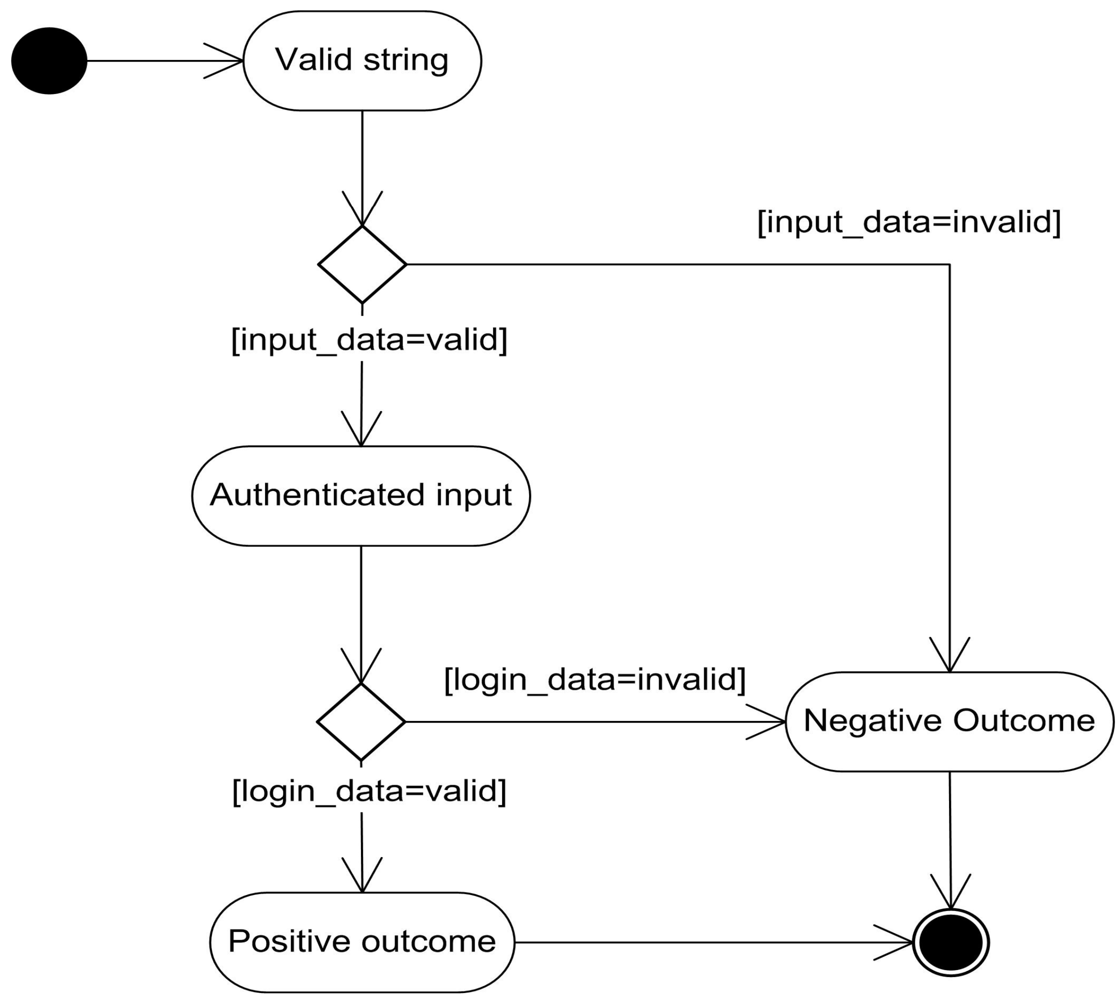
Activity Diagram from previous



Sub-UCs to Login Example



Rules Activity Diagram Example



Consolidating UC Descriptions

Ask yourself these questions:

- Do all actors interacting with a given UC have communication association to it?
- Are there common roles amongst actors?
- Are there UC similarities?
- Are there special cases of a UC?
- Are all system functions catered for by UCs?

UCD Relationships (1/2)

- Association relationship
- Extend relationship

  «extend»
- Include relationship

  «include»
- Generalisation relationship

UCD Relationships (2/2)

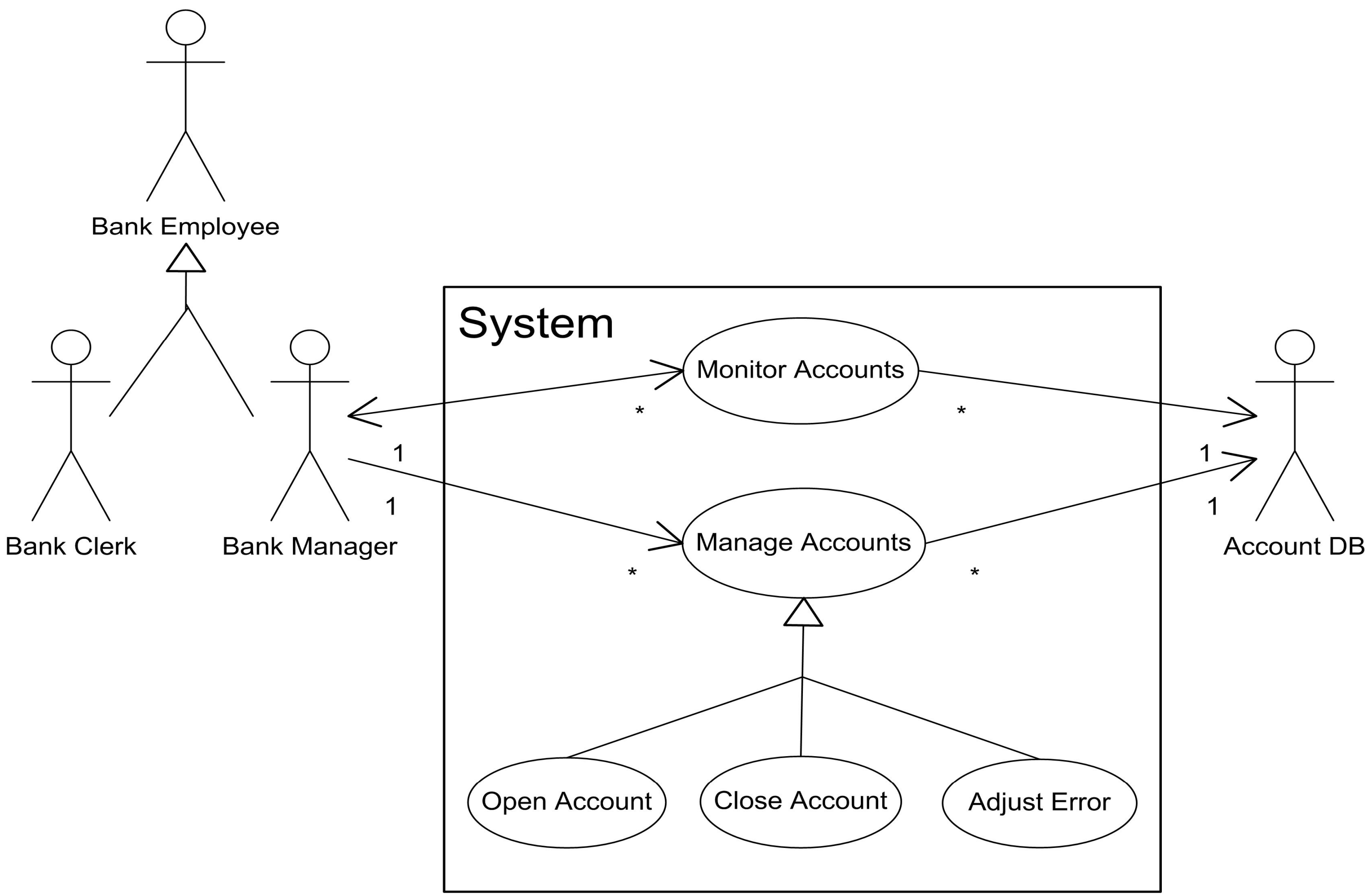
- Associations
  - Links actors to their UCs
- Use (or include)
  - Drawn from base UC to used UC, it shows inclusion of functionality of one UC in another (used in base)
- Extend
  - Drawn from extension to base UC, it extends the meaning of UC to include optional behaviour
- Generalisation
  - Drawn from specialised UC to base UC, it shows the link of a specialised UC to a more generalised one

UCD Definition Summary

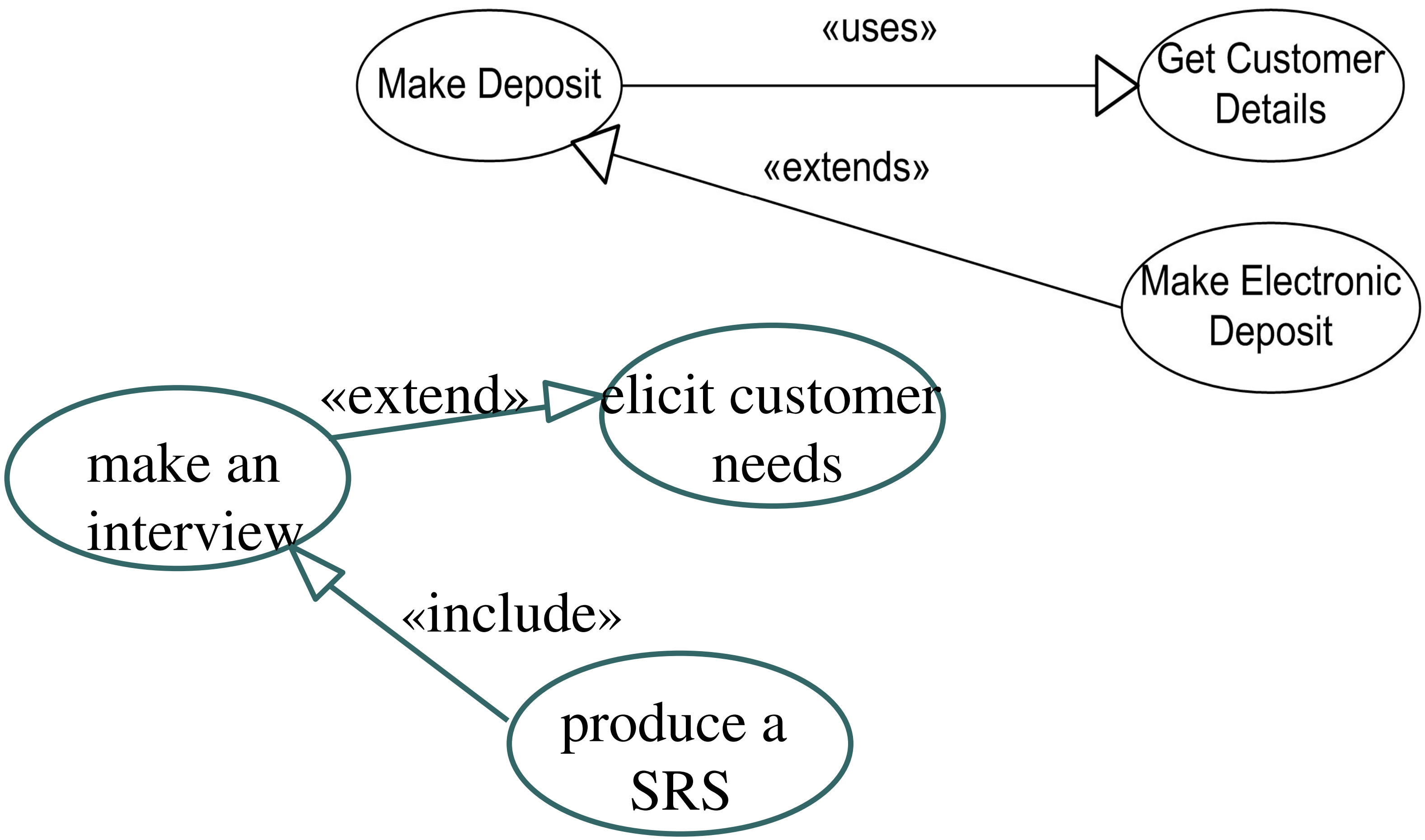
Use-Case diagrams:

- show use-cases and actors
- connected by “associations”
- refined by inheritance stereotypes
  - “uses”
    - re-use of a set of activities (use-cases)
    - partitioning of activities
    - points to the re-used use-case
  - “extends”
    - variation of a use-case
    - points to the standard use-case

UCD Relationship Example
(1/2)



UCD Relationship Example
(2/2)



What a UCD is - and what it isn't

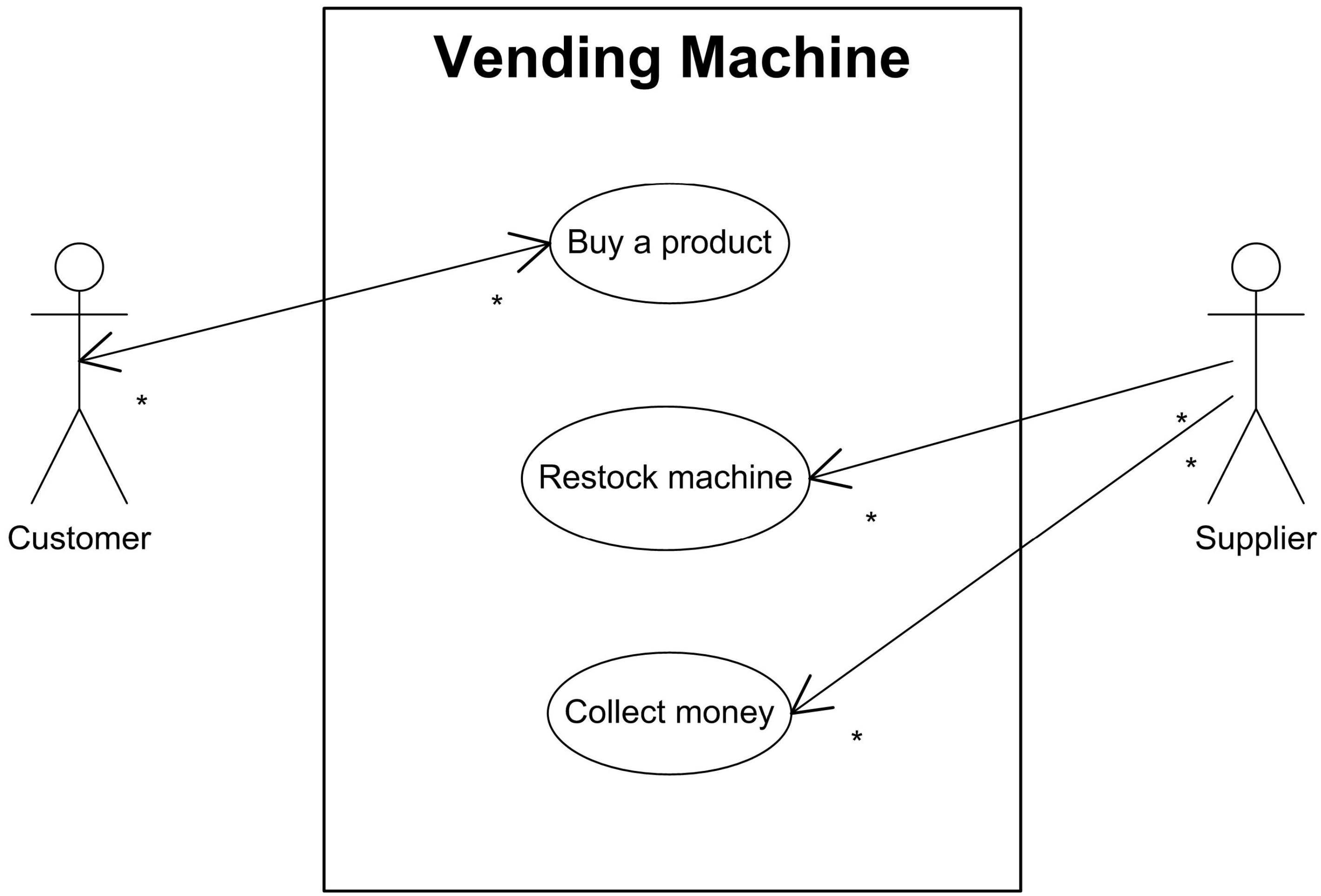
- Attention focuser on the part of the business process that is going to be supported by the IS.
- It is the end-user perspective model.
- It is goal driven
- Helps to identify system services.
- Are not used as DFDs.
- Sequences, branching, loops, rules, etc. cannot (and should not) be directly expressed.
- Are often combined with activity diagrams, which serve as their refinement.

UCD Case Study (1/3)

Vending Machine

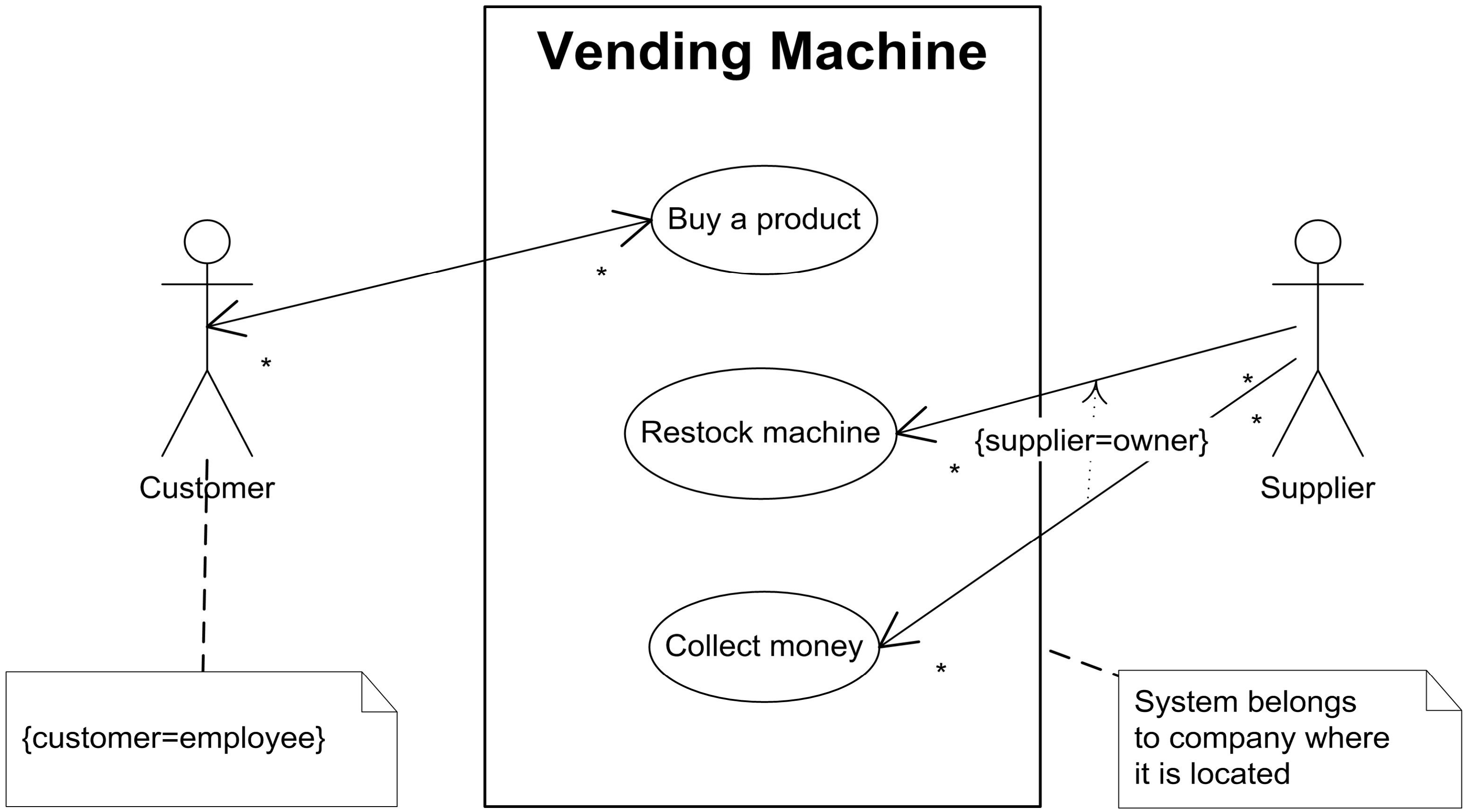
- After client interview the following system scenarios were identified:
    - A customer buys a product
    - The supplier restocks the machine
    - The supplier collects money from the machine
- On the basis of these scenarios, the following three actors can be identified:
    - Customer; Supplier; Collector

UCD Case Study (2/3)



UCD Case Study (3/3)

- Introducing annotations (notes) and constraints.



Testing UCs

- Verification
  - Confirmation of correct development according to system requirements.
- Validation *(only when working parts become available)*
  - Confirmation of correct system functionality according to end-user needs.
- Walking the UC
  - This is basically, interchangeable role play by the system developers.

Workshop Activity -9-

Create a simple UCD (i.e. no “uses” or “extends” relationships) for a course registration system described as follows:

“The course registration system should allow students to register for and drop courses. The system’s administrator should be able to add and delete courses from the system as well as to cancel planned courses. If a planned course is cancelled the relevant instructor should be notified through the system.”

(Loosely adapted from Lee, 2002)

Workshop Activity -10-

Create a UCD showing UC relationships (i.e. with "uses" or "extends" relationships and any actor generalisations) for an automated medical appointment system described as follows:

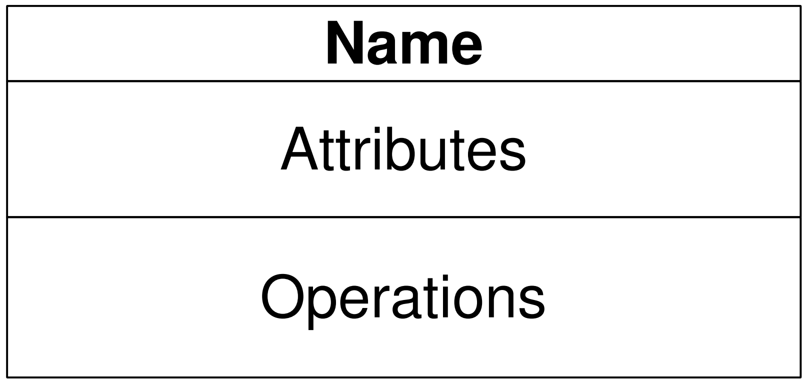
"The appointment system should allow new or existing patients to make medical appointments according to doctor-controlled availability schedules. Medical Centre management should be able to view current schedule information."

(Loosely adapted from Dennis, 2002)

The UML Class Diagram

- Is a static diagram (describes system structure)
  - Combines a number of model elements:
    - Classes
    - Attributes
    - Operations (methods)
    - Associations
    - Aggregations
    - Compositions
    - Generalisations

A UML Class



Properties of class diagrams:

- Static model;
- Models structure *and* behaviour;
- Used as a basis for other diagrams;
- Easily converted to an object diagram.

Determining Classes (1/2)

- Is there data that requires storage, transformation or analysis?
- Are there external systems interacting with the one in question?
- Are any class libraries or components being use (from manufacturers, other colleagues or past projects)?
- Does the system handle any devices?
- Does the system model organisational structures?
- Analyse all actor roles.

Determining Classes (2/2)

- **Textual Analysis** *(based on Dennis, 2002)*
  - ***A common or improper noun*** implies a class
  - ***A proper noun or direct reference*** implies an object (instance of a class)
  - ***A collective noun*** implies a class made up of groups of objects from another class
  - ***An adjective*** implies an attribute
  - ***A "doing" verb*** implies an operation
  - ***A "being" verb*** implies a classification relationship between an object and its class
  - ***A "having" verb*** implies an aggregation or association relationship
  - ***A transitive verb*** implies an operation
  - ***An intransitive verb*** implies an exception
  - ***A predicate or descriptive verb phrase*** implies an operation
  - ***An adverb*** implies an attribute of a relationship or an operation

UML Class Attributes (1/2)

- Very system dependent
- Describe characteristics of objects belonging to that class
- Can be informative - or confusing
- Has a definite type
  - Primitive (Boolean, integer, real, enumerated, etc.)
  - language specific
  - other classes
  - any user defined type
- Has different visibility, including:
  - public (viewed and used from other classes)
  - private (cannot be accessed from other classes)

UML Class Attributes (2/2)

- Can be given a default value
- Can be given class-scope
- Can list possible values of enumeration
- Directly implementable into most modern programming languages with object-oriented support *(e.g. Java)*

Attribute syntax:

`Visibility name:type=init_value{property_string}`

UML Class Attribute Examples

UNIXaccount

+ username : string
+ groupname : string
+ filesystem_size : integer
+ creation_date : date
- password : string

UNIXaccount

+ username : string
+ groupname : string = “staff"
+ filesystem_size : integer
+ creation_date : date
- password : string

Invoice

+ amount : real
+ date : date = current date
+ customer : string
+ specification : string
- administrator : string = "unspecified"
- <u>number_of_invoices : integer</u>

Invoice

+ amount : real
+ date : date = current date
+ customer : string
+ specification : string
- administrator : string = "unspecified"
- <u>number_of_invoices : integer</u>
+ status : status = unpaid { unpaid, paid }

UML Class-to-Java Example

```
Public class UNIXaccount
{
  public string username;
  public string groupname = "csai";
  public int filesystem_size;
  public date creation_date;
  private string password;
  static private integer no_of_accounts = 0
  public UNIXaccount()
  {
    //Other initialisation
    no_of_accounts++;
  }
  //Methods go here
};
```

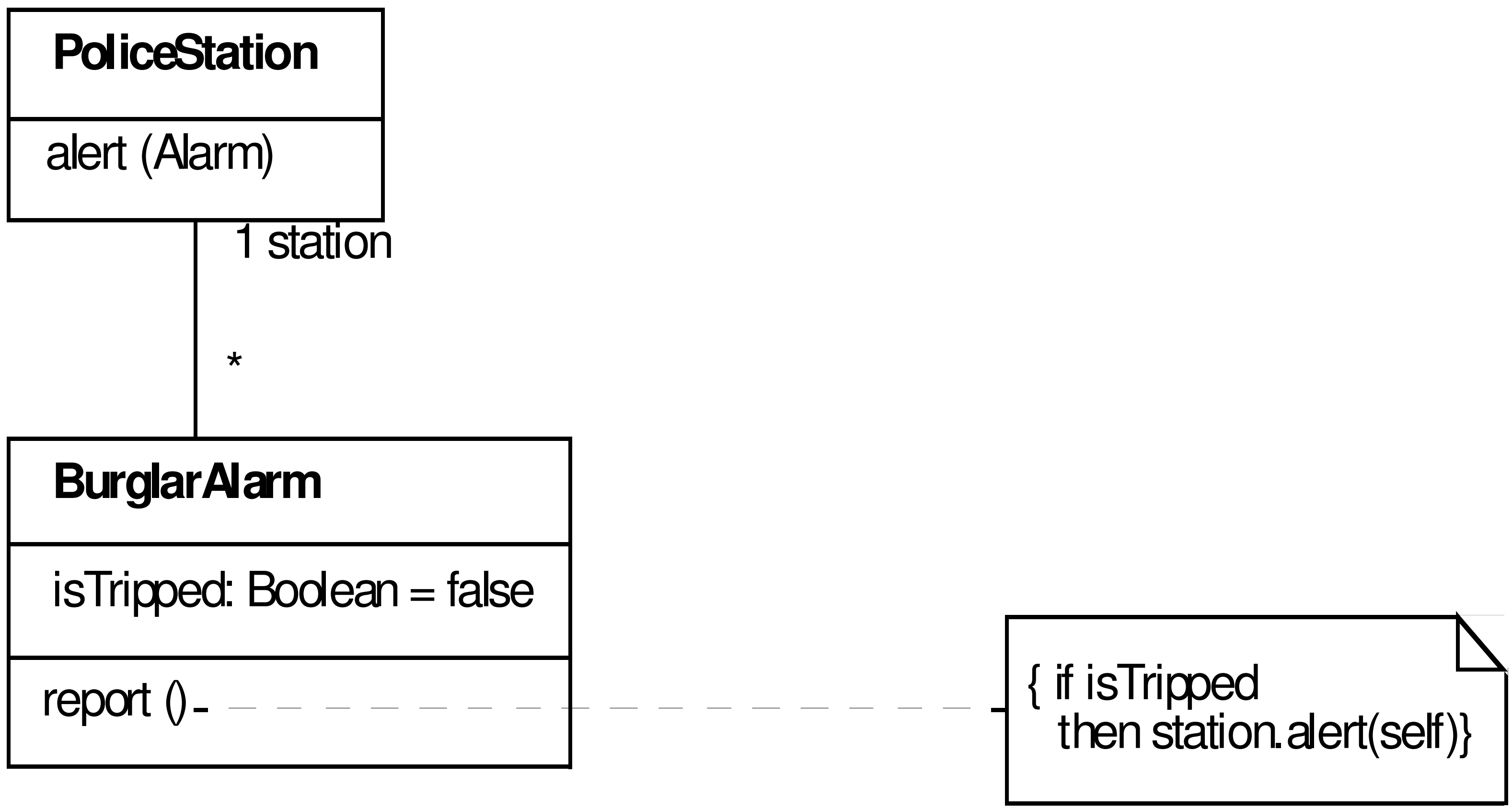
UNIXaccount
+ username : string + groupname : string = “staff" + filesystem_size : integer + creation_date : date - password : string - no_of_accounts : integer = 0

Operations (Methods)

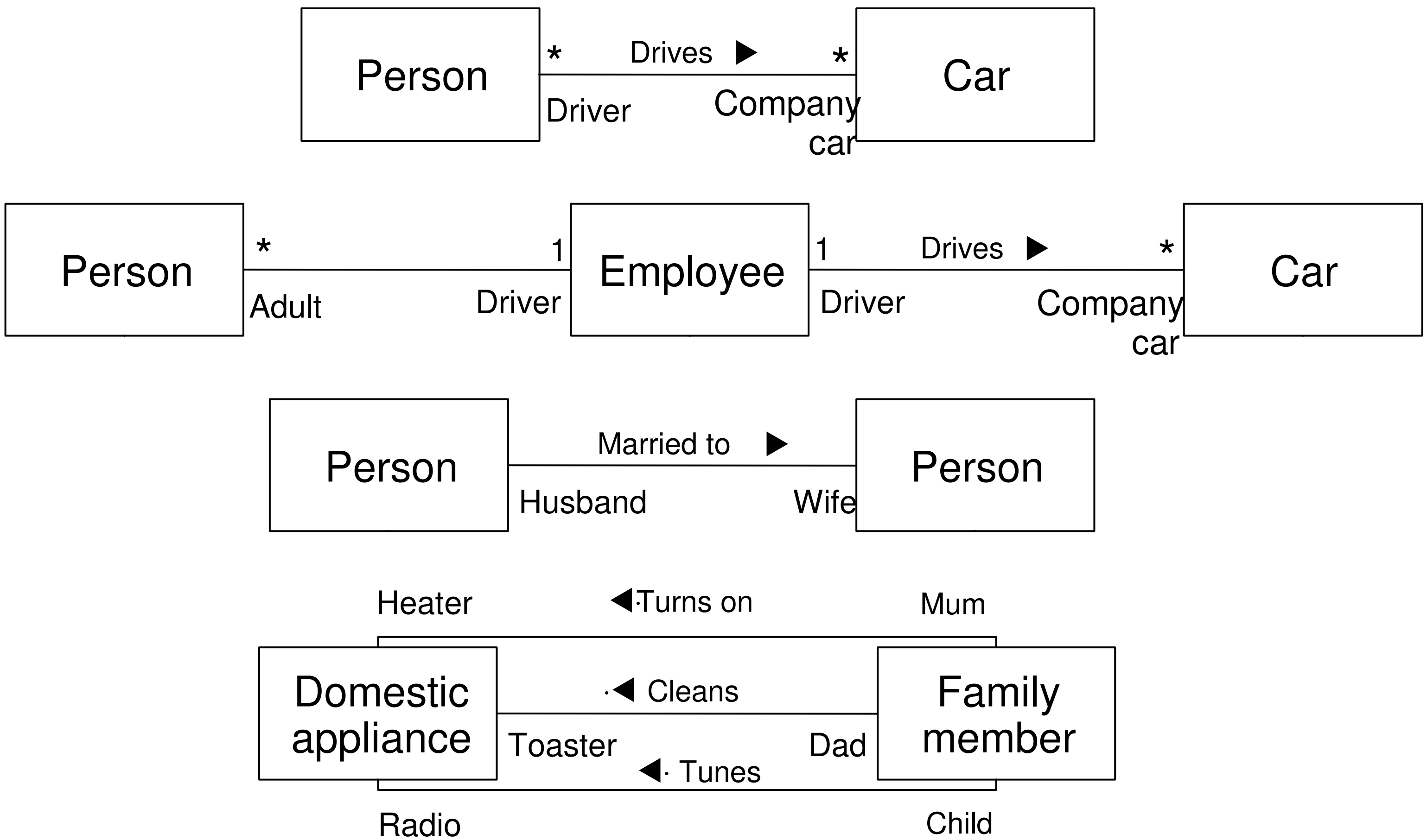
```
Public class Figure
{
  private int x = 0;
  private int y = 0;
  public void draw()
  {
    //Java code for drawing figure
  }
};
Figure fig1 = new Figure();
Figure fig2 = new Figure();
fig1.draw();
fig2.draw();
```

Figure
- x : integer = 0 - y : integer = 0
+ draw()

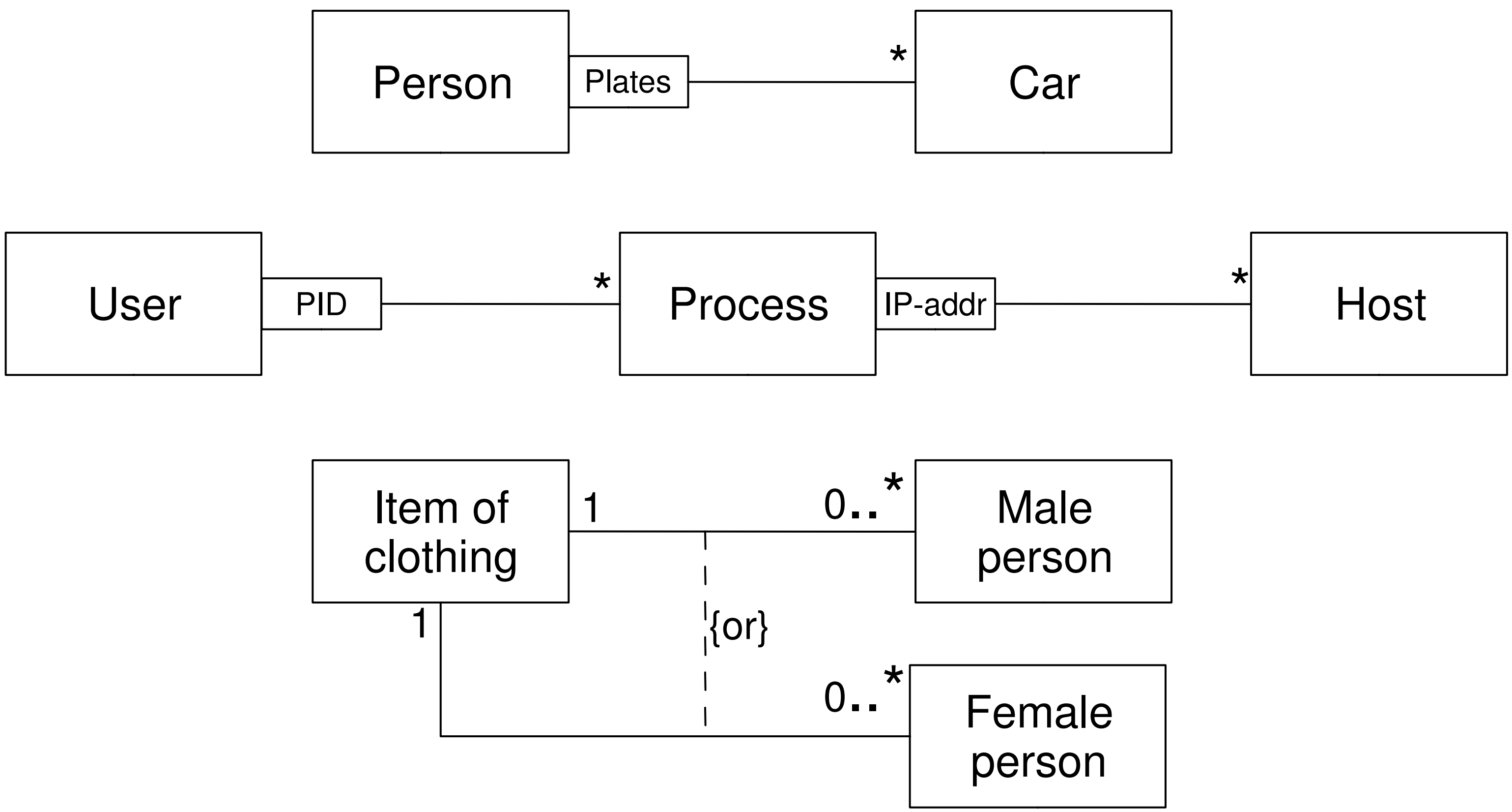
Constraints on Operations



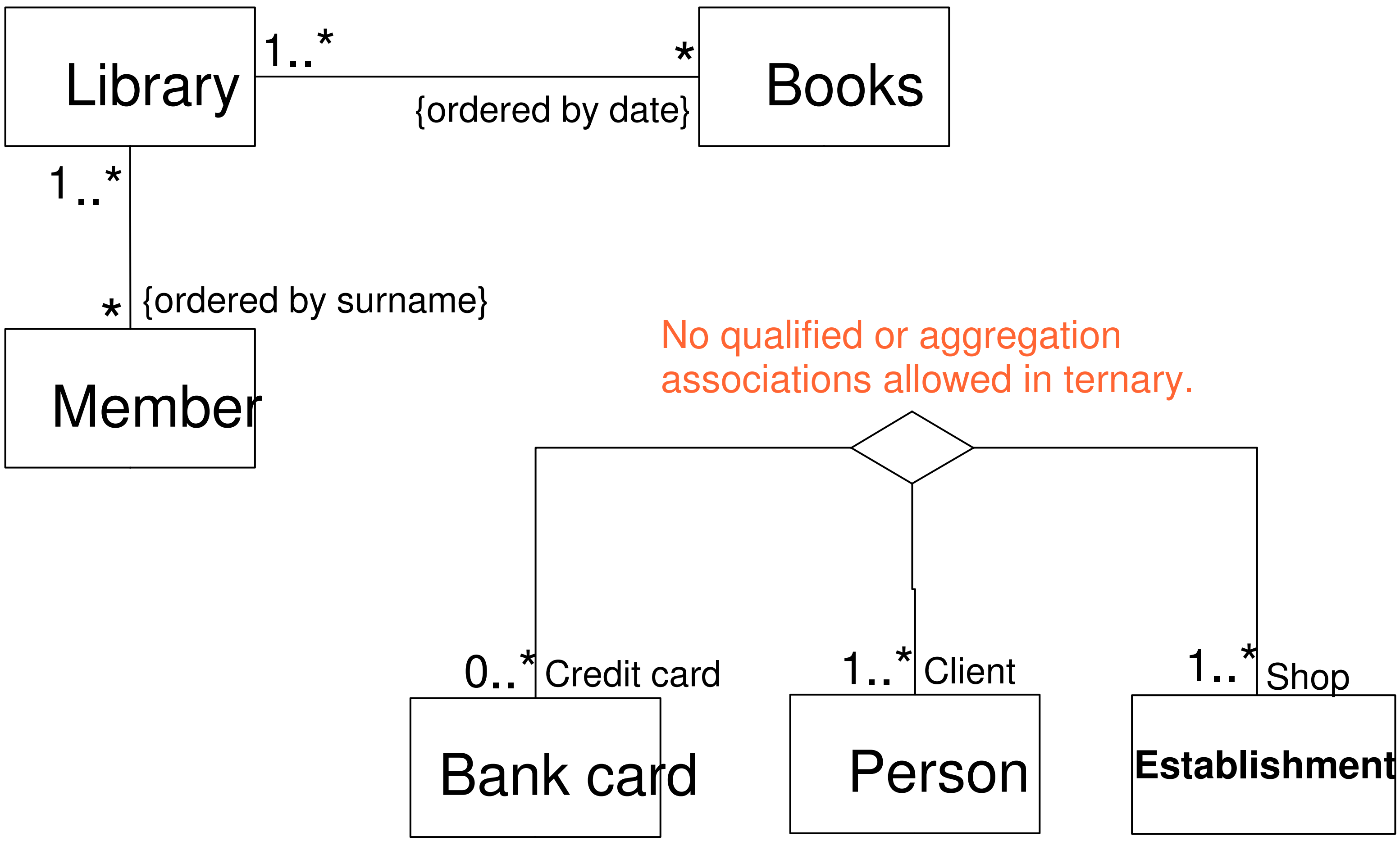
Association Examples



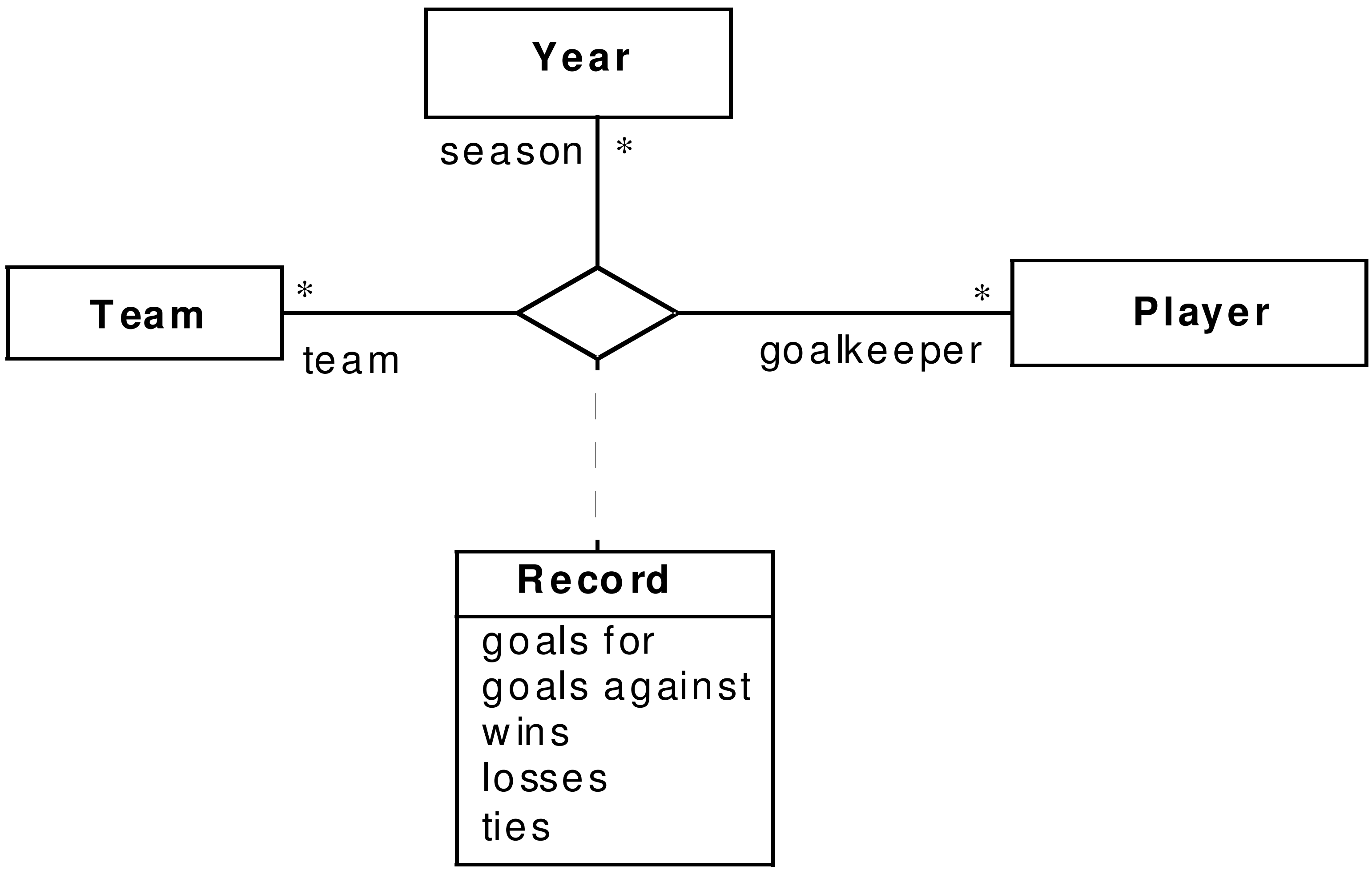
Qualified and "Or" Associations



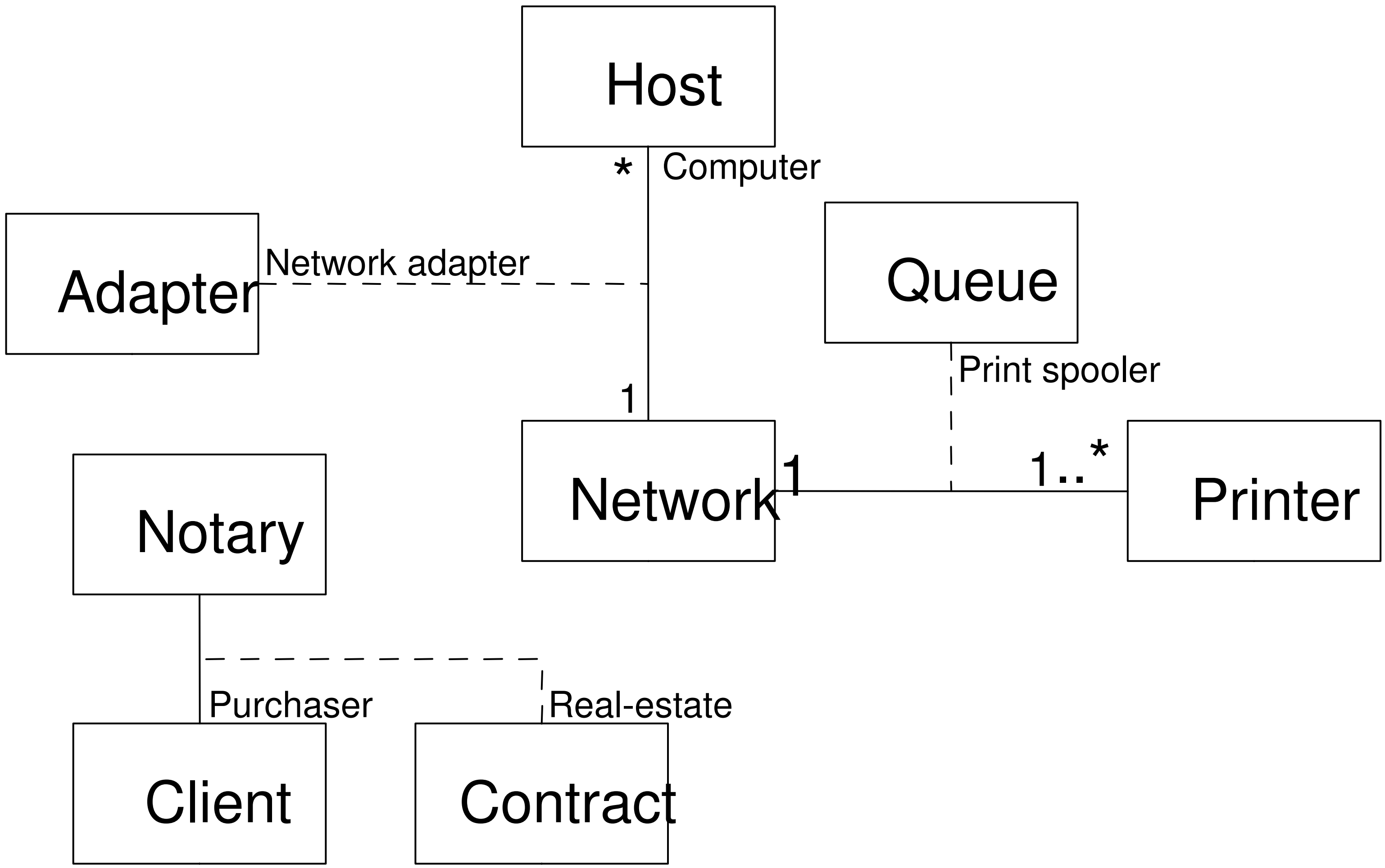
Ordered and Ternary Associations



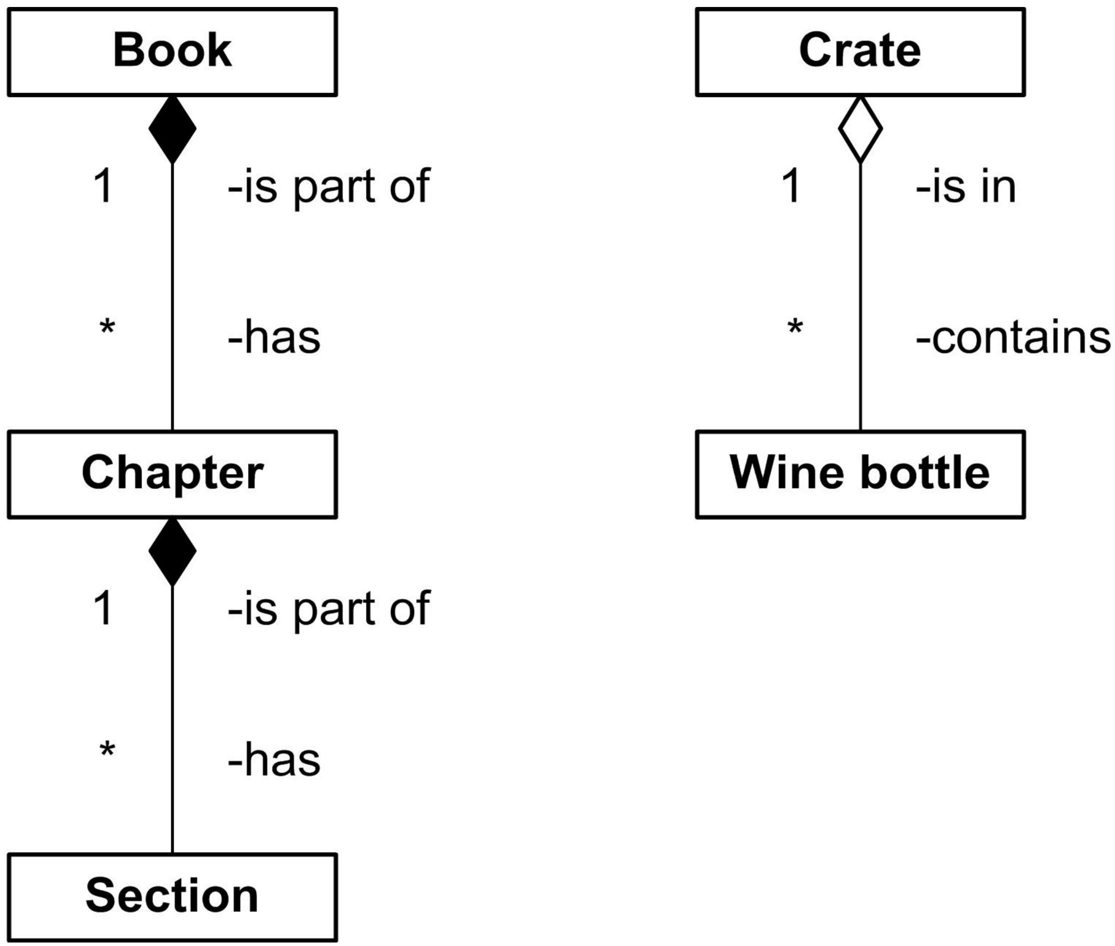
Another Ternary Association Example



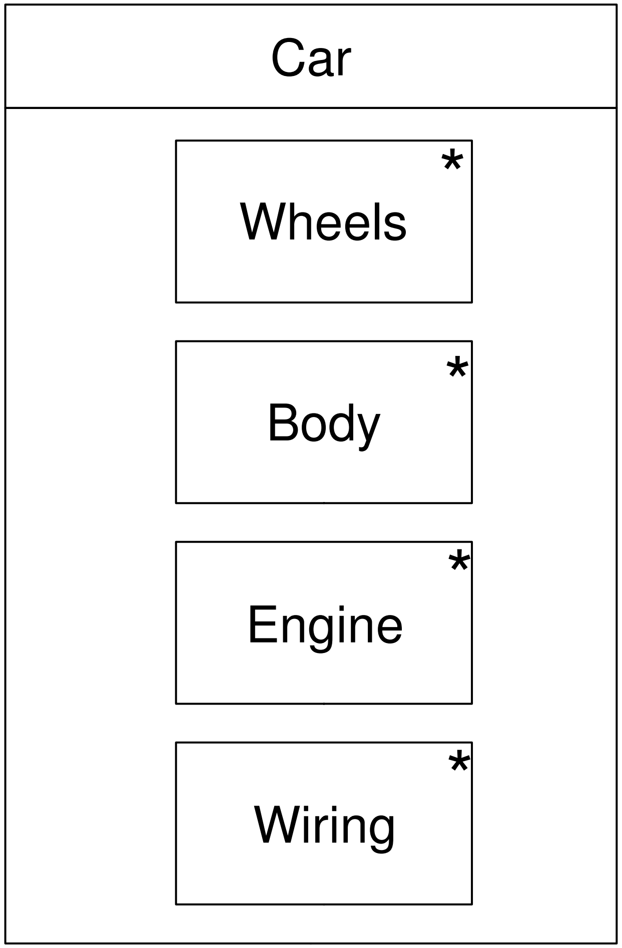
Association Classes



Association by Aggregation

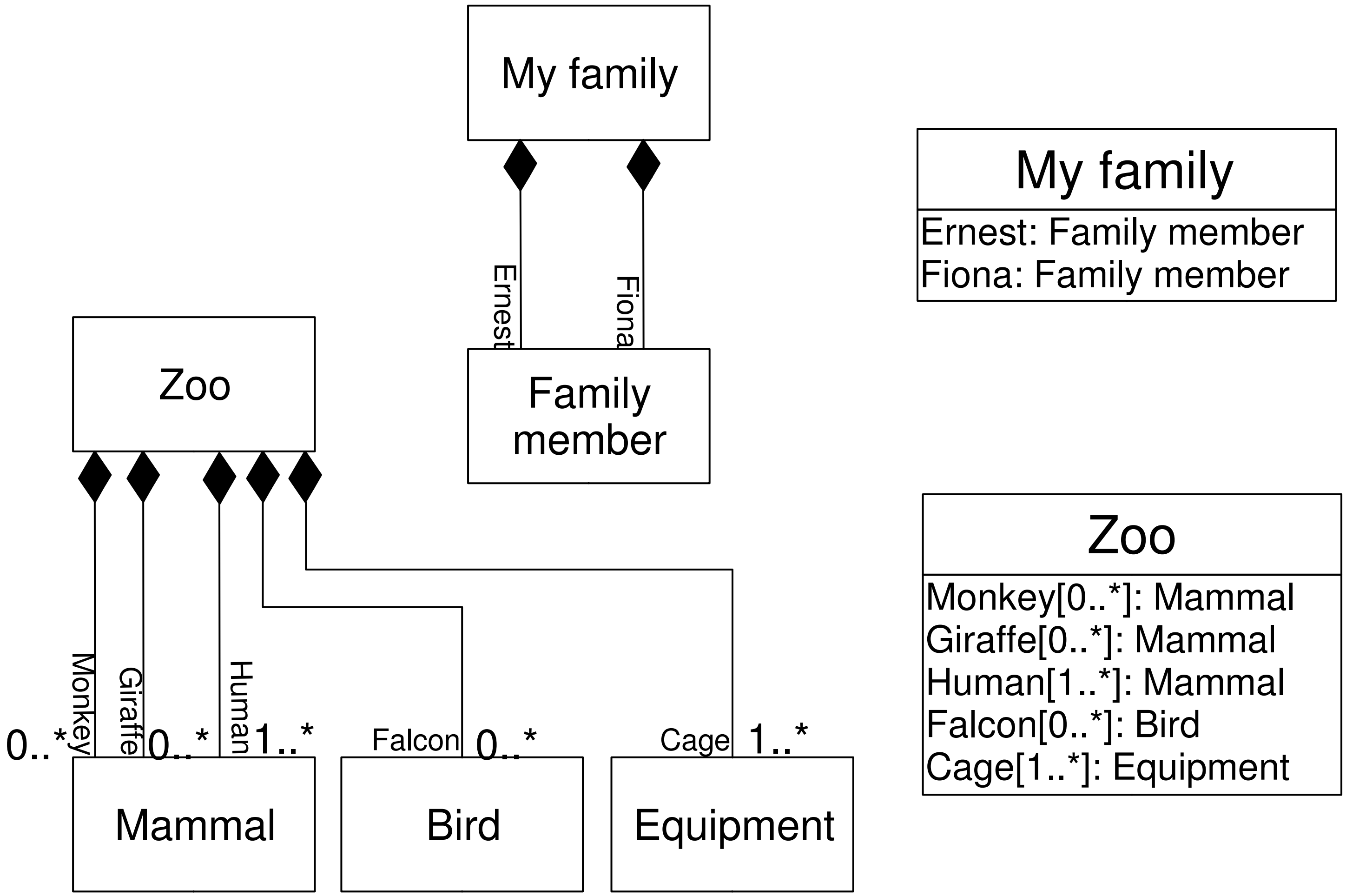


Alternative Notation for Composition Association

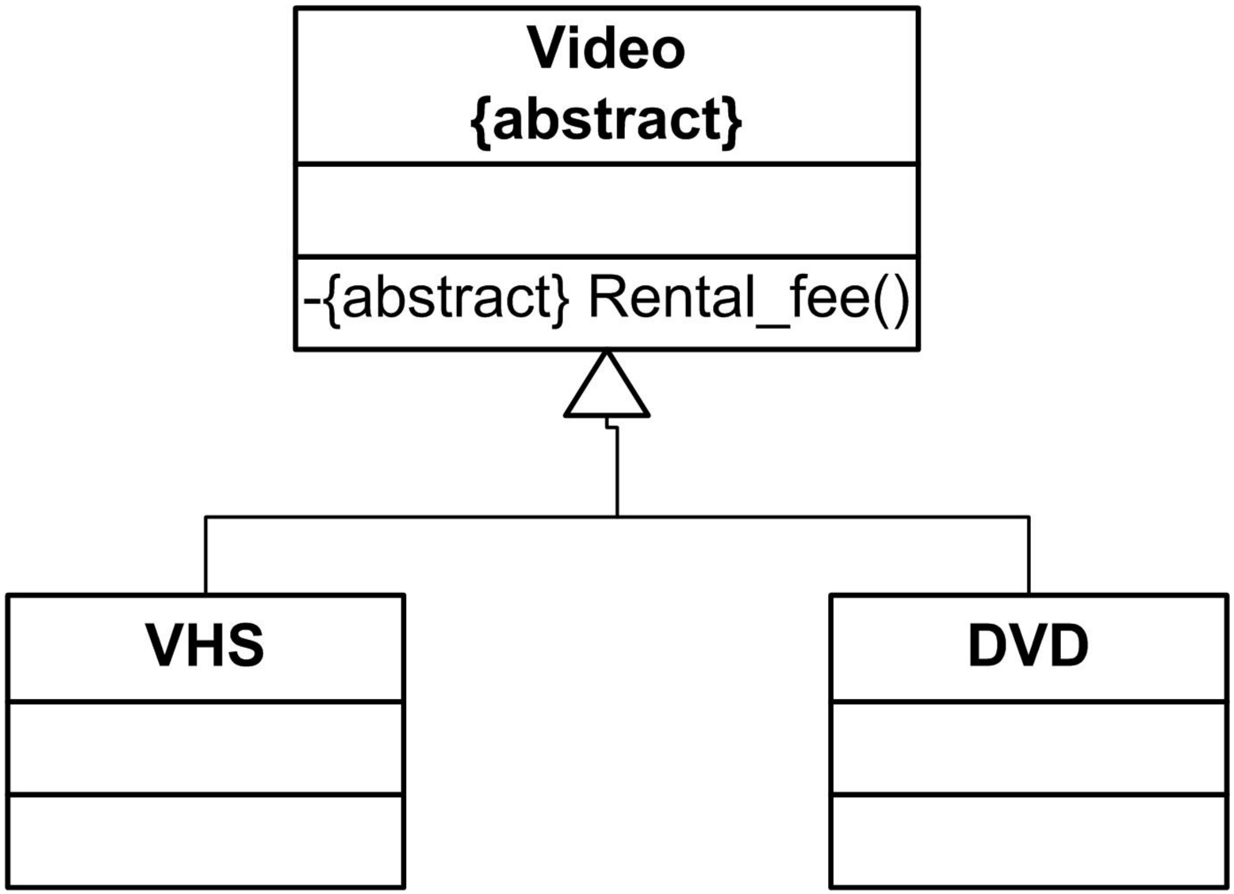


Note that association multiplicity is shown within the classes

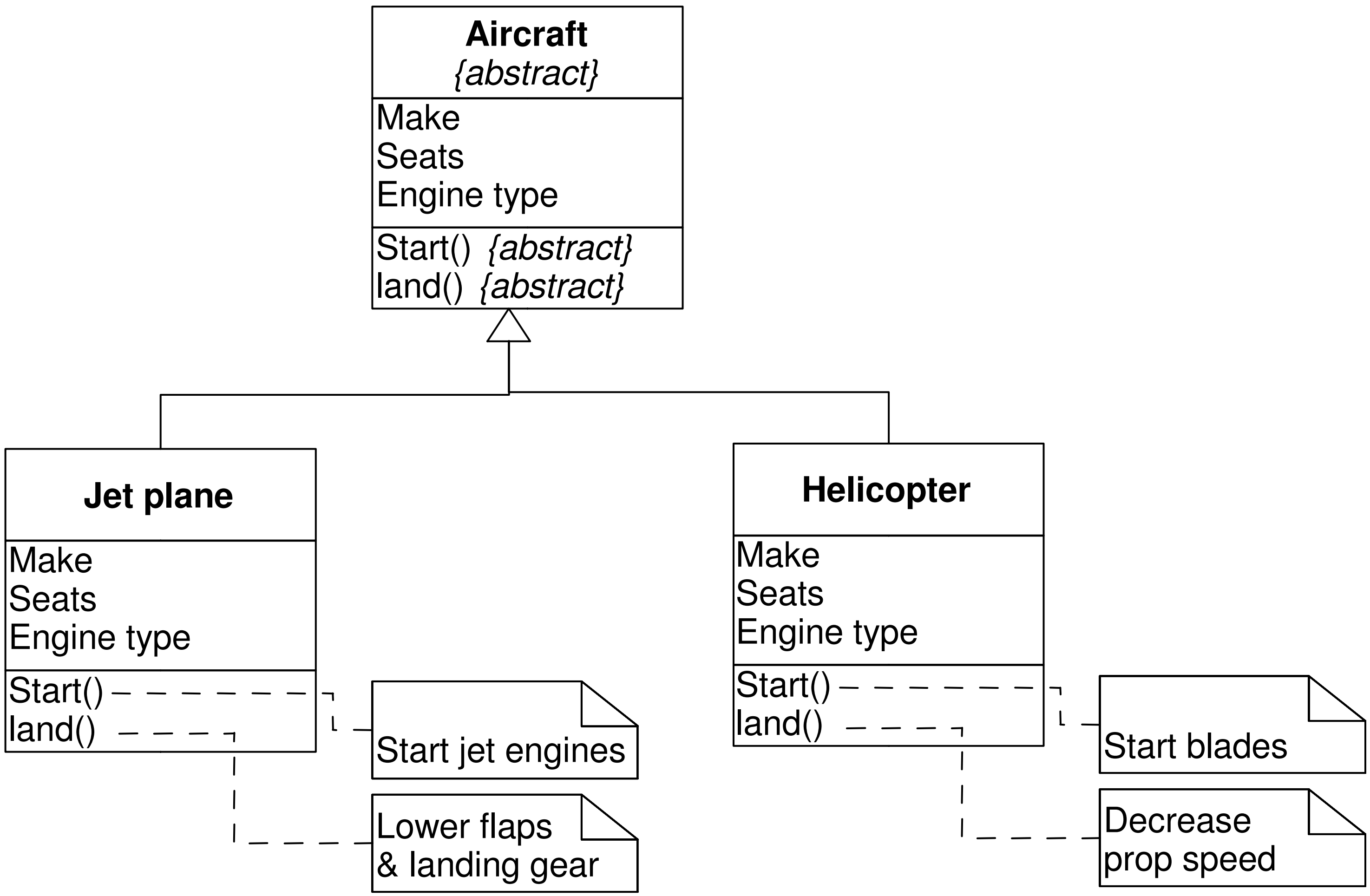
Roles in Aggregation



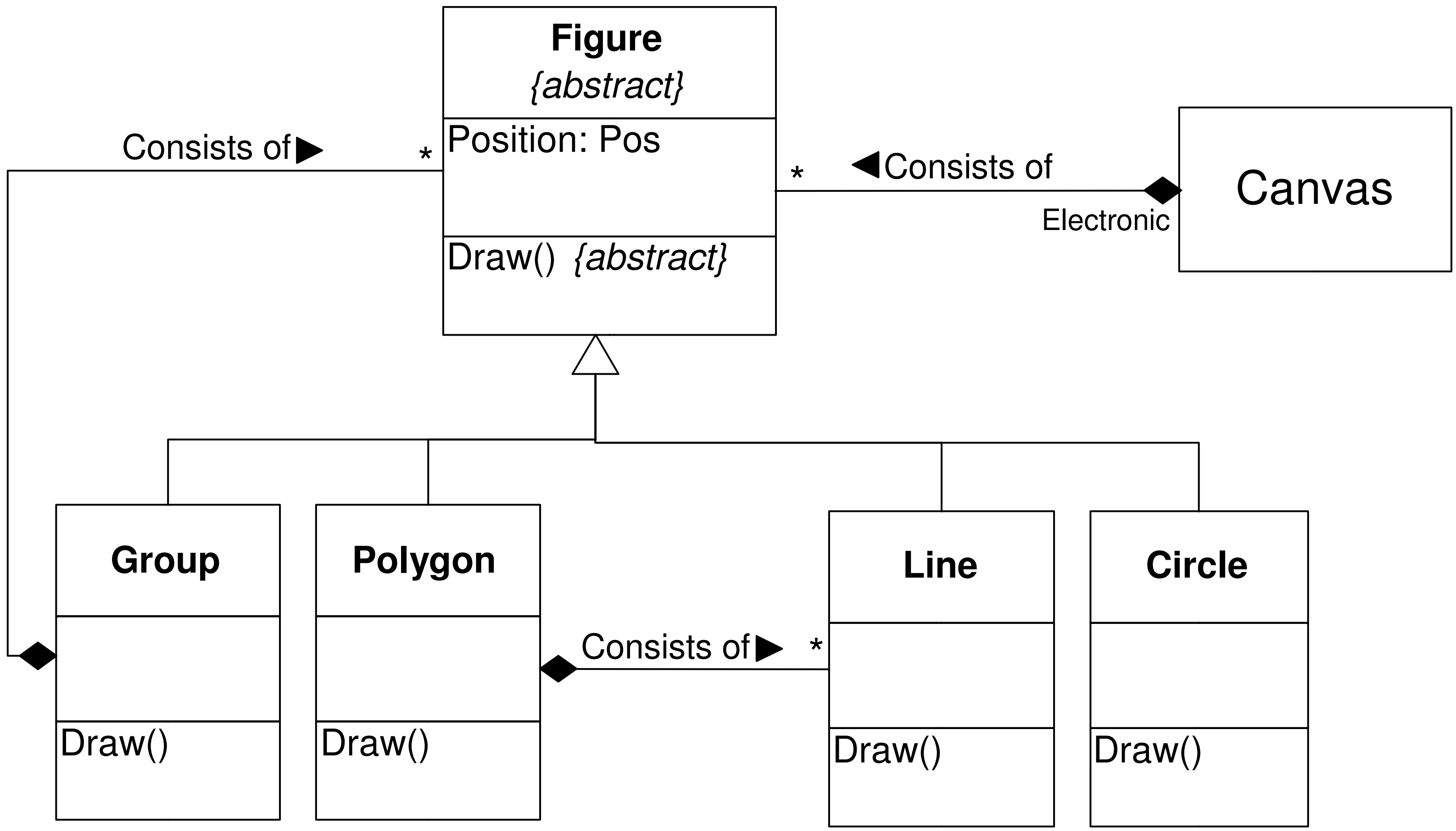
Abstract Classes



Abstract Classes and Generalisation Example



Aggregation and Generalisation



Implementing it (e.g. in Java)

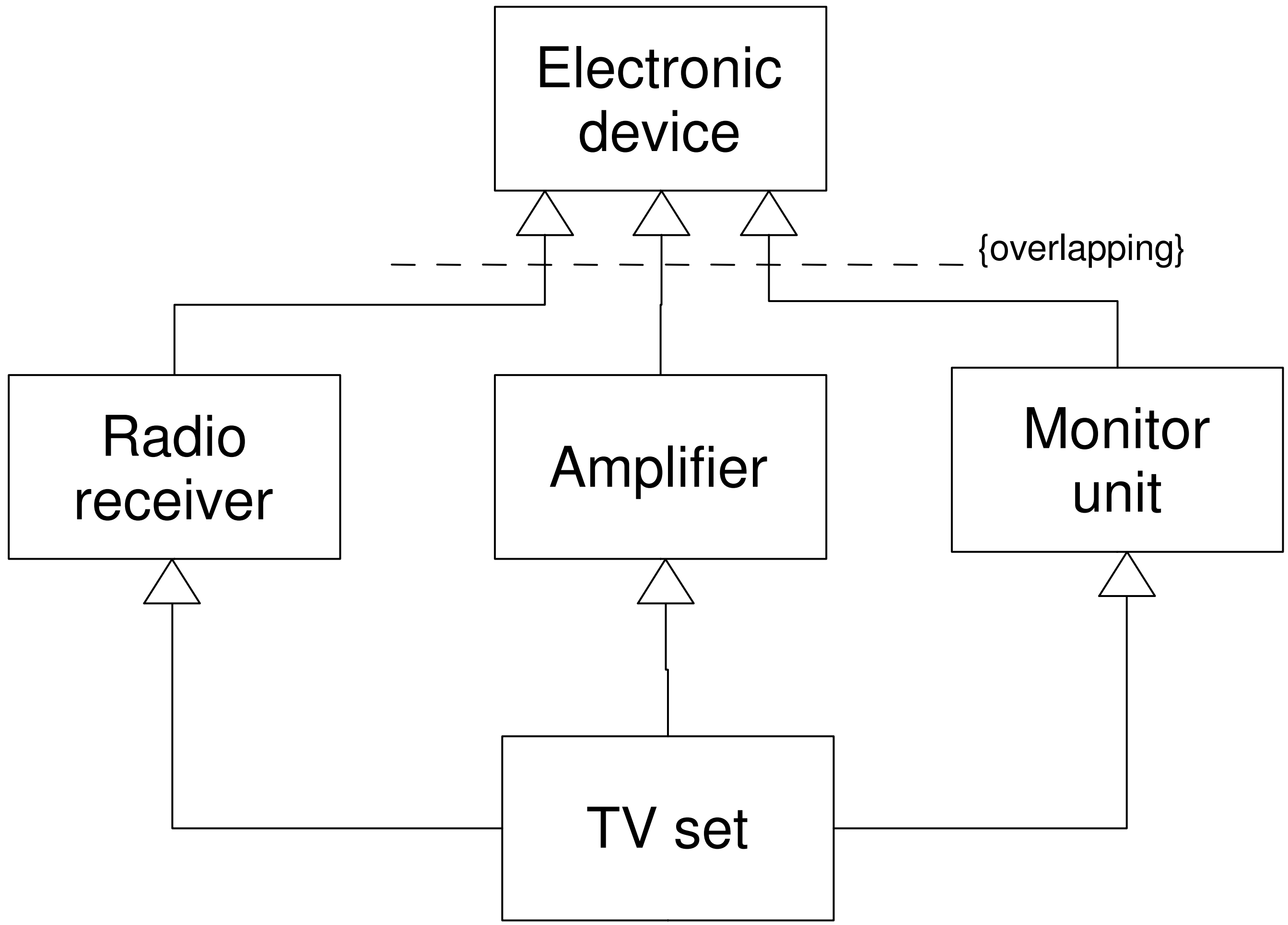
```
abstract public class Figure
{
 abstract public void Draw();
 Pos position;
}
public class Group extends Figure
{
 private FigureVector consist_of;
 public void Draw()
 {
  for (int i = 0; i < consist_of.size(), i++)
  {
   consist_of[i].draw();
  }
 }
}
public class Polygon extends Figure
{
 public void Draw()
 {
  /* something similar to group
     only using lines instead */
 }
}
```

```
public class Line extends Figure
{
 public void Draw()
 {
  /* code to draw line */
 }
}
public class circle extends Figure
{
 public void Draw()
 {
  /* code to draw circle */
 }
}
```

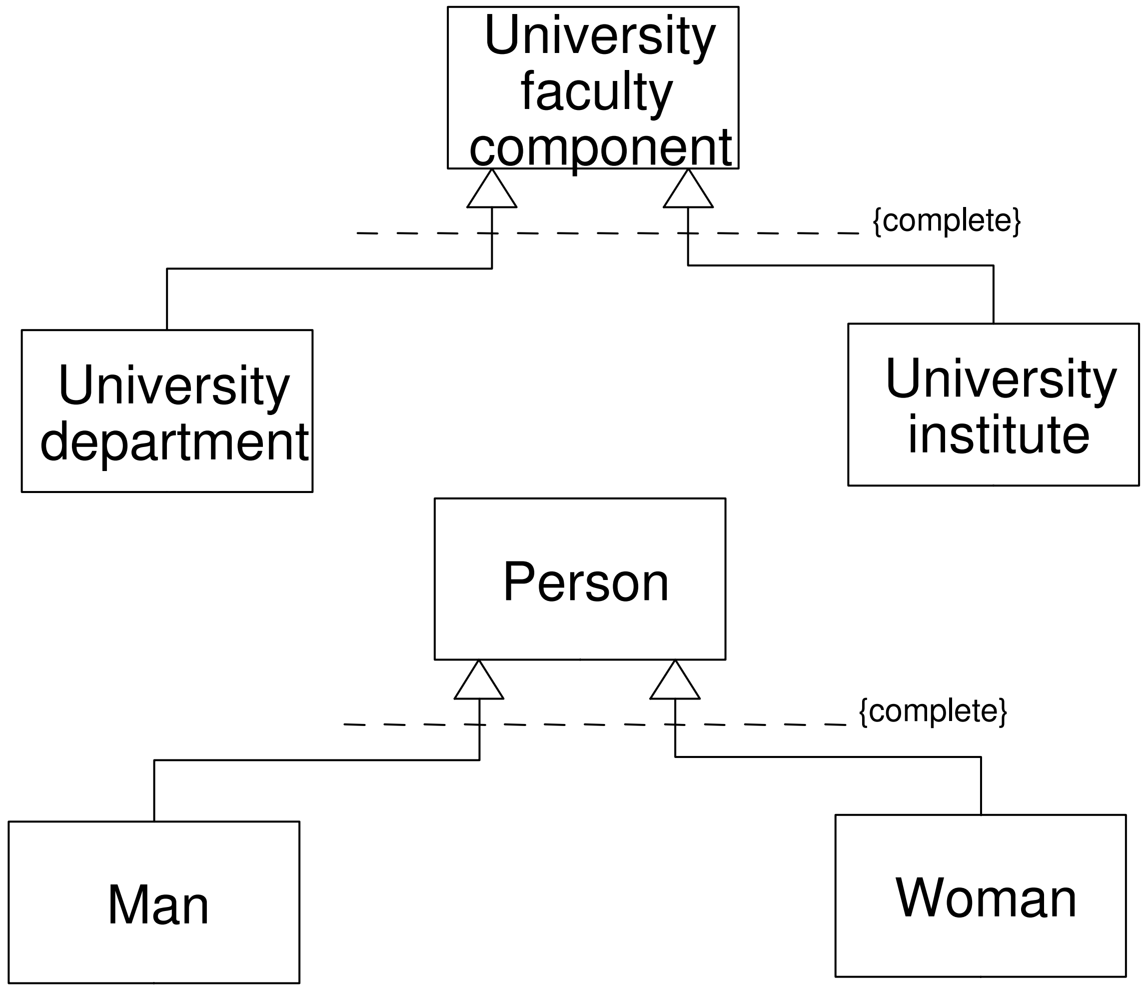
Constrained Generalisations

- Overlapping
  - A type of inheritance whereby sharing of common sub-classes by other sub-classes is allowed.
- Disjoint *(the default)*
  - The opposite of overlapping.
- Complete
  - A type of inheritance whereby the existing sub-classes are said to fully define a given super-class. No further sub-classing may be defined.
- Incomplete *(the default)*
  - Further sub-classes can be added later on to more concretely specify a given super-class.

Overlapping Generalisation



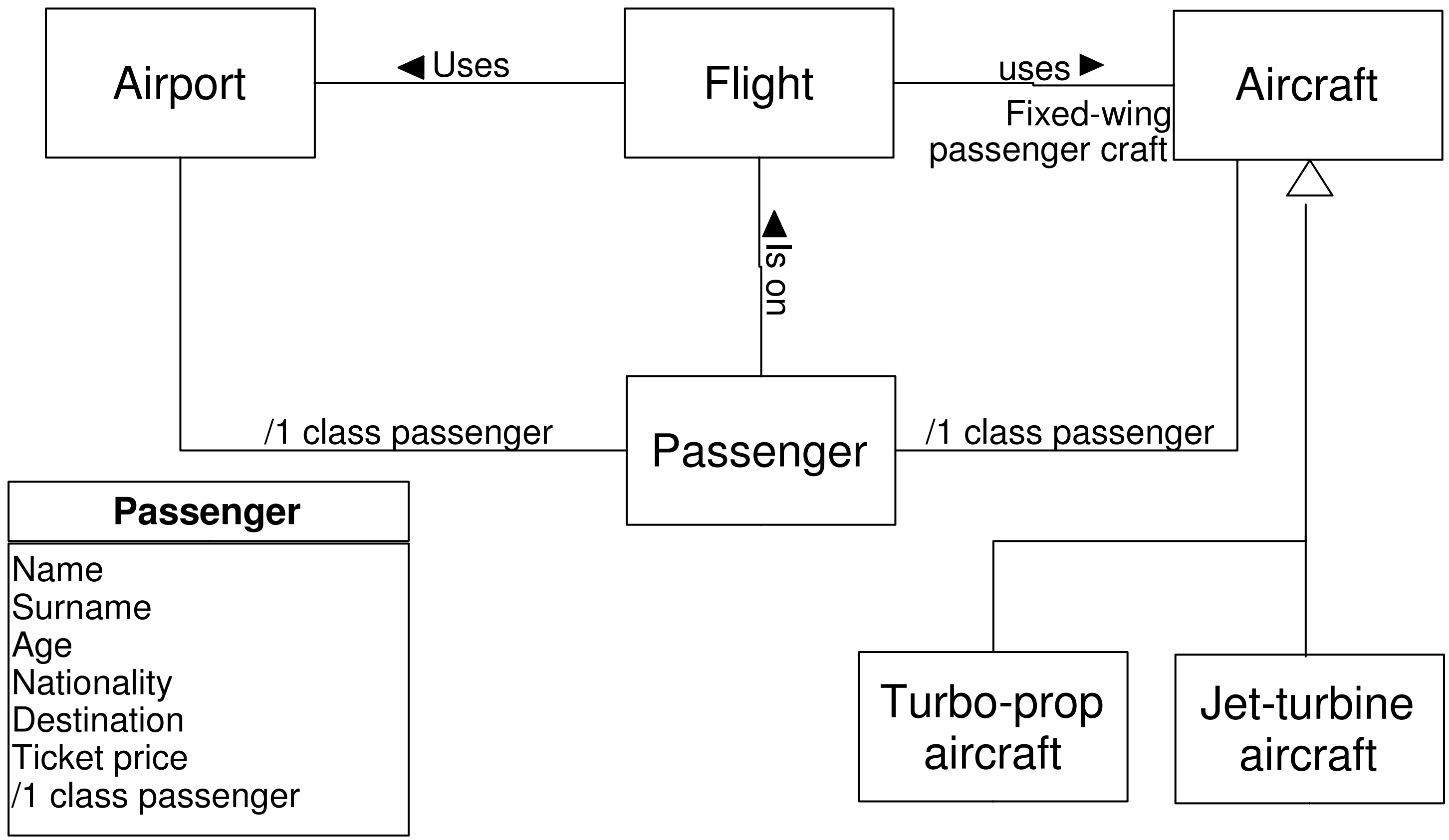
Complete Generalisation



Expressing Rules in UML

- Rules are expressed using constraints and derivations
  - Constraints were mentioned earlier *(e.g. or-associations, ordered associations, inheritance constraints, etc.)*
  - Derivations are rules governing how entities can be derived *(e.g. age = current date - DOB)*

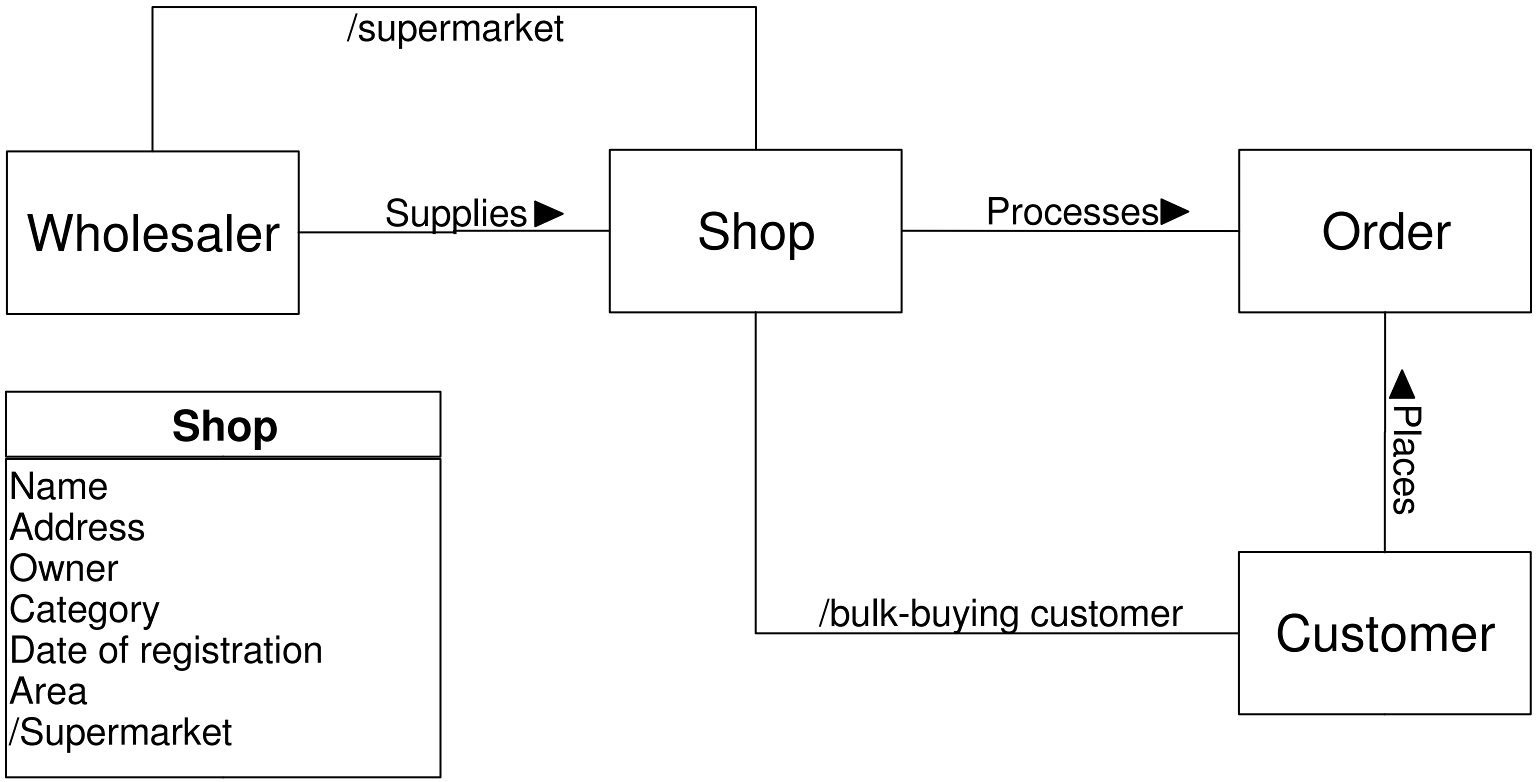
Example of Derived Associations



{1 class passenger = = (Ticket price > 400)}

N.B. Relation cardinality is omitted for example clarity

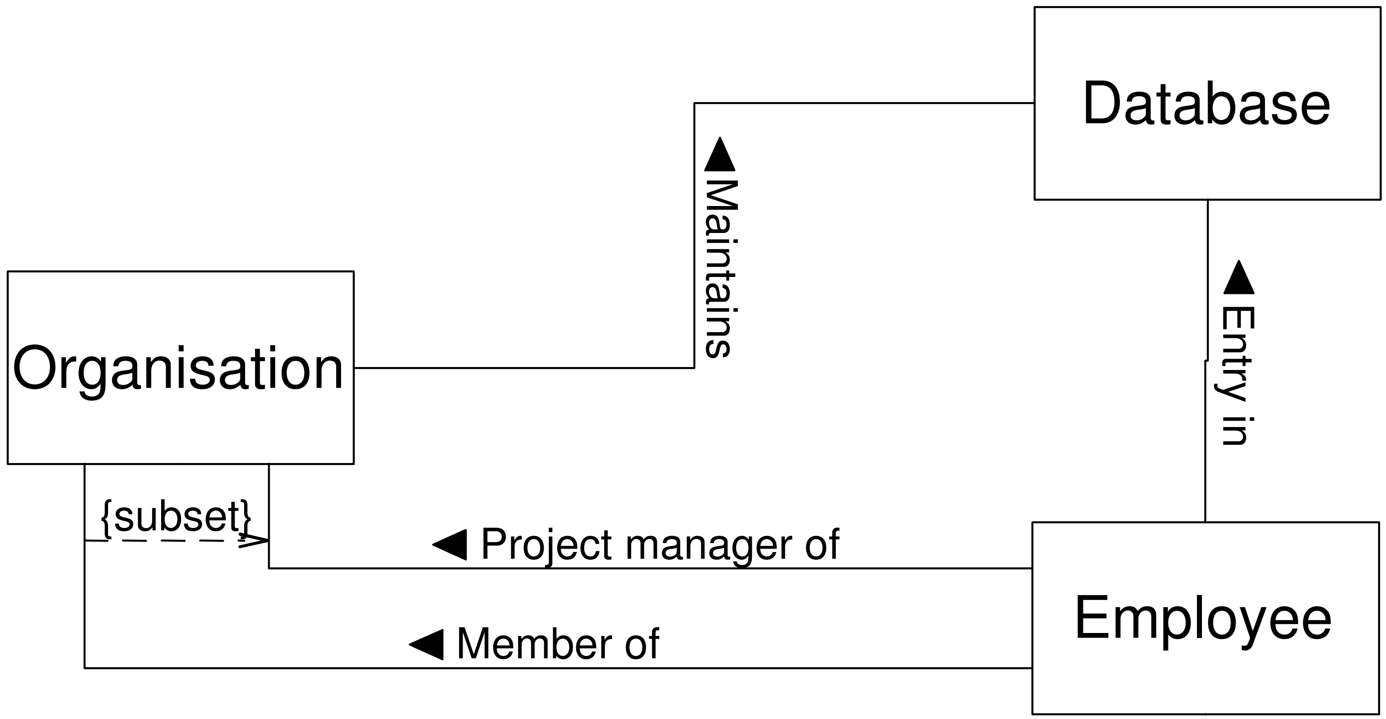
Another Example of a Derived Association



{Supermarket = = (Area > 200 && Category = "dept")}

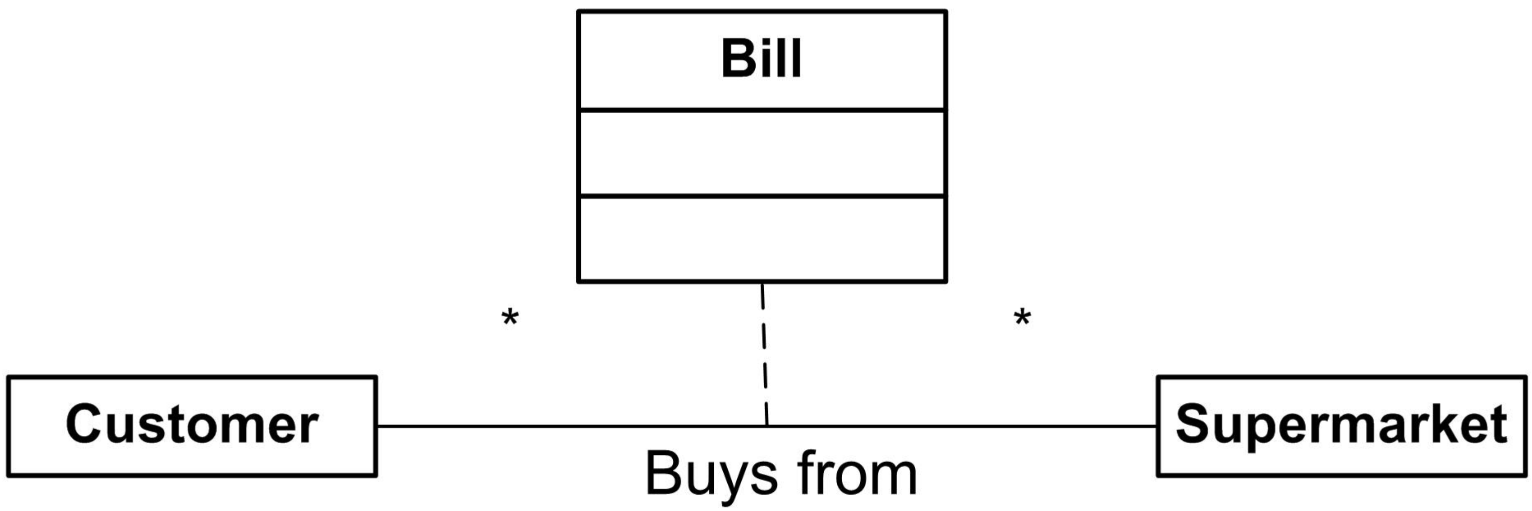
N.B. Relation cardinality is omitted for example clarity

Example of a Constraint Association

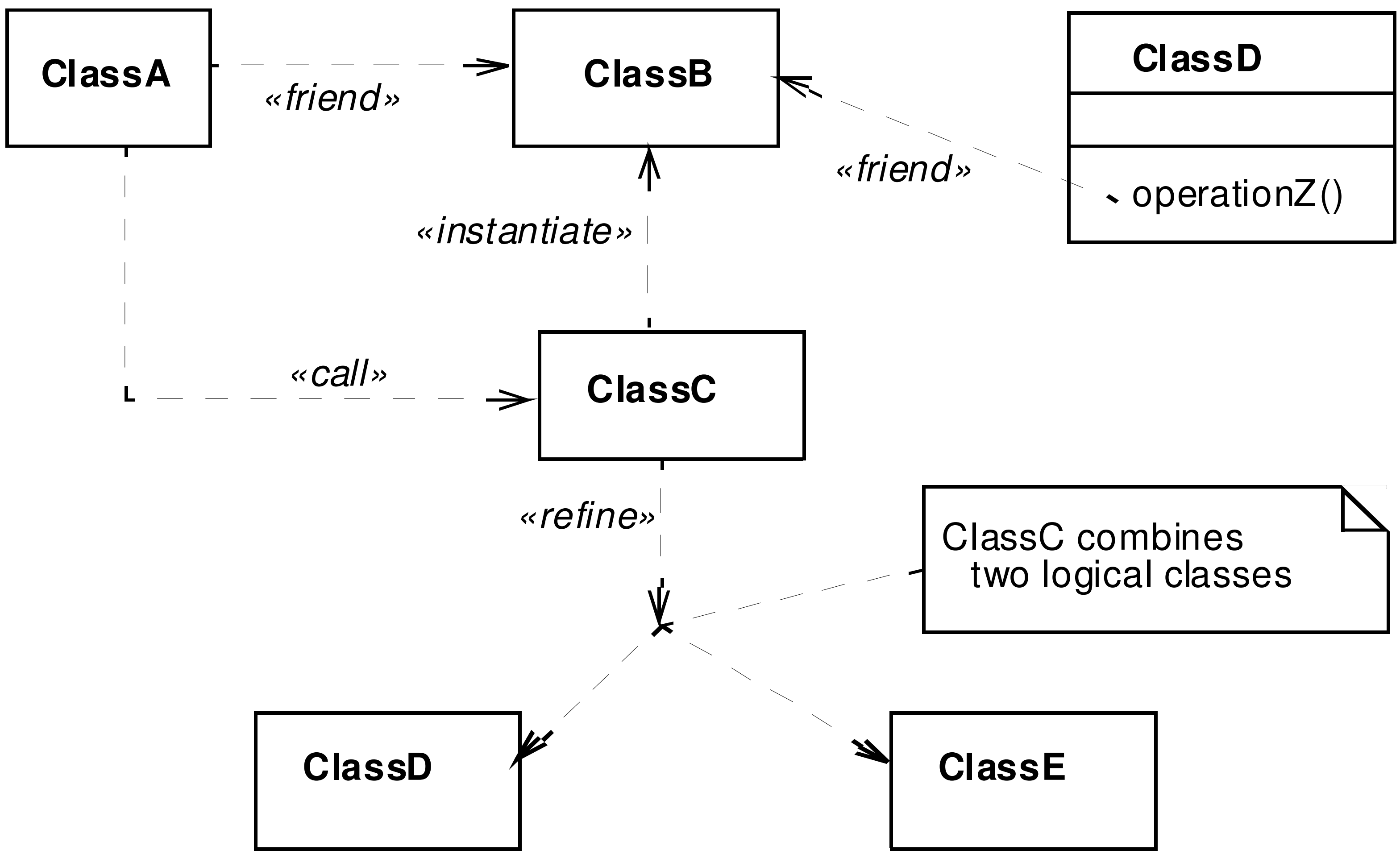


N.B. Relation cardinality is omitted for example clarity

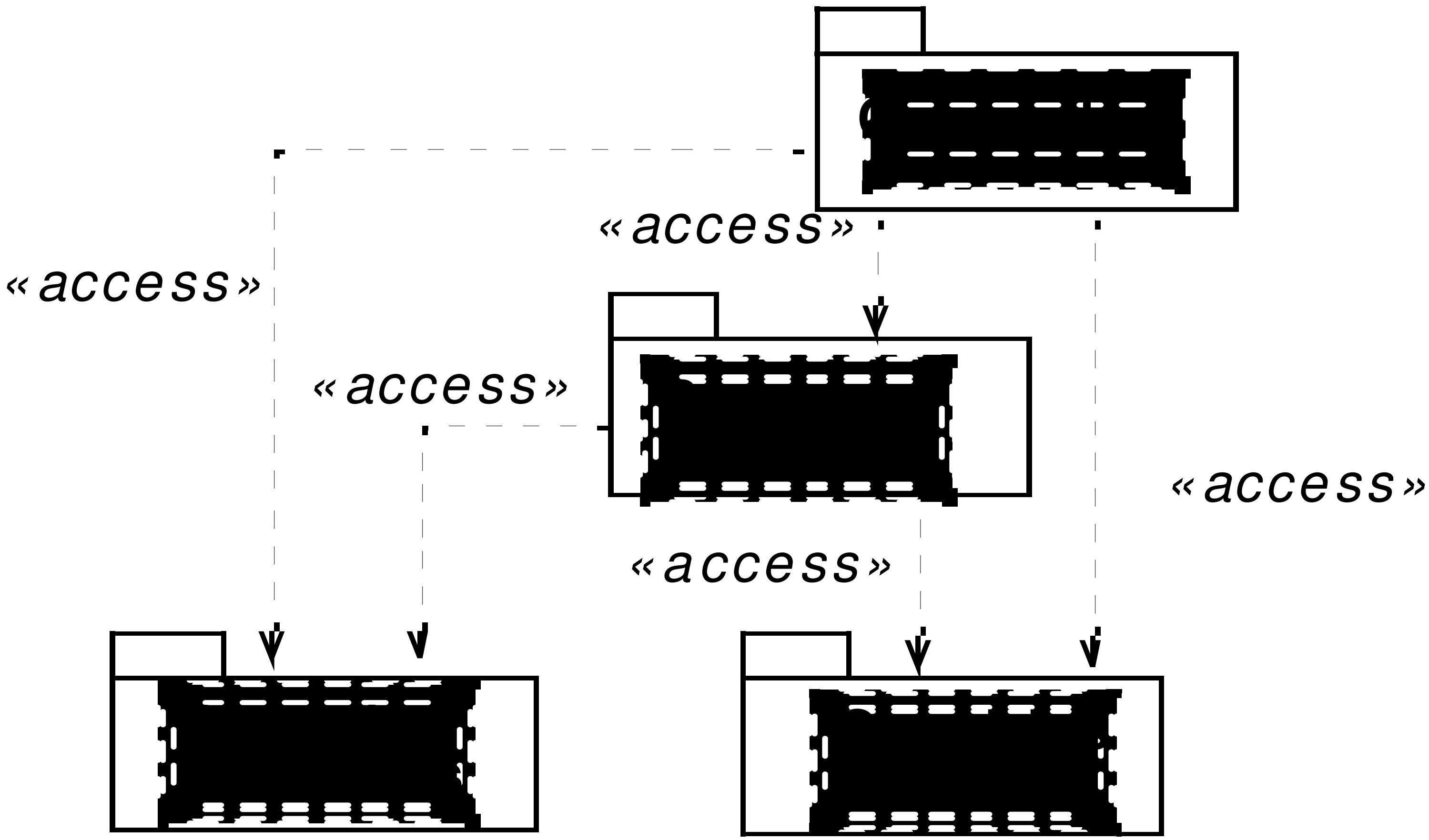
Association Class



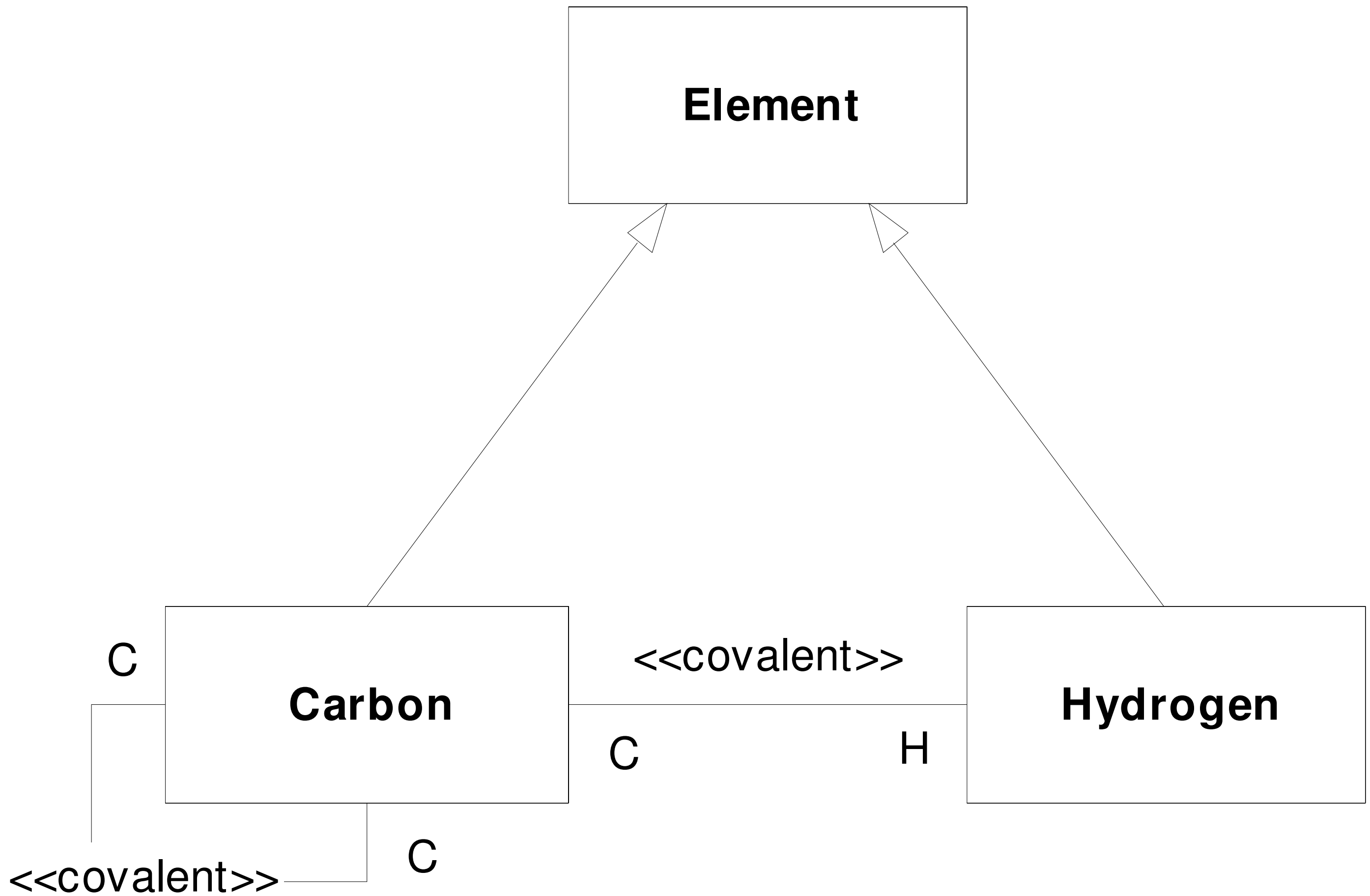
Class Dependencies



Concrete Dependency Example

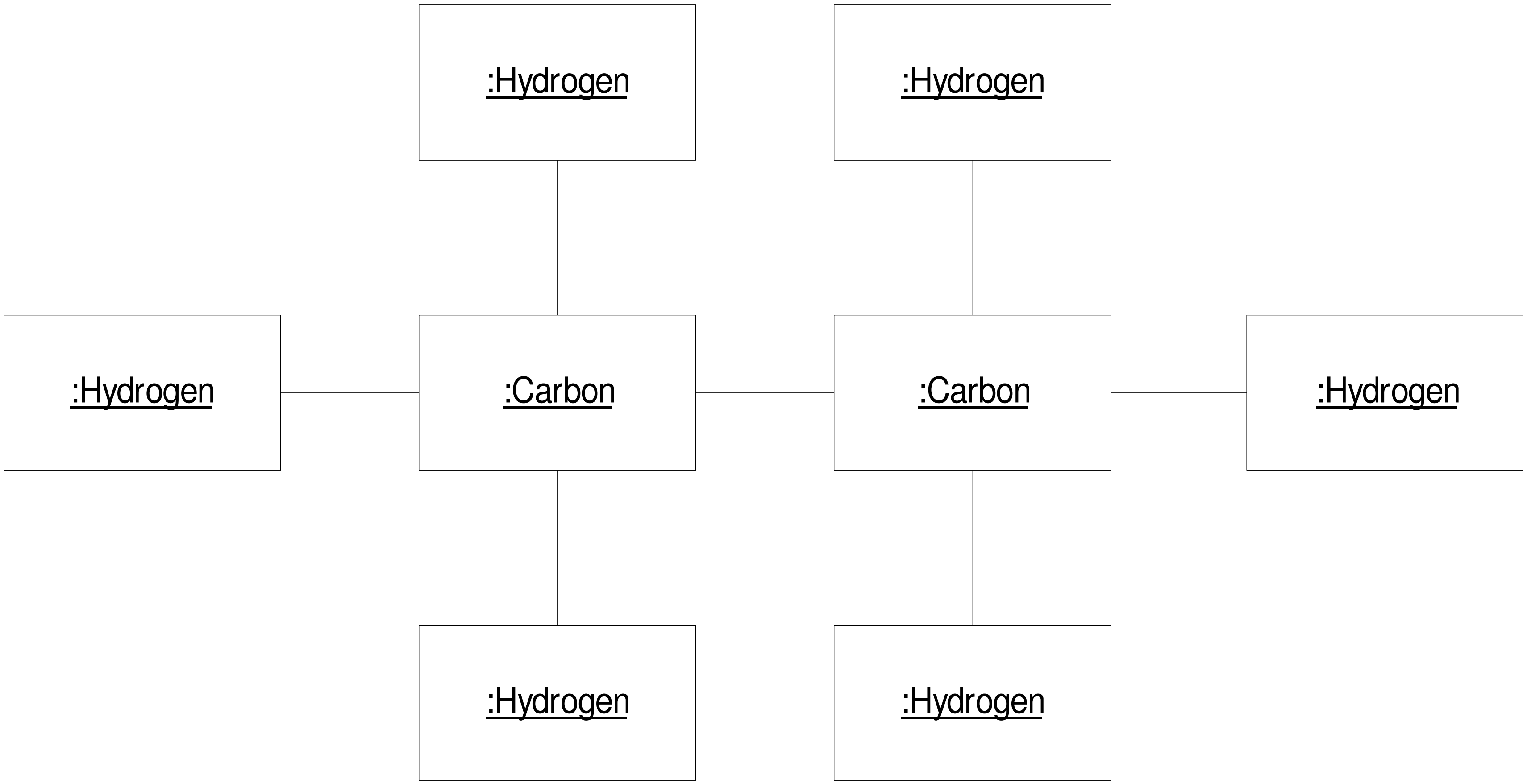


Class Diagram Example

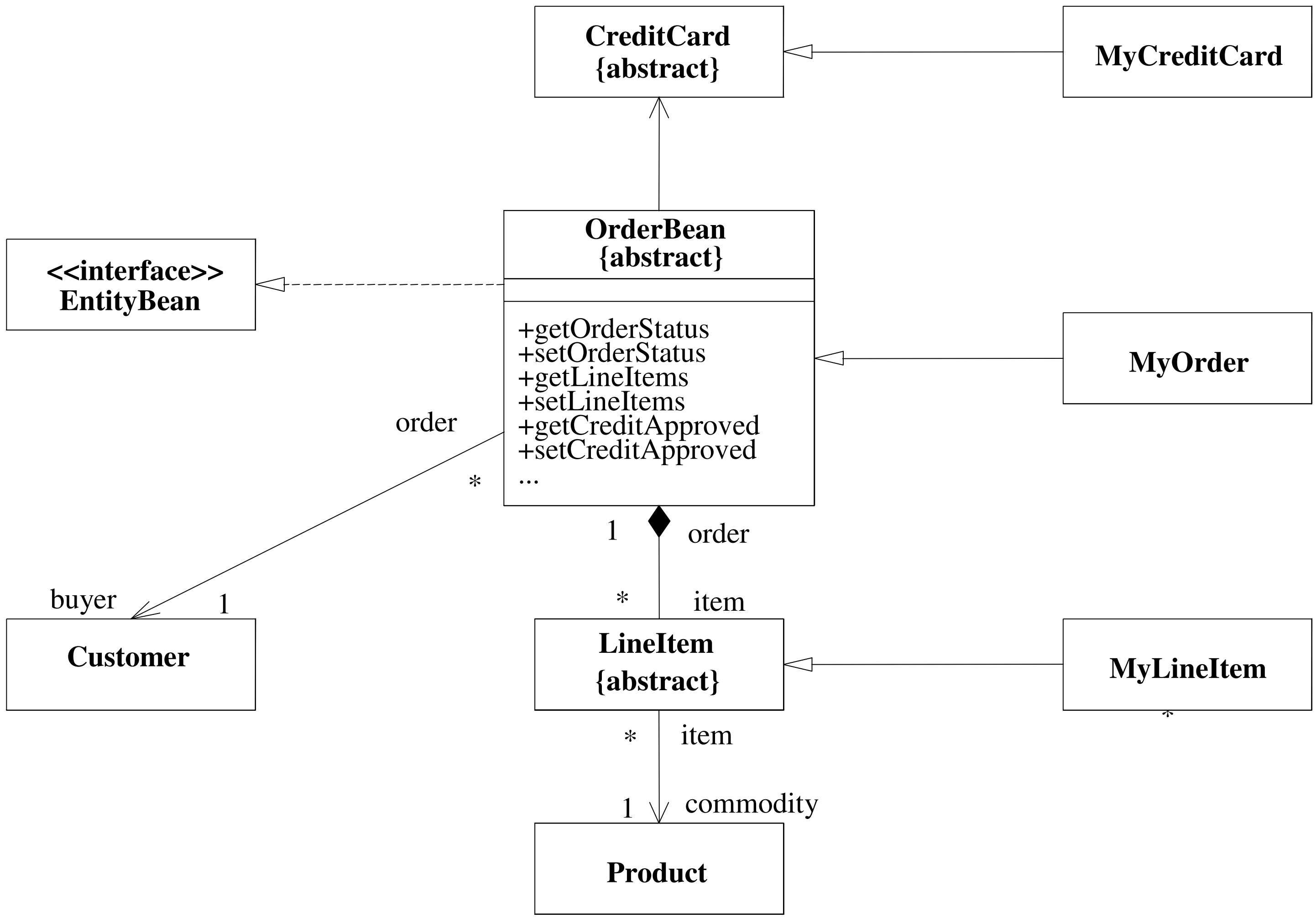


Instantiation of Class Diagram

(in previous slide)

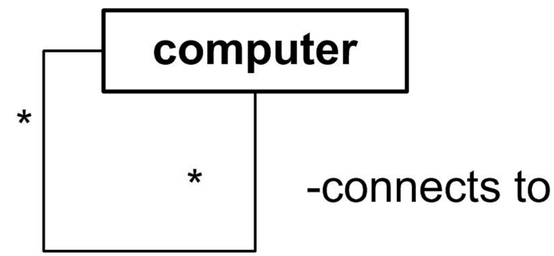


Another Class Diagram Example

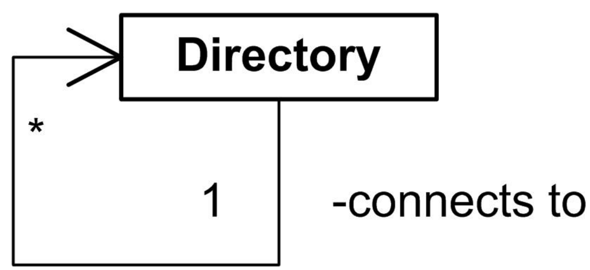


Try This Yourselves…

- Create a class diagram to represent a arbitrary interconnection of computers



  - Create a class diagram to represent a hierarchical directory system in any OS



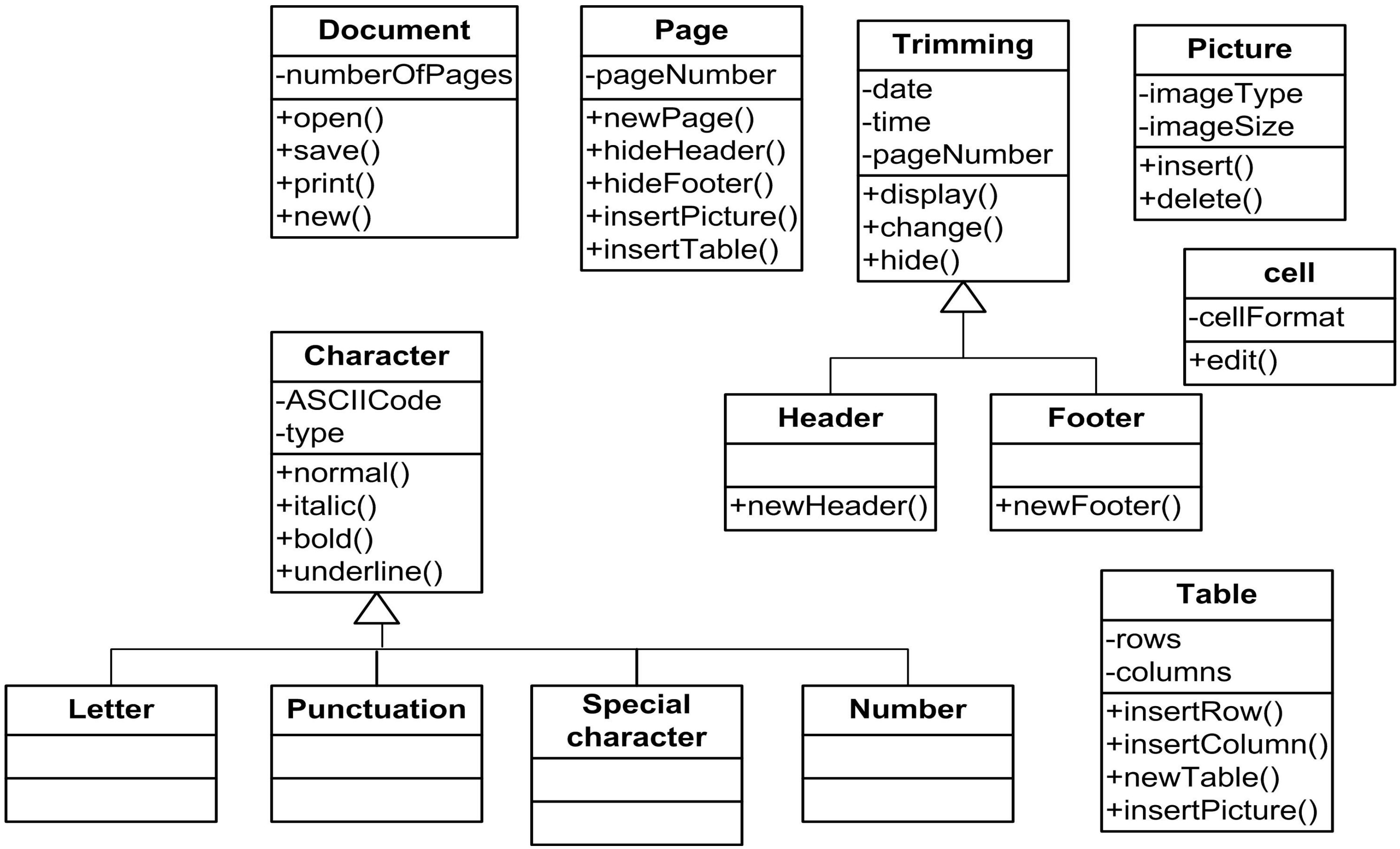
CD Case Study (1/3)

Describing the use of a word processor

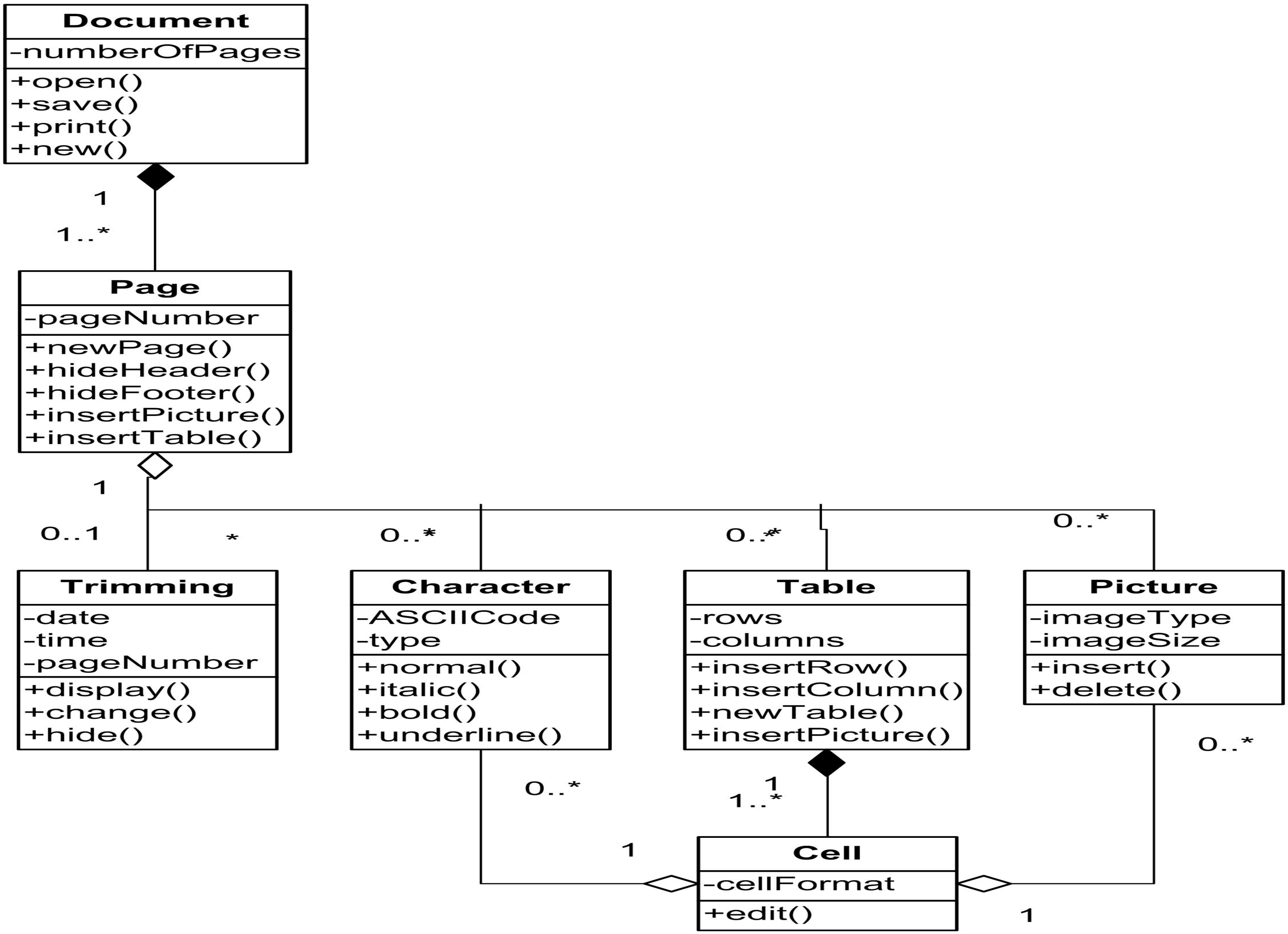
A <u>user</u> can *open* a new or existing <u>document</u>. <u>Text</u> is *entered* through a <u>keyboard</u>. A document is made up of several <u>pages</u> and each page is made up of a <u>header</u>, <u>body</u> and <u>footer</u>. <u>Date</u>, <u>time</u> and <u>page number</u> may *be added* to header or footer. Document body is made up of <u>sentences</u>, which are themselves made up of <u>words</u> and <u>punctuation characters</u>. Words are made up of <u>letters</u>, <u>digits</u> and/or <u>special characters</u>. <u>Pictures</u> and <u>tables</u> may *be inserted* into the document body. Tables are made up of <u>rows</u> and <u>columns</u> and every <u>cell</u> in a table can contain both text and pictures. Users can *save* or *print* documents.

CD Case Study (2/3)

- Nouns (underlined in previous) are either classes or their attributes
- Verbs (italicised in previous) are class operations
- Main handled entity: document



CD Case Study (3/3)



Some Points to Ponder

1. Give two examples to distinguish between aggregation and composition.
2. Explain the concept of an abstract class – give one example.
3. When do you think the use of generalisation is not justified in model building?
4. Link in a class diagram with generalisation the classes: "Campaign", "Advert", "Newspaper advert", "Magazine advert", "Advert copy", "Advert graphic", and "Advert photograph".
5. Draw a class diagram for the following classes:
    - Film (title; producer; length; director; genre)
    - Ticket (price; adult/child; show time; film)
    - Patron (name; adult/child; DOB)
6. Link the classes in (5) through message passing and services offered for *any one* scenario of your choice.

Workshop Activity -11-

Draw a CD for a patient billing system. Include only the attributes that would be appropriate for the system's context.

Patient (name, gender, address, ID, tel., DOB, blood type, occupation, pass-times, adverse habits, insurance carrier, dietary preferences)

Doctor (name, category, specialist, warrant No., preferred sport, address, tel., DOB, weekly income, VAT No.)

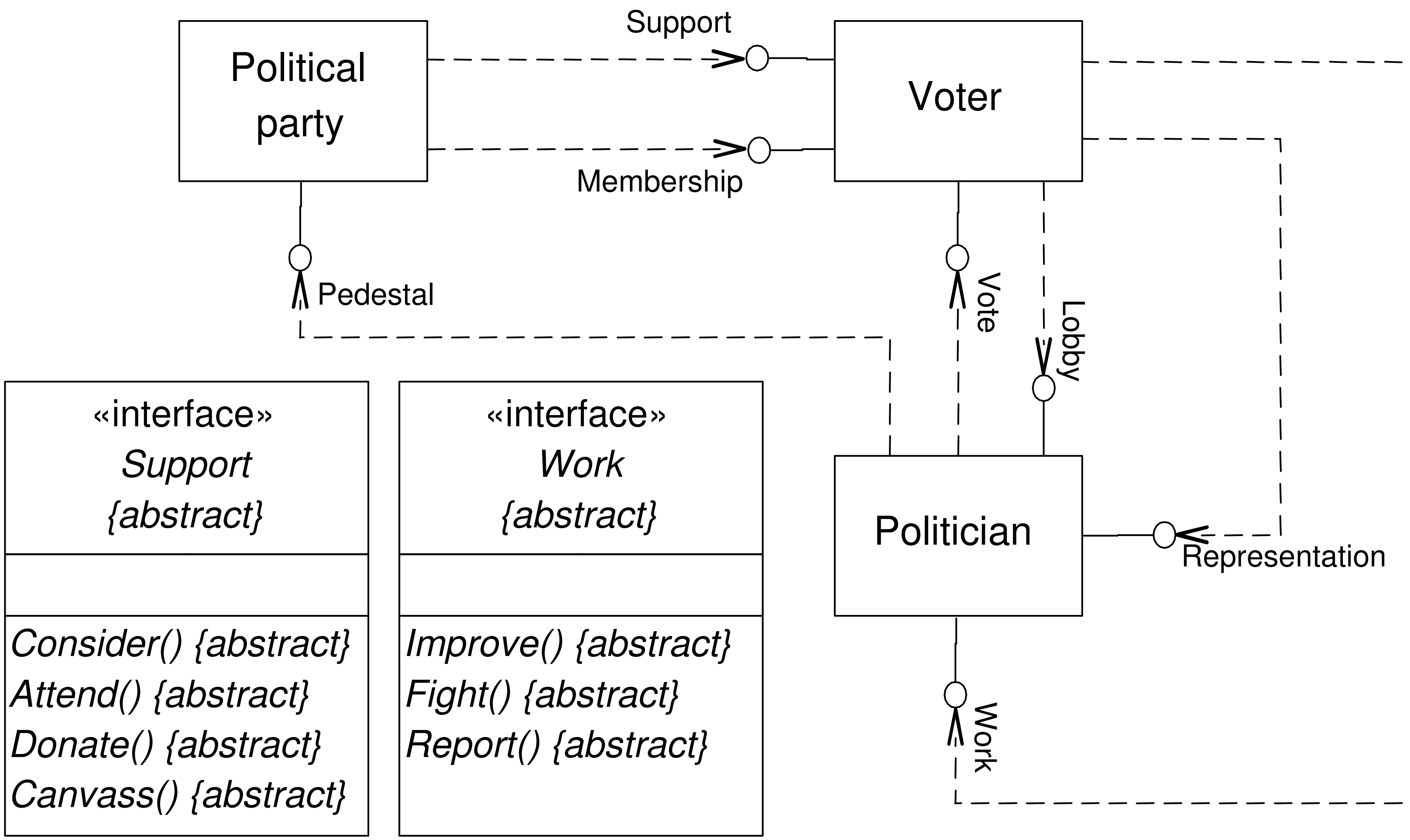
Insurance carrier (date of establishment, name, registration ID, company staff size, address, tel., contact person name)

Create two object diagrams (ODs) based on the CD you develop.

UML Interfaces

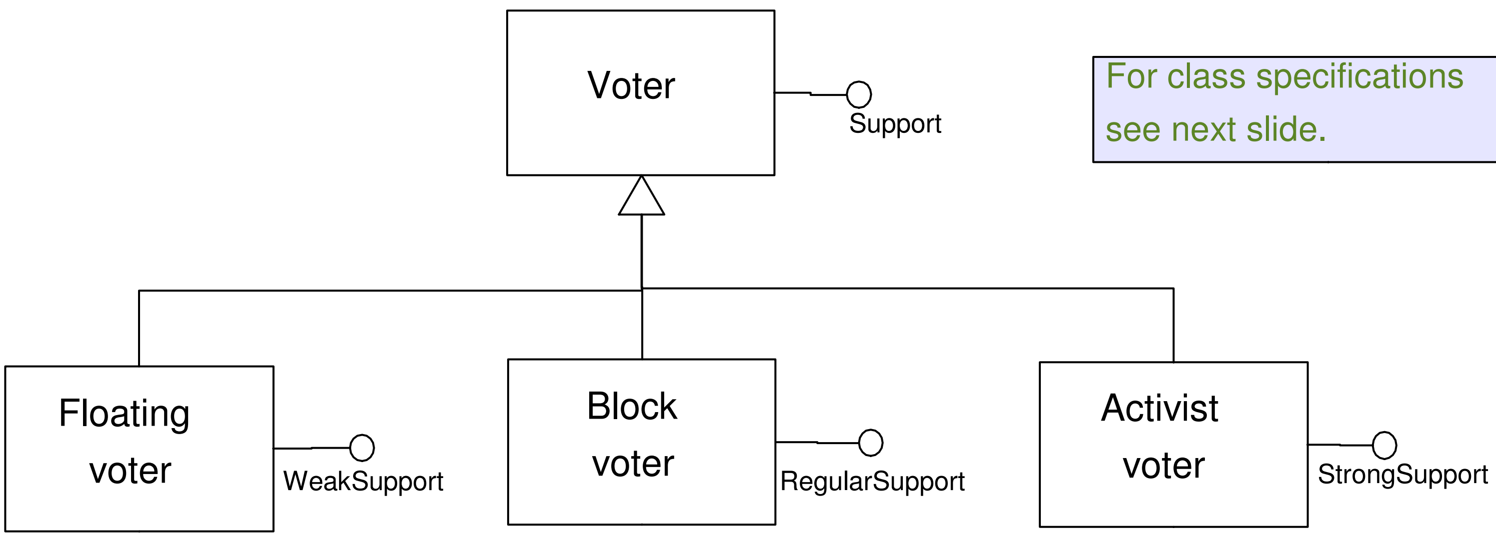
- Interfaces are associated with supporting model elements (package, component, class).
- Act as contact points between collaborating model elements and/or their clusters.
- Equivalent to such programming structures as OLE/COM or Java interfaces.
- An interface is abstractly defined.
- An interface is composed of signatures, that as a whole, specify the behaviour of a model element.

UML Interface Example



UML Interface Specialisation

- Interfaces are subject to inheritance in the same way as classes are. Interface inheritance can be shown on a class diagram.



For class specifications see next slide.

N.B. Only one interface is shown for example clarity

UML Interface Classes *(based on previous example)*

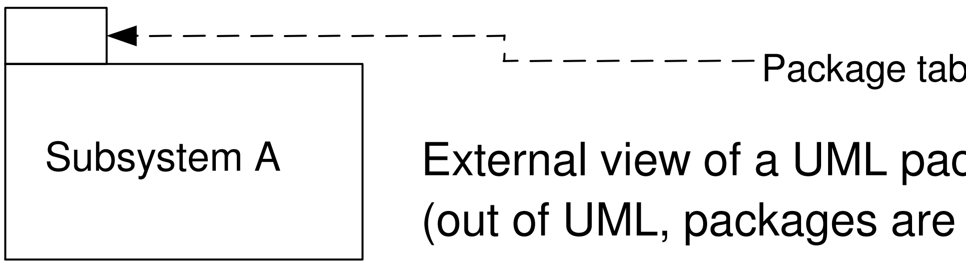
«interface» *WeakSupport* *{abstract}*	«interface» *RegularSupport* *{abstract}*	«interface» *StrongSupport* *{abstract}*
Consider() {abstract}	*Consider() {abstract}* *Attend() {abstract}*	*Consider() {abstract}* *Attend() {abstract}* *Donate() {abstract}* *Canvass() {abstract}*

Please note, that whether regular support should include either party activity attendance or the donation of funds (or indeed both) is something of which I haven't the vaguest idea. It is, however, irrelevant to this example.

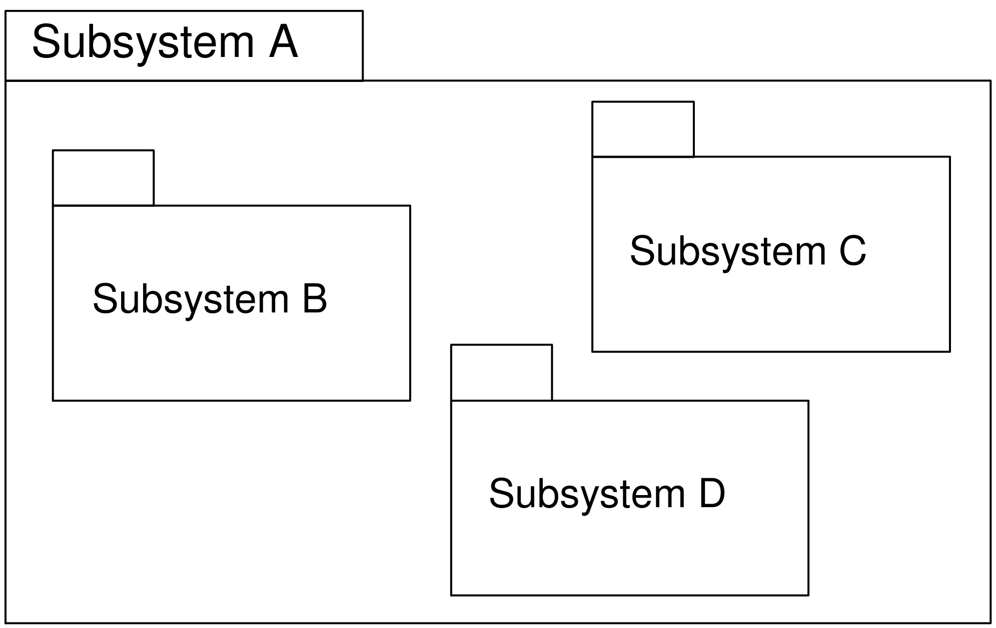
UML Packages

- Can be considered as a general purpose grouping mechanism (as opposed to a regular UML diagram)
- May be used to group different types of model elements
- Model elements in a package (group) are taken to be related semantically
- Packages can only be related by dependencies, refinements, or generalisations
- Any one modelling element can be located in only one package (i.e. Packages cannot share model elements)

Examples of UML Packages and their Logical Grouping

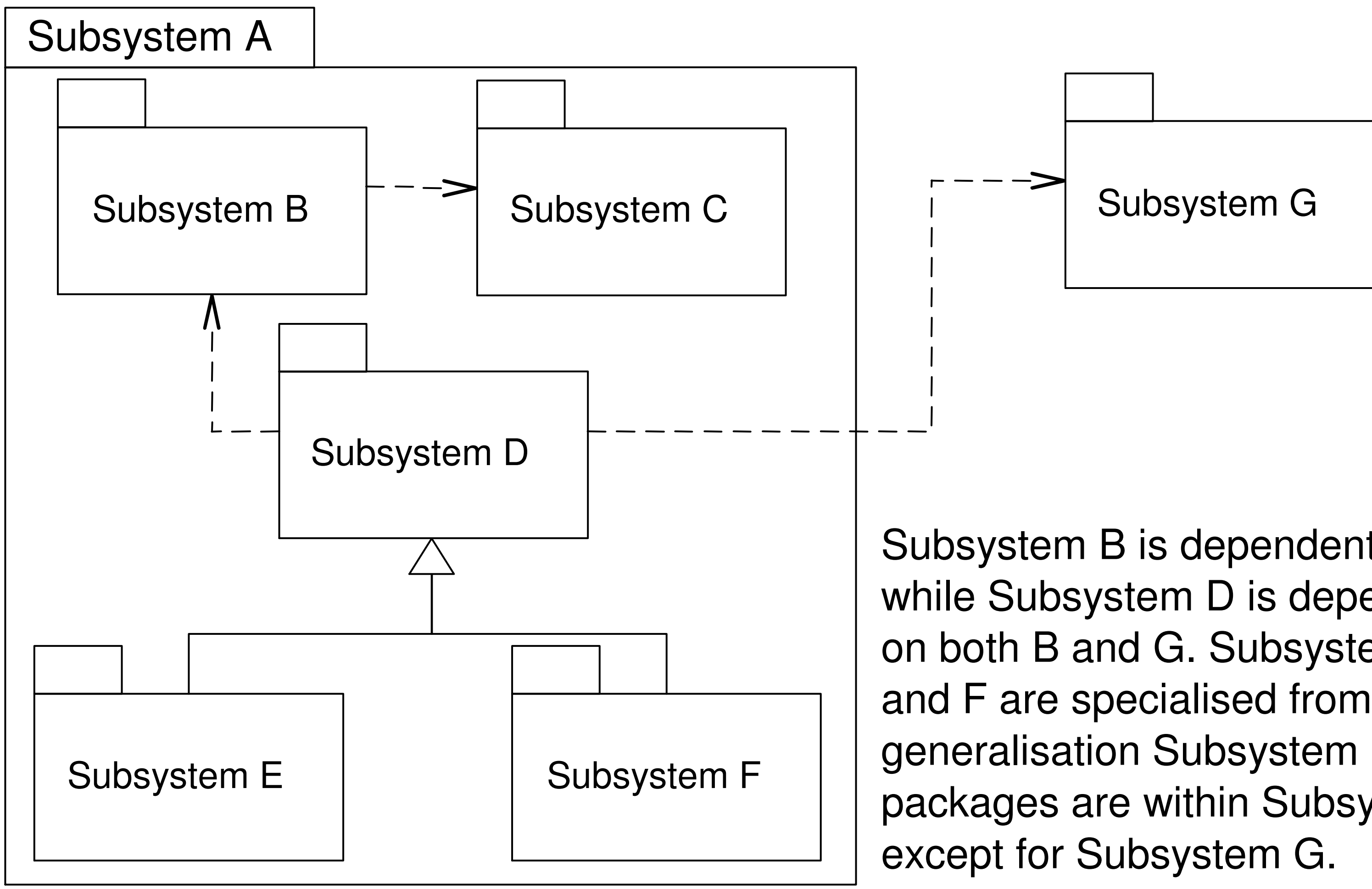


External view of a UML package named "Subsystem A" (out of UML, packages are referred to as a subsystems)



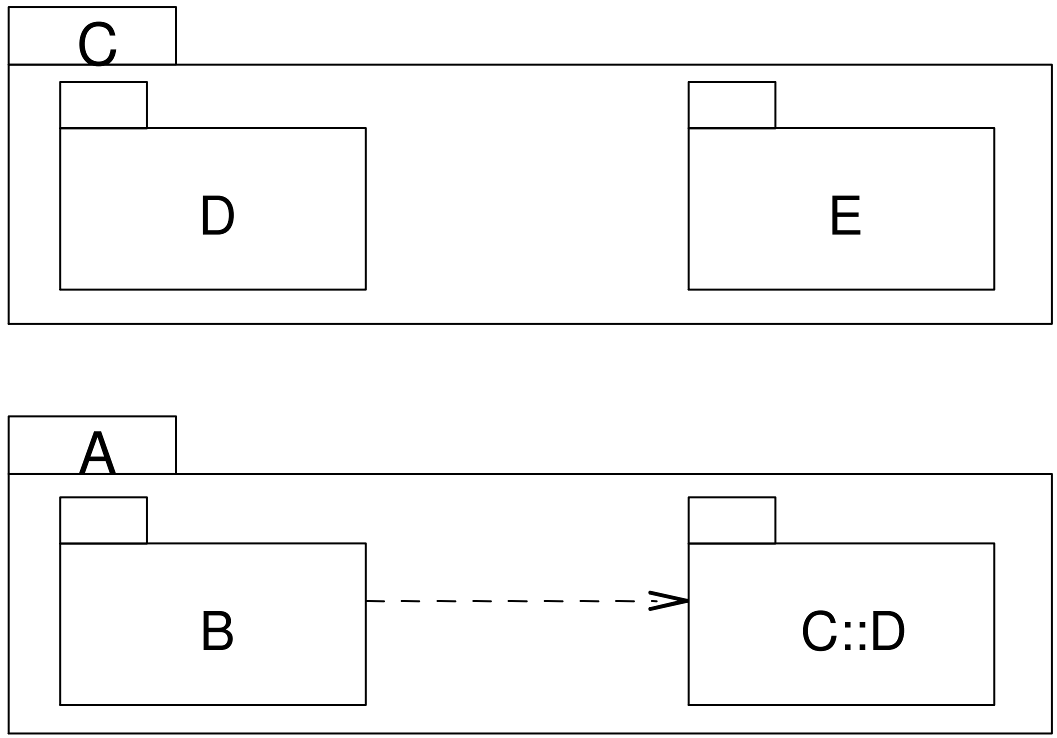
Expanded view of "Subsystem A" showing that it groups together three other packages named "Subsystem B/C/D". Note, that the name of an expanded package is indicated in the package tab. This is another way of showing composed aggregation.

Examples of Relationships between UML Packages



Subsystem B is dependent on C while Subsystem D is dependent on both B and G. Subsystems E and F are specialised from the generalisation Subsystem D. All packages are within Subsystem A except for Subsystem G.

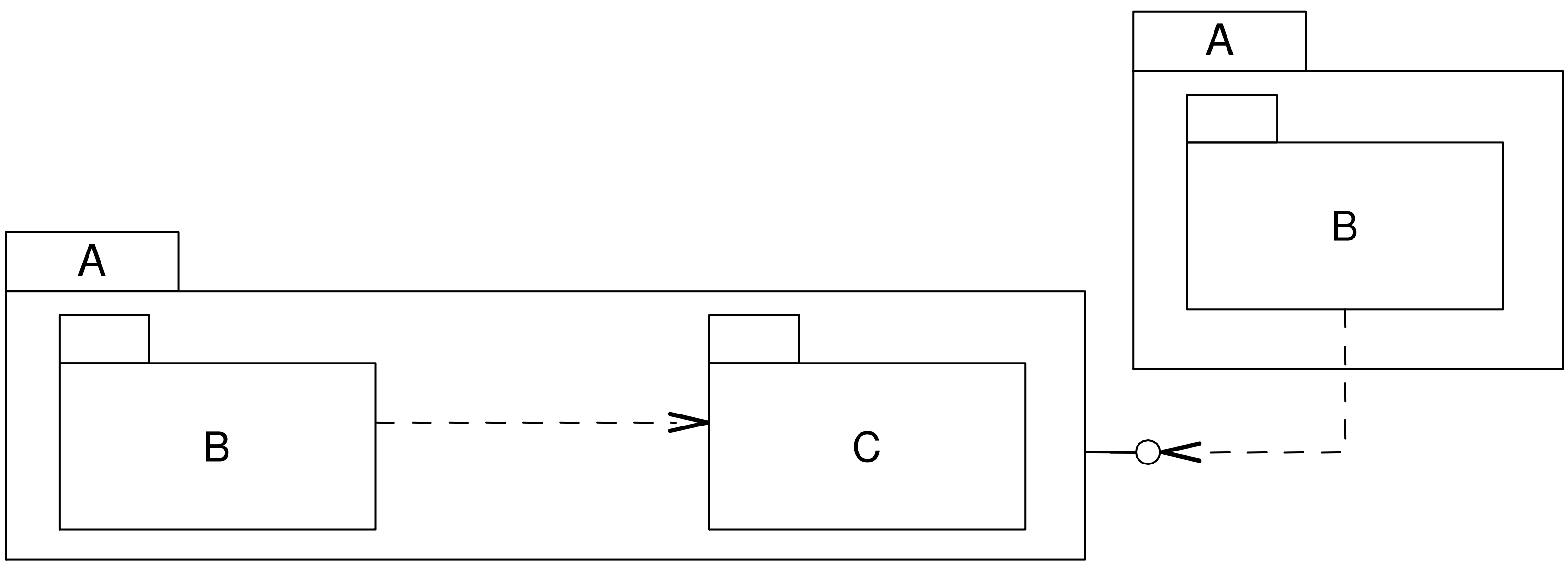
Examples of UML Package Importation



In the above example, package "B" is dependant on the Imported package "D" from package "C".

Even Packages have Faces!

- Publishes package behaviour
- Same symbol as for a class interface
- Classes within the given package then implement the particular interface



Packaging Steps

1. Set the context
2. Cluster classes together based on shared relationships
3. Model clustered classes as a package
4. Identify dependency relationships amongst packages
5. Place dependency relationships between packages

Workshop Activity -12-

Package and dependency-link the classes in the following system:

Assuming that an automated medical appointment system is to be partitioned as:

- HCI layer
- Problem Definition layer
- Data Management layer

Furthermore, system classes are identified as:

- Patient UI
- Appointment UI
- Patient management
- Appointment management
- Patient data management
- Appointment data management
- Patient DB
- Appointment DB

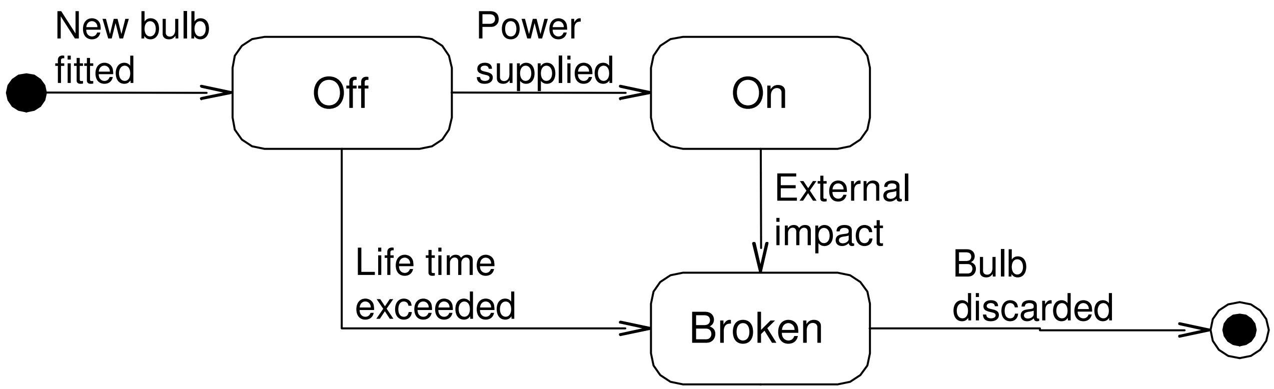
Workshop Activity -13-

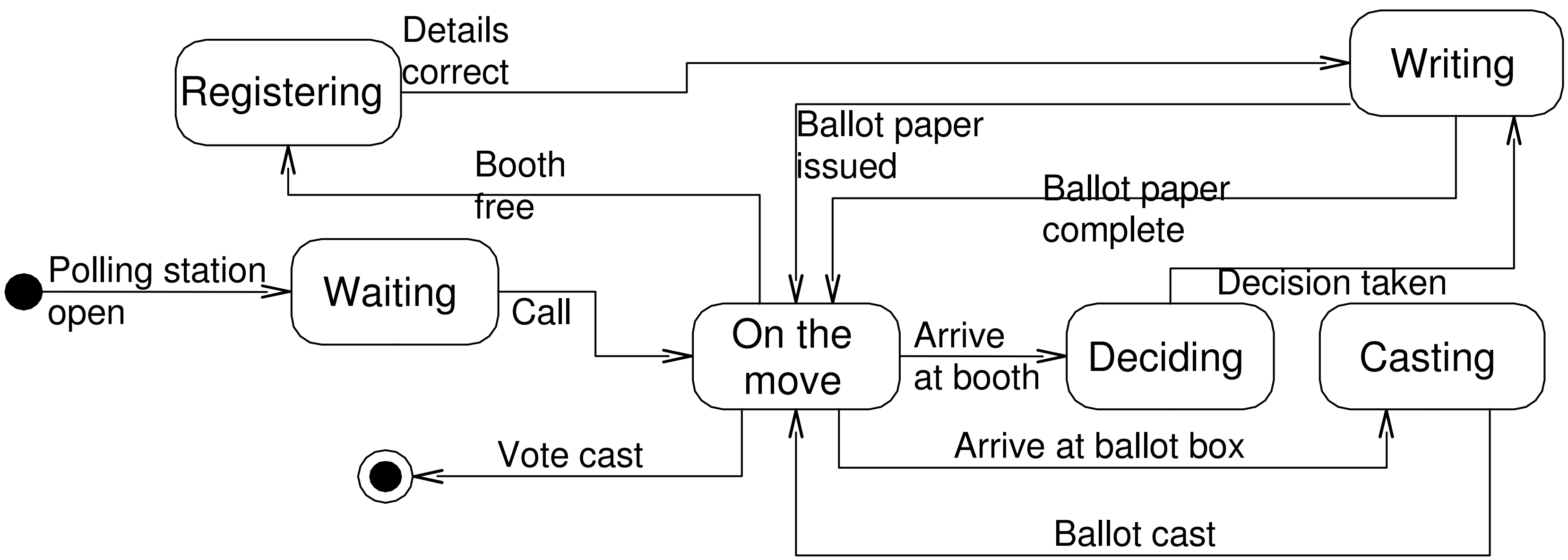
Package the UCs of a HL UCD modelling the functionality of an Internet Banking system. The system may be real or hypothetical. Include dependency relationships in your diagram.

UML State Diagrams

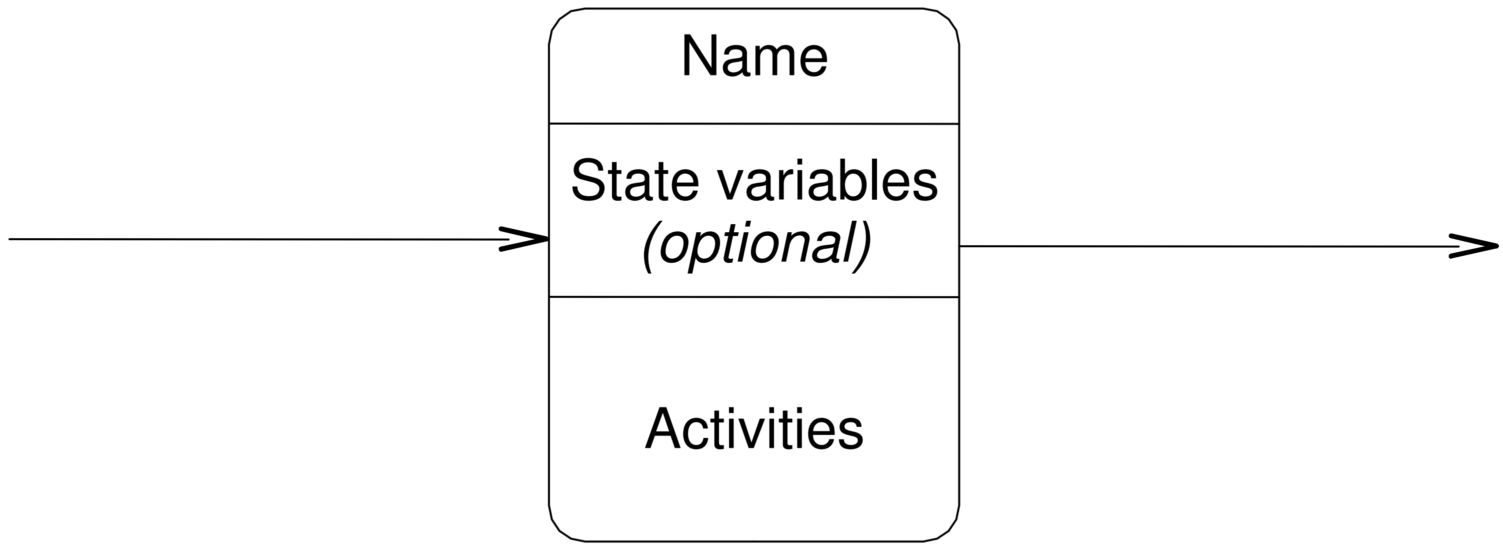
- Usually associated to classes and define their behaviour according to the current state of their objects and affecting events.
- Events are taken to be messages, condition flags, errors or the passage of time.
- UML state diagrams contain at least one starting point ● and one end point ◉ . However the latter can be more than one.
- States can have internal variables and activities associated with them. This information can be hidden to reduce diagram complexity.

Examples of UML State Diagrams





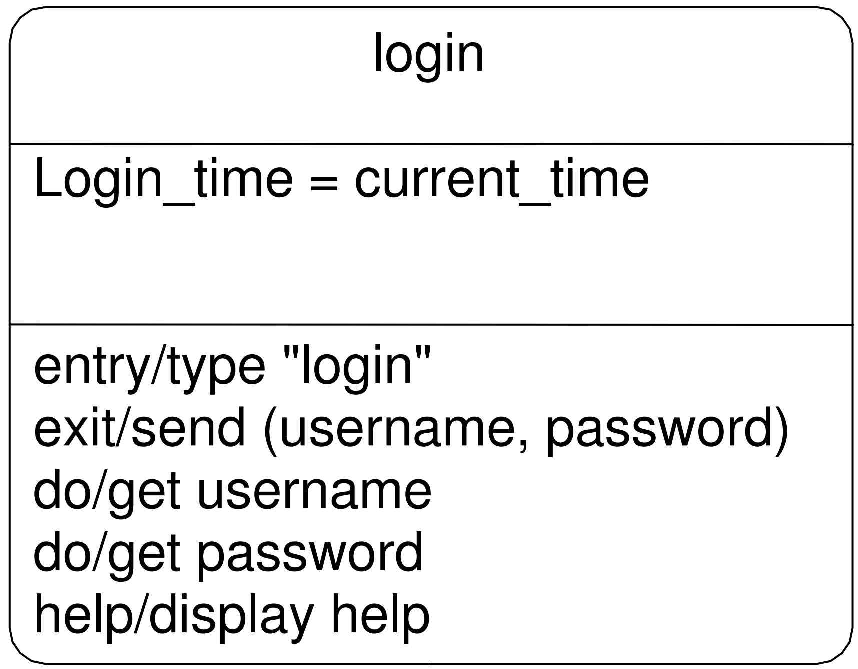
“Compartments” of a UML State



Standard Events in “Activity Compartment”

- Entry
  - Specify actions to be taken on *entry* into the given state (e.g. Assignment to an attribute or sending a message, etc.)
- Exit
  - Specify actions to be taken on *exit* from the given state (e.g. Run housekeeping program or send message informing of termination, etc.)
- Do
  - Specify actions to be taken *while* in the given state (e.g. Processing data, polling, etc.)

Example of Activity Syntax

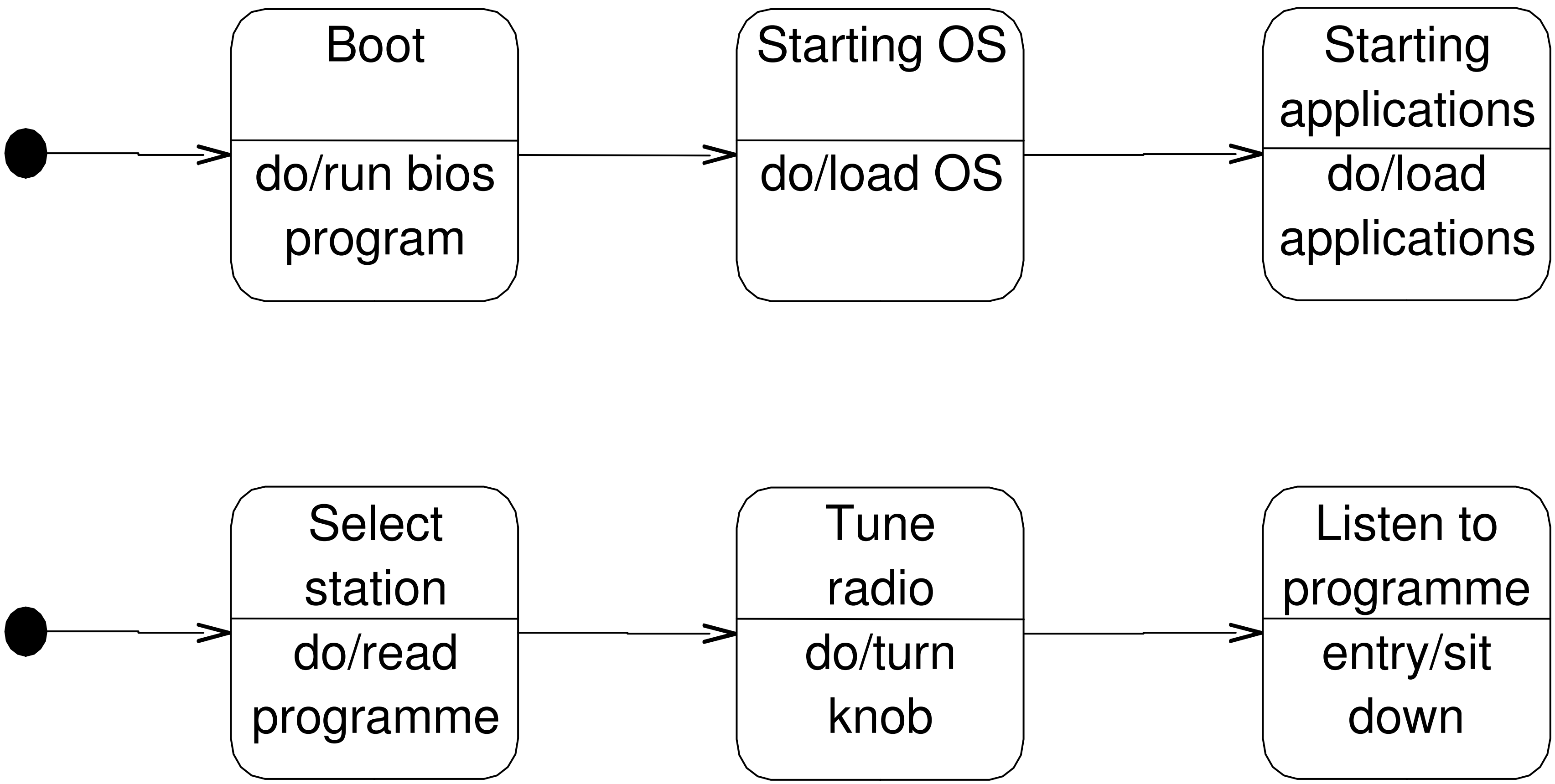


Note, that actions while in a state, are an on-going process, while events associated with a transition are stimuli to the triggering of that transition. Therefore, it is possible that events on transitions act as interrupts to on-going processes in any given state.

Auto-Triggering of Transitions

- Events <u>can</u> be associated to UML transitions
- UML transitions can be left "event-less"
- If "event-less", then transition triggering in UML becomes dependent on internal state actions
- An "event-less" UML transition will auto-trigger when <u>all</u> its associated internal actions get executed.

Auto-Triggering Examples



UML State Transitions

Full transition syntax is as follows:

`event-signature'['guard-condition']/'action-expression'^'send-clause`

`event-name'('parameter','...')'`

`parameter-name':'type-expression,...`

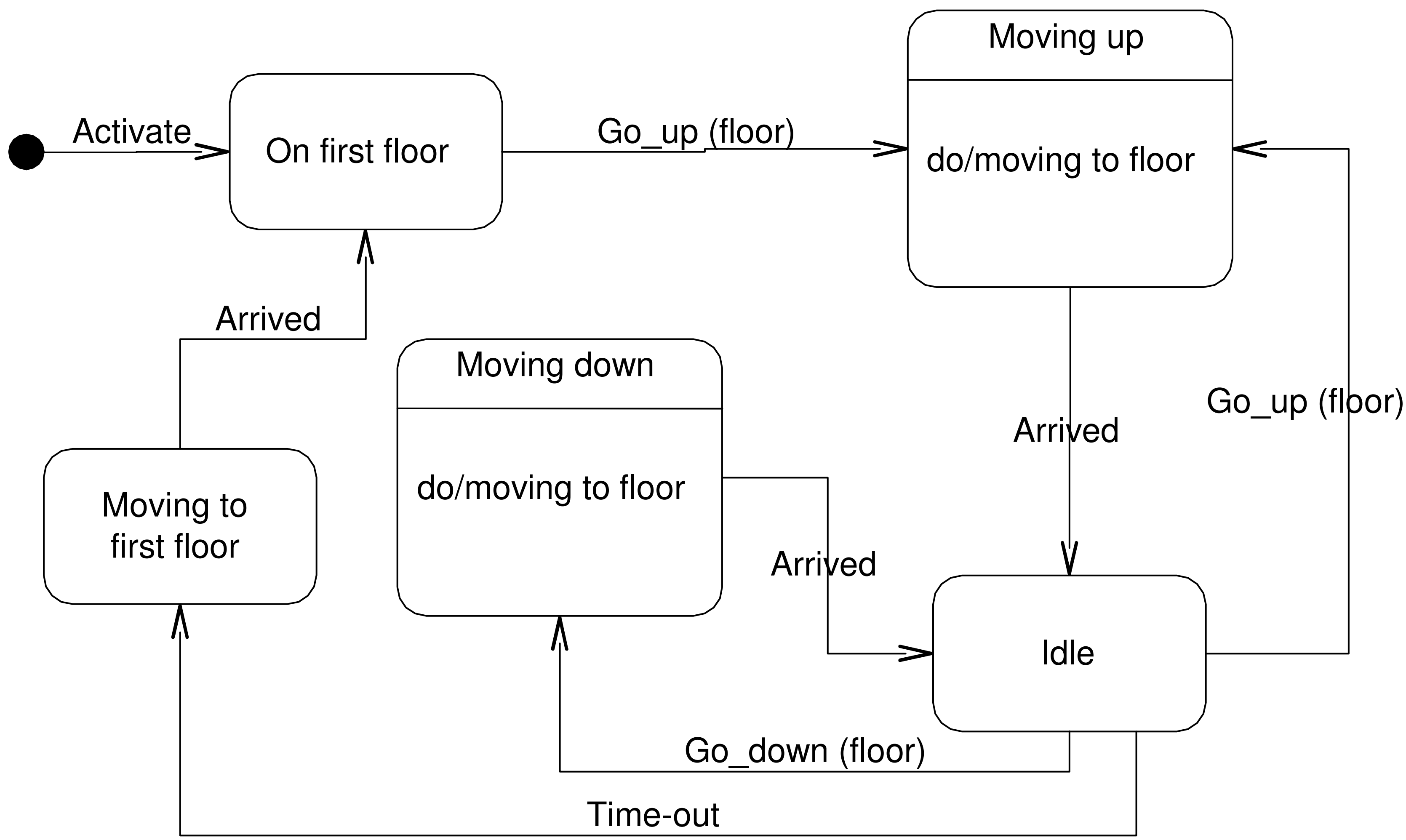
`destination-expression'.'destination-event-name'('argument','...')'`

The "destination-expression" evaluates to an object or a set of obje

UML Event-Signature Examples

```
Print_request(pupu:string)

draw(f:figure, c:colour)

redraw

message_received

go_up(floor)

ready

reach_end(section:text_part)
```

Event-Signature Usage Example



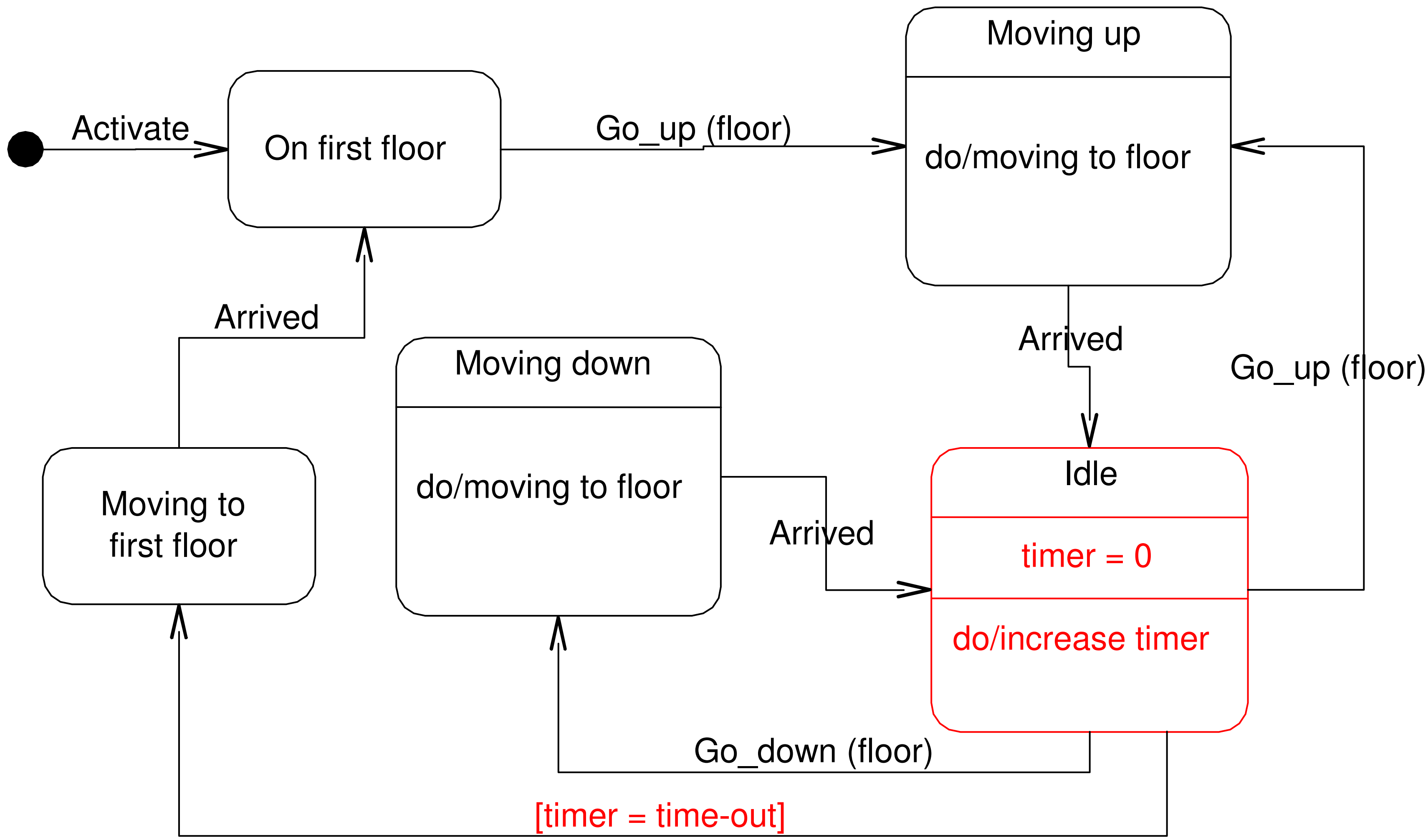
UML Guard-Conditions

- Is a Boolean expression
- Is associated with a UML transition
- Is 'AND-ed' with the event-signature (if present)
- Control whether a given transition will trigger or not

Examples include:

```
[t = 15s]
[retries > 3]
withdrawal(amount) [balance >= amount]
read (buff:char) [buff_elements > 0]
```

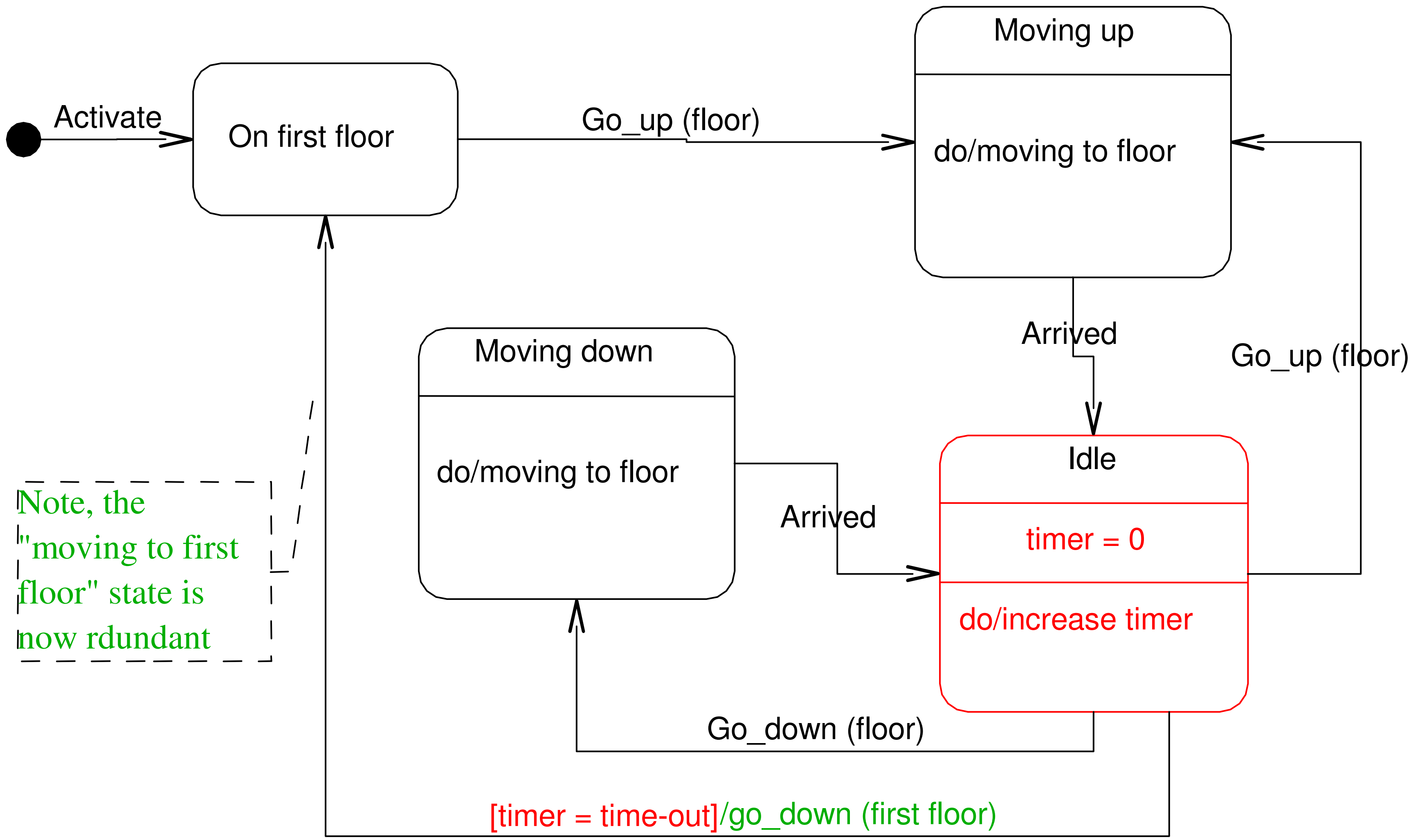
Guard-Condition Usage Example



UML Action-Expressions

- Action-expressions execute when a transition happens
- Not to be confused with internal activities in the activity compartment of a UML state
- Must use parameters existing within the object being modelled by the given state diagram or parameters existing within the associated event-signature
- Zero, one or more action-expressions can be listed per transition using the "/" symbol as delimiter. In the case of more than one, execution is from left to right.

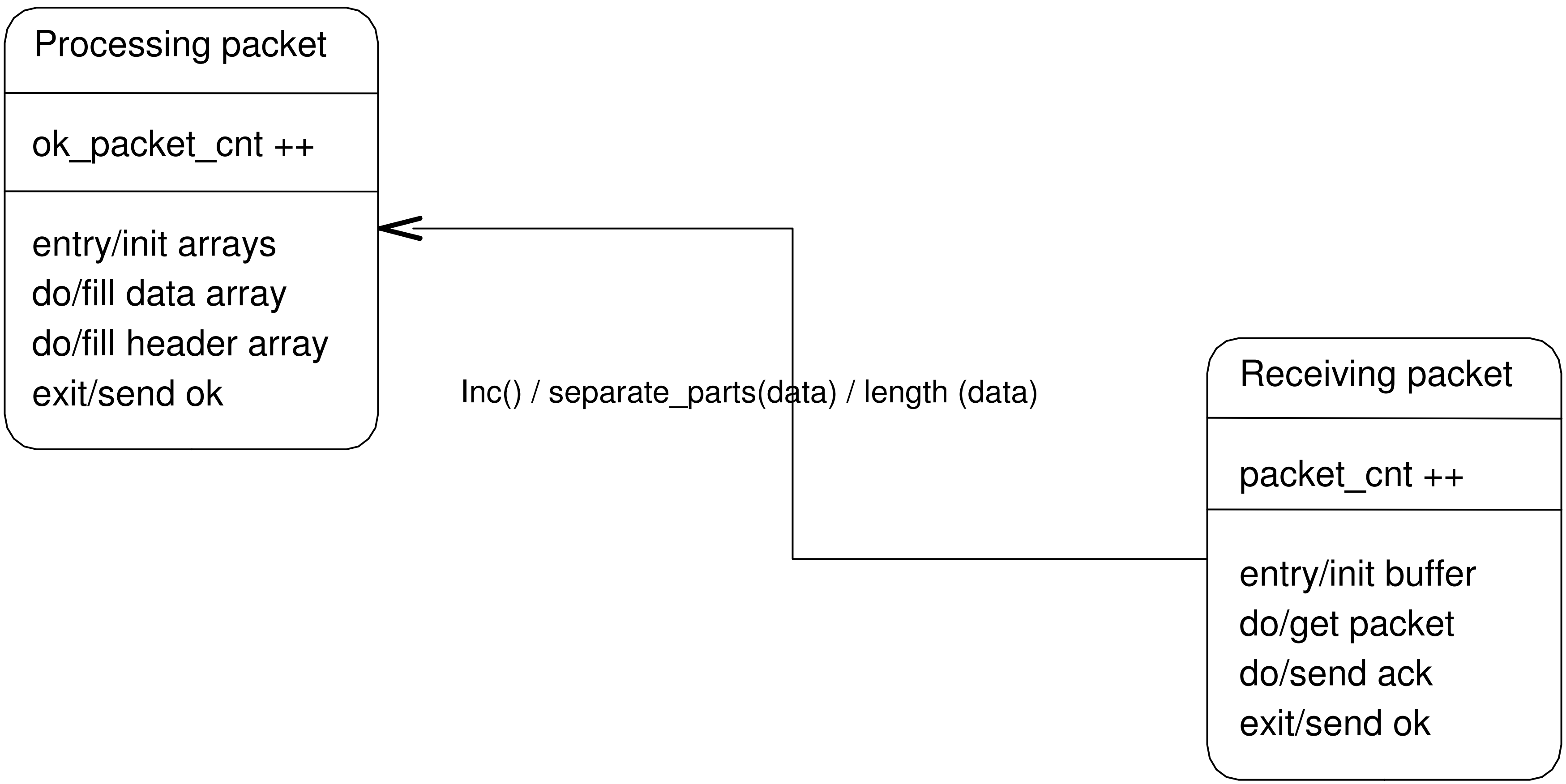
Action-Expression Usage Example



Multiple Action-Expressions

- Must all appear on the transition arc
- Must be separated by the "/" character
- Are taken to execute in sequence from left to right
- Transition containing **only** action-expressions are possible (i.e. With no event-signatures and guard-conditions)
- Nested action-expressions are not allowed
- Recursive action-expressions are not allowed

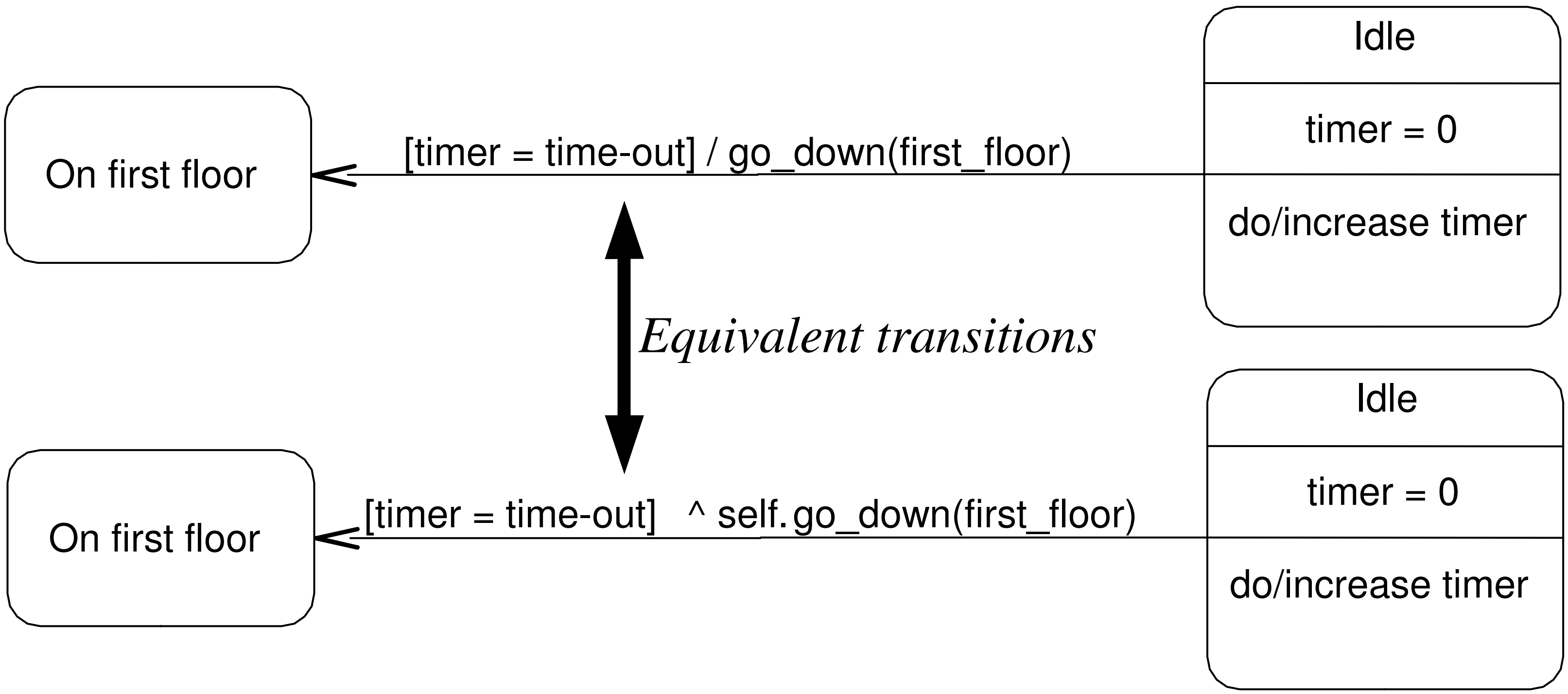
Examples of Multiple Action-Expressions



The Send-Clause

- Also a form of action(-expression)
- Explicitly designed syntax to show message passing
- The destination of the message could be an object or a set of objects
- The destination object can be the object described by the state-diagram itself

Send-Clause Examples



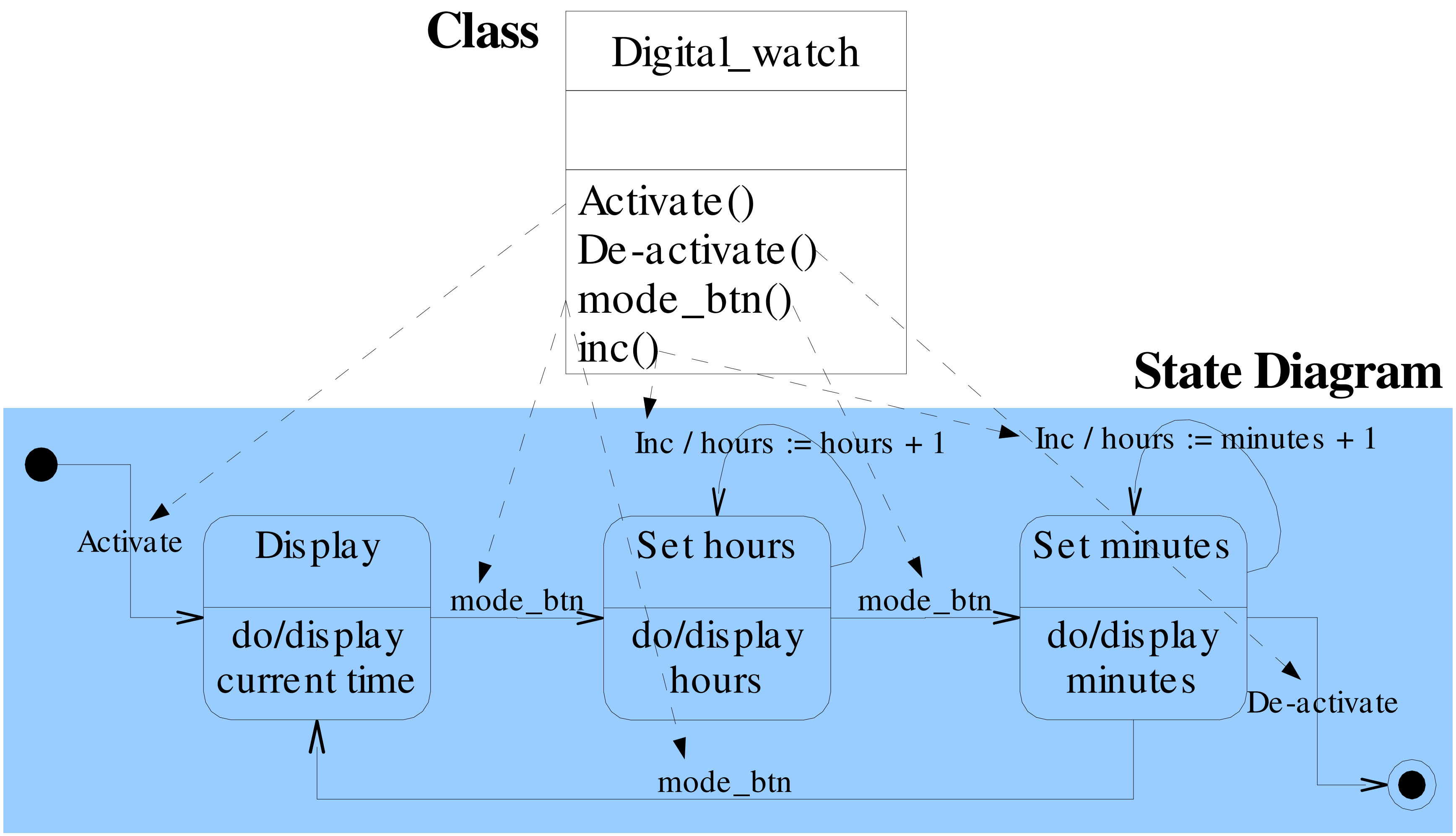
Other examples:
out_of_paper() ^indicator.light()
request_withdrawal(amount) / show_amount() ^account.debit(amount)
left_mouse_btn_down(location) / colour := pick_colour(location)
^pen.set(colour)

UML Events

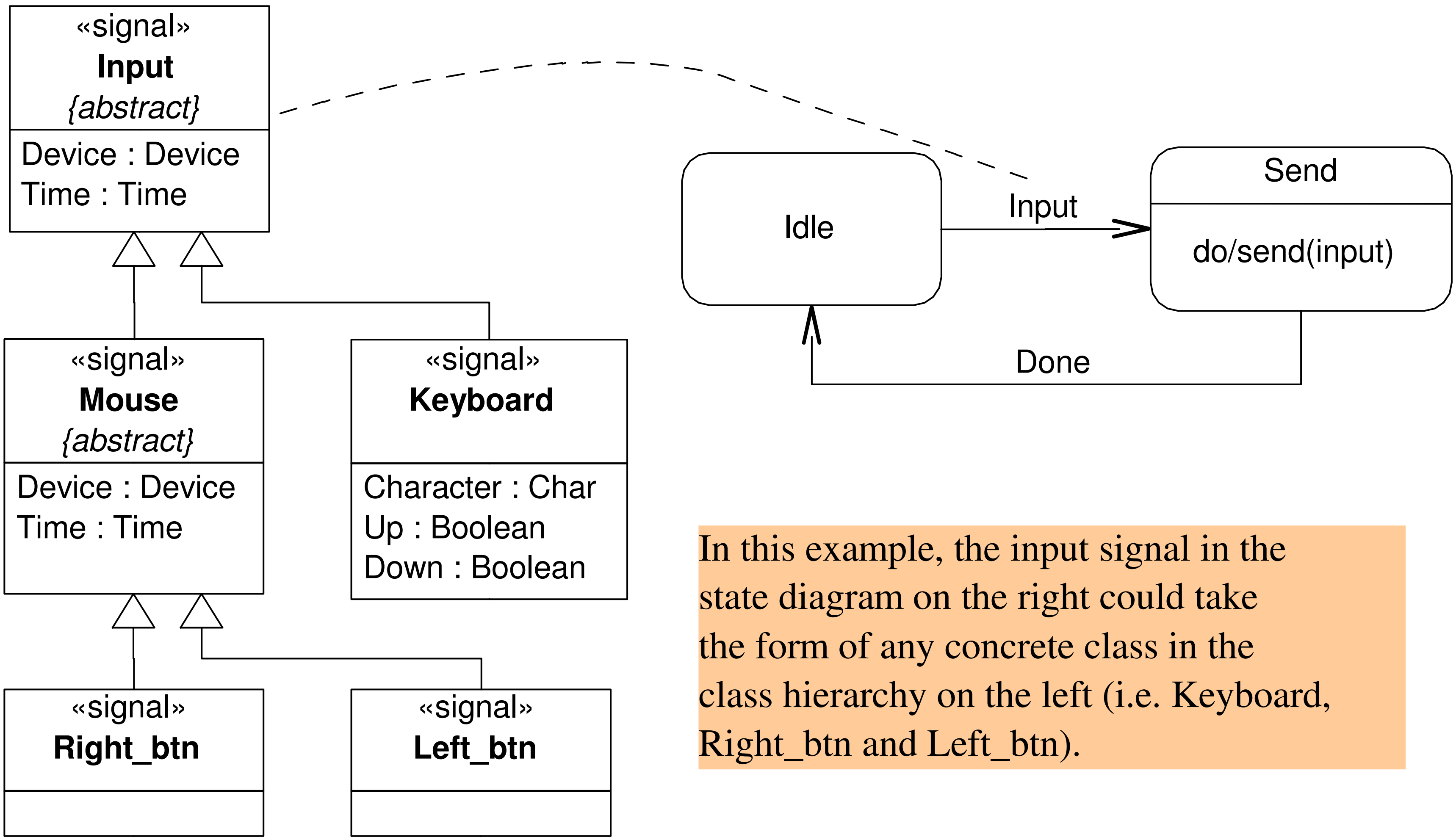
UML defines four categories of events:

- *A condition becoming true* (i.e. A Boolean condition and shown as guard-condition)
- *Receipt of an explicit signal, itself an object, from another object* (i.e. A message and shown as an event-signature)
- *Receipt <u>of a call</u> on an operation by another object (or by the object itself)* (i.e. Also a form of message and also shown as an event-signature)
- *Passage of a designated period of time* (i.e. Time calculation and shown as a time-expression)

Relationship of Events to Class Operations

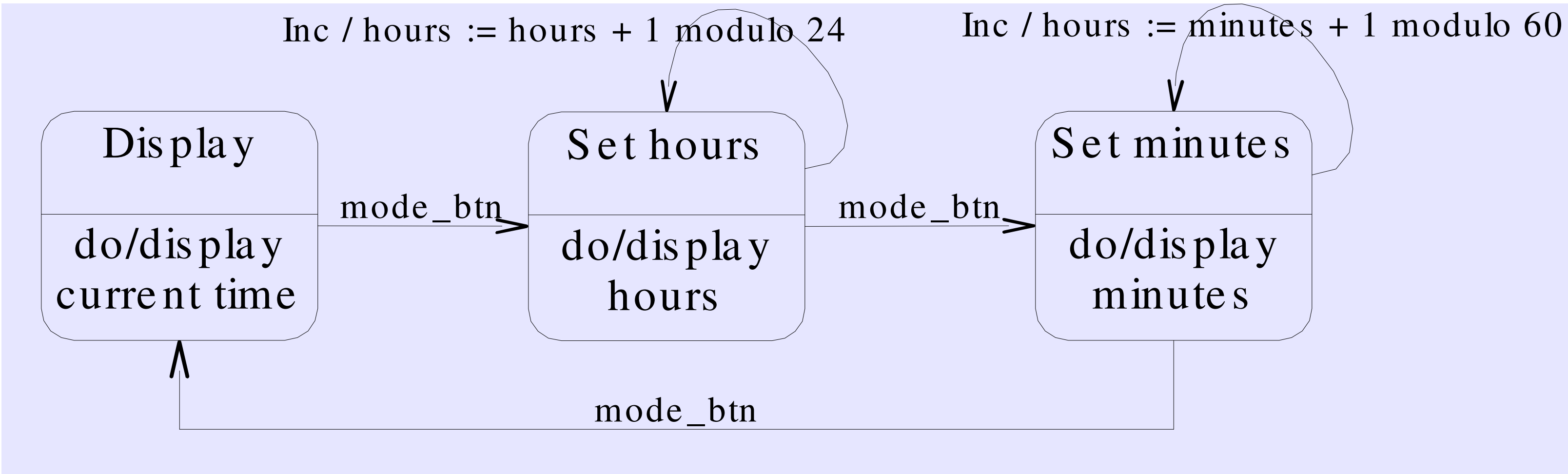


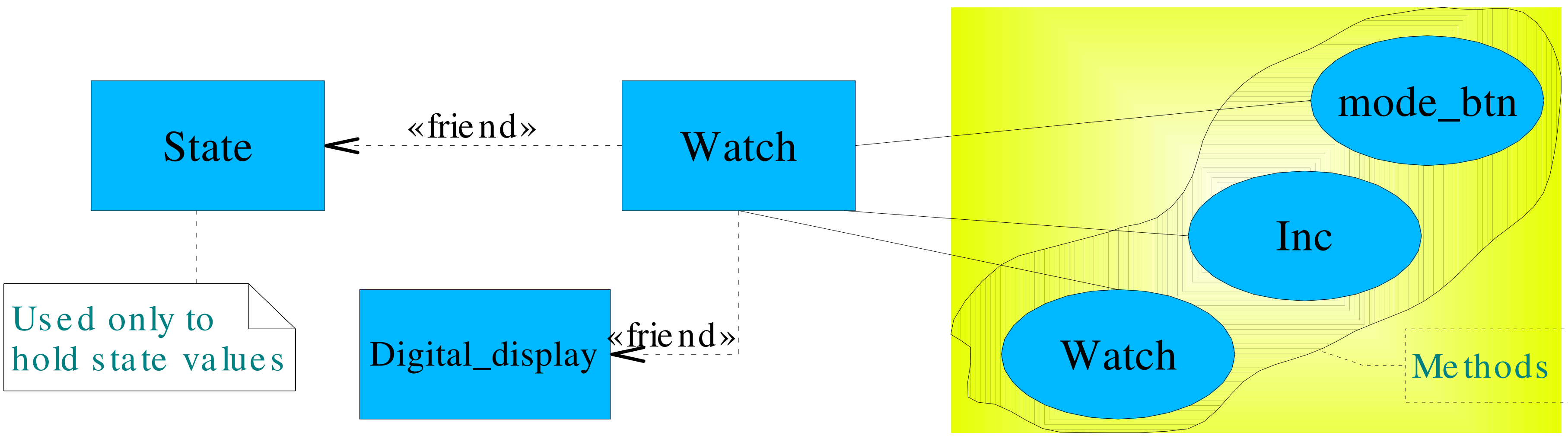
Example of Signal Class Structuring and Polymorphism



In this example, the input signal in the state diagram on the right could take the form of any concrete class in the class hierarchy on the left (i.e. Keyboard, Right_btn and Left_btn).

Java implementation of a UML State Diagram (1/3)





Java implementation of a UML State Diagram (2/3)

```
Public class State
{
  public final int Display = 1;
  public final int Set_hours = 2;
  public final int Set_minutes = 3;
  public int value;
}
public class Watch
{
  private State state = new State();
  private Digital_display LCD = new Digital_display();

  public Watch()
  {
    state.value = State.Display;
    LCD.display_time();
  }
  public void mode_btn()
  {
    // Cycle through actions depending on current state (see next slide)
  }
  public void inc()
  {
    // Update corresponding LCD segments (see next slide)
  }
```

Please note, that the class Digital_display is omitted to keep the example's code shorter and more to the point.

Java implementation of a UML State Diagram (3/3)

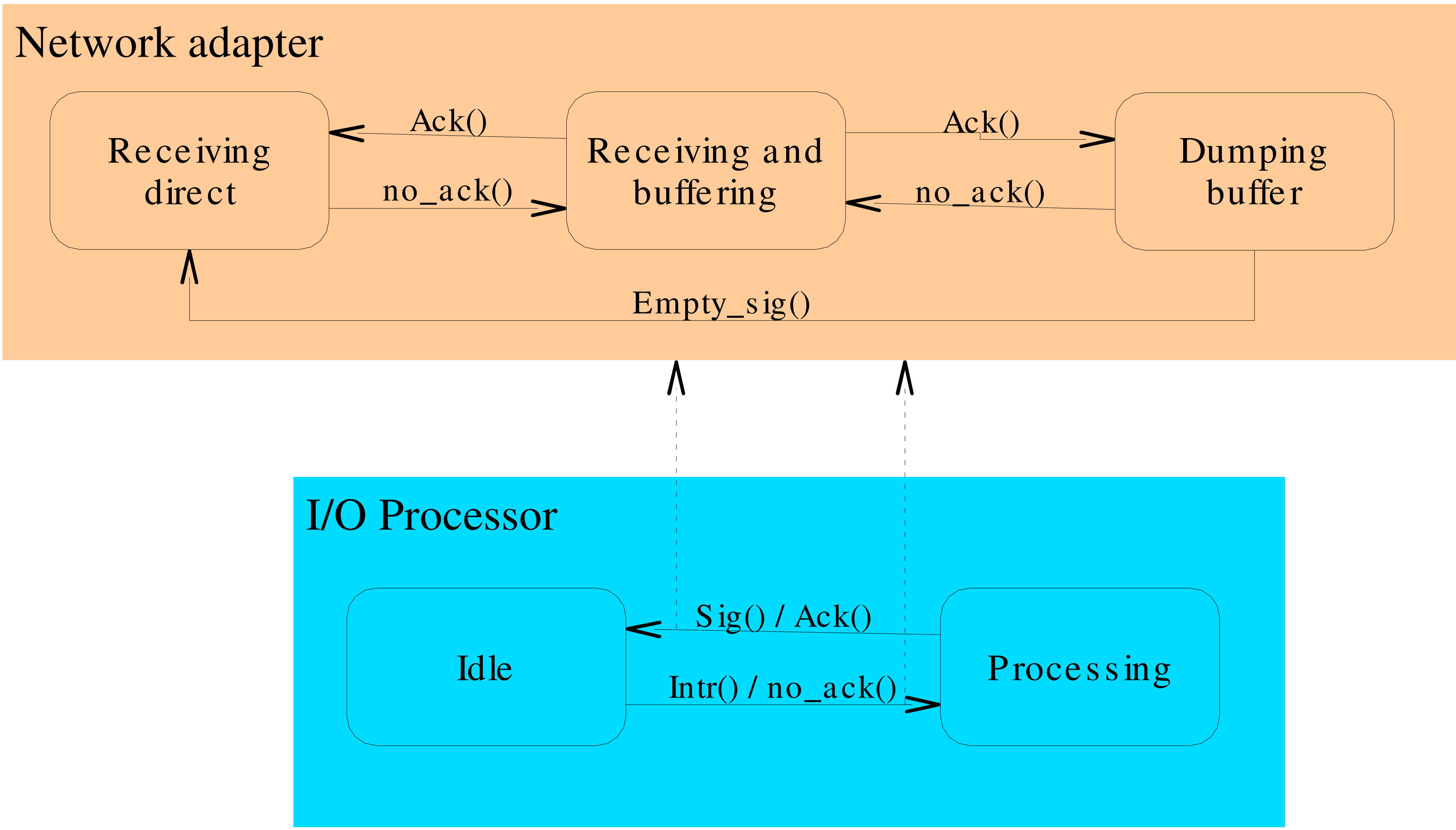
```
// Cycle through actions depending on
// current state (from previous slide)
Public void mode_btn()
{
  switch (state.value)
  {
    case State.Display :
      LCD.display_time();
      state.value = State.Set_hours;
      break;
    case State.Set_hours :
      LCD.display_hours();
      state.value = State.Set_minutes;
      break;
    case State.minutes :
      LCD.display_minutes();
      state.value = State.Display;
      break;
  }
}
```

```
// Update corresponding LCD segments
// (from previous slide)
Public void inc()
{
  switch (state.value)
  {
    case State.Display :
      ;
      break;
    case State.Set_hours :
      LCD.inc_hours();
      break;
    case State.minutes :
      LCD.inc_minutes();
      break;
  }
}
```

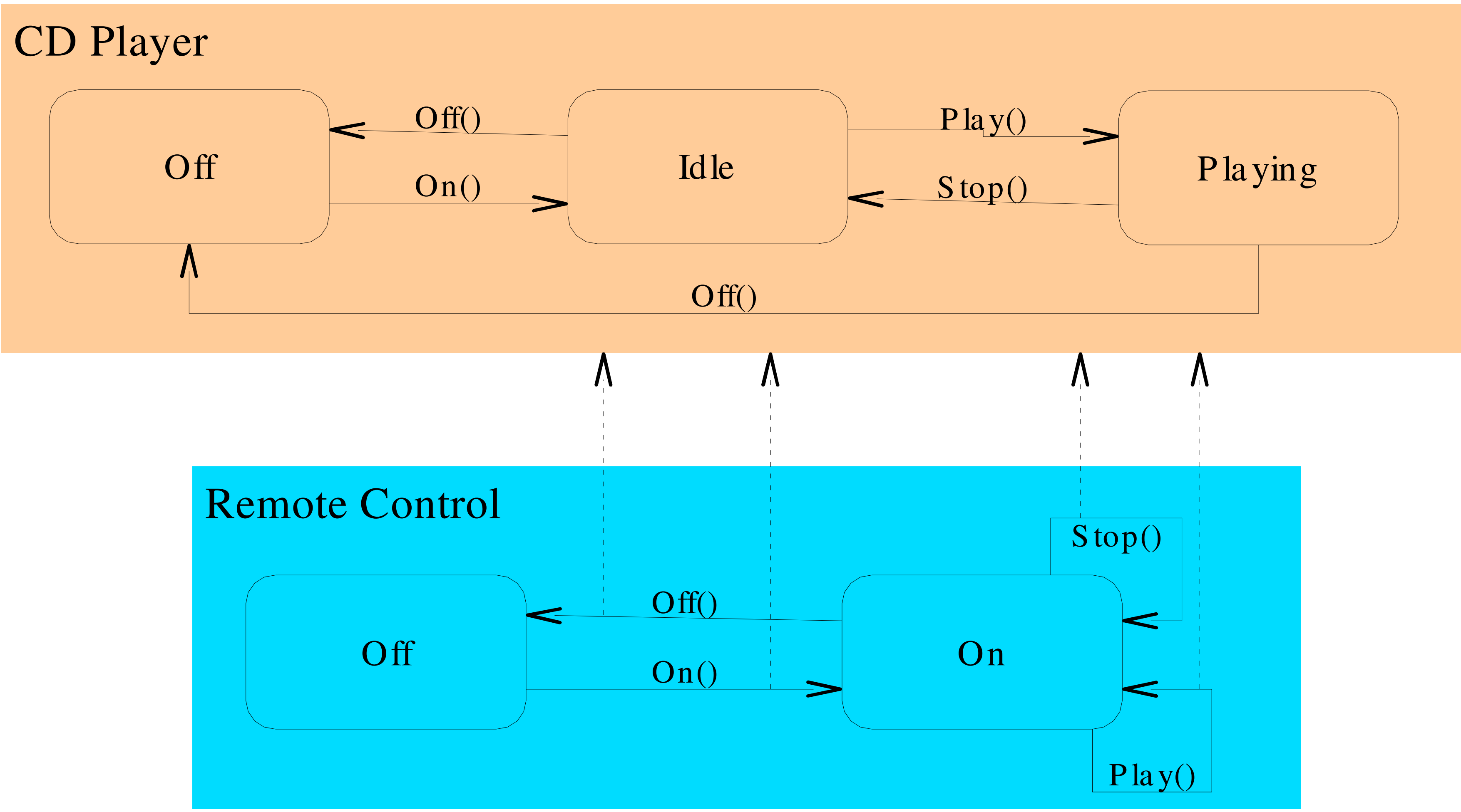
Messaging between UML State Diagrams

- Used to communicate operations or messages between state diagrams
- Can be implemented by action-expressions (as described earlier) or by dashed arrows
- The Dashed arrows can originate from either a specific state diagram transition or from the state diagram as a whole
- The target state diagram MUST contain the appropriate event-signatures to "catch" any sent messages

Examples of UML State Diagram
Messaging (1/2)

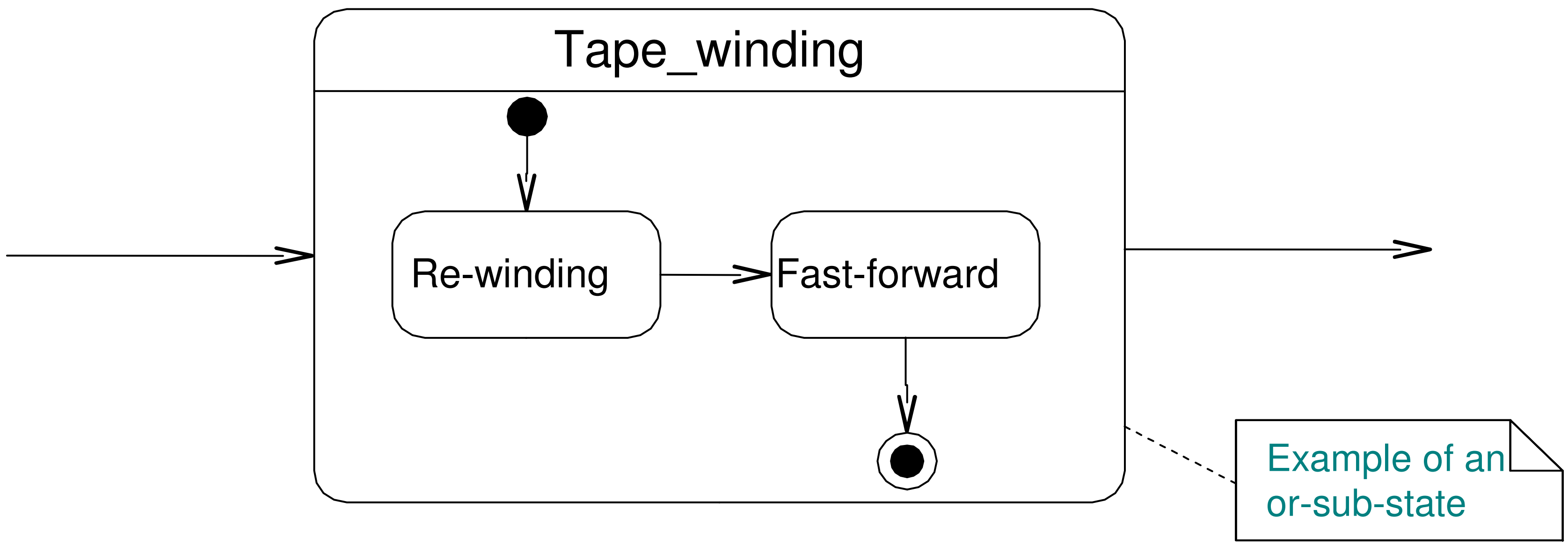


Examples of UML State Diagram Messaging (2/2)

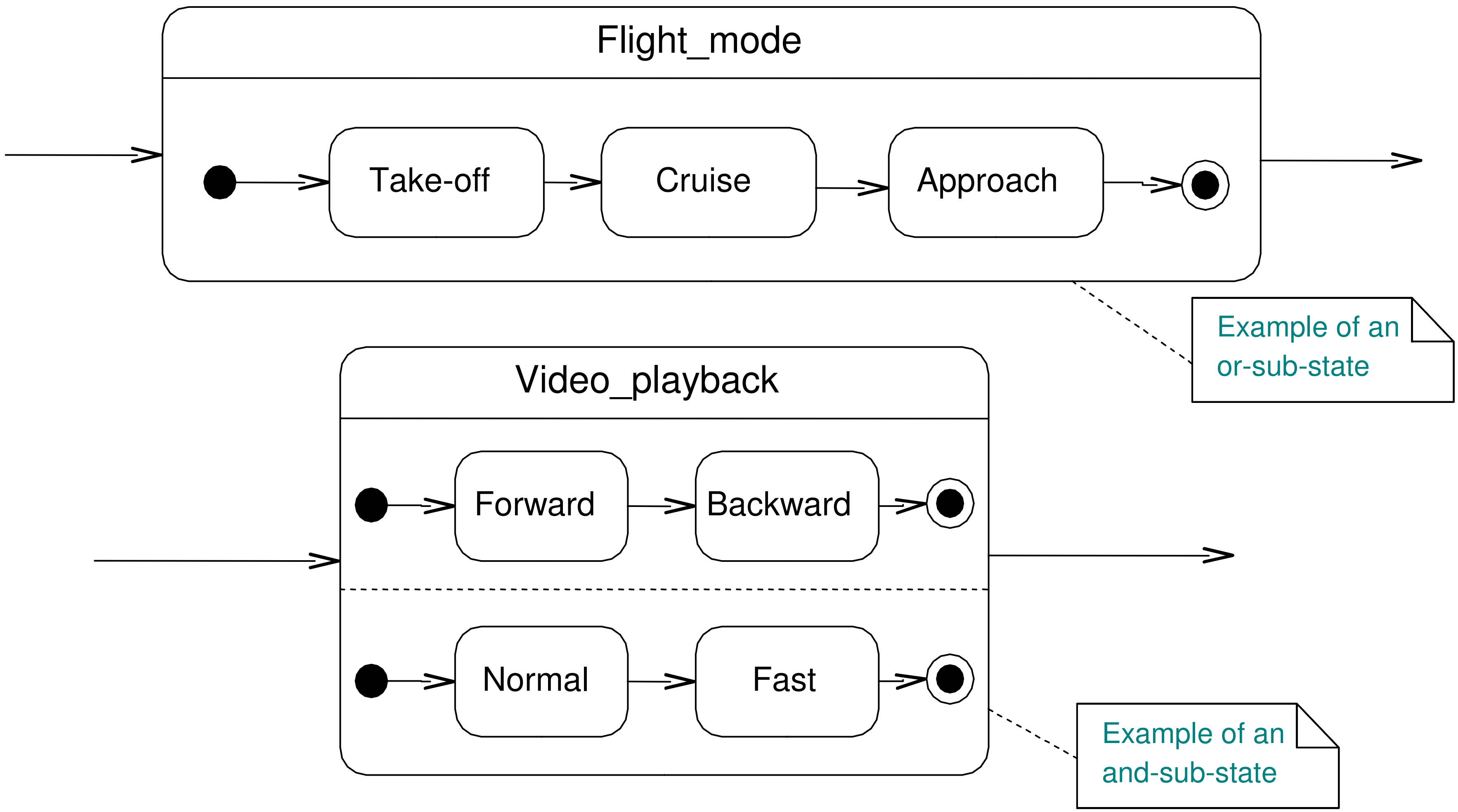


UML Sub-States

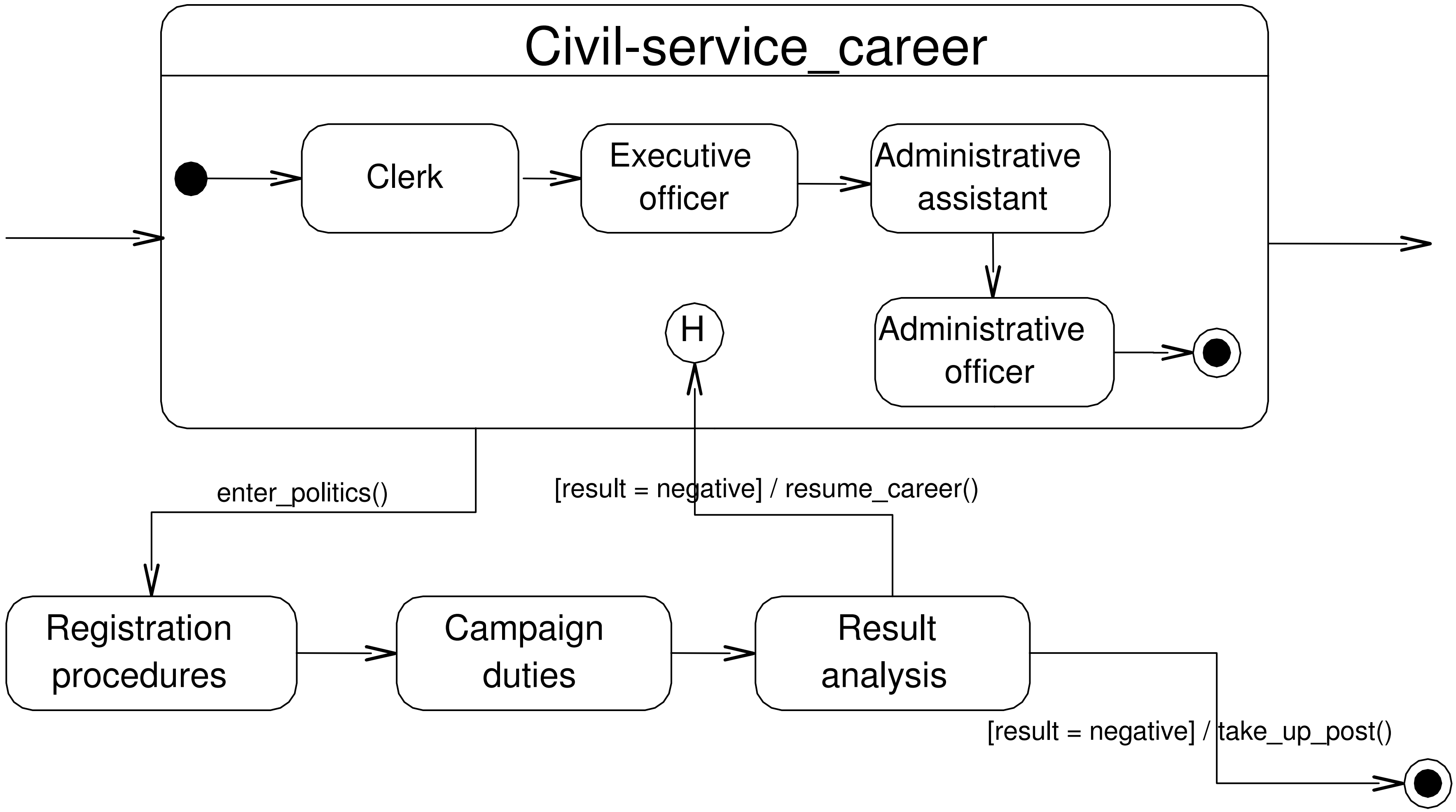
- A mechanism for nesting states
- Take the form of "and-sub-states" or "or-sub-states



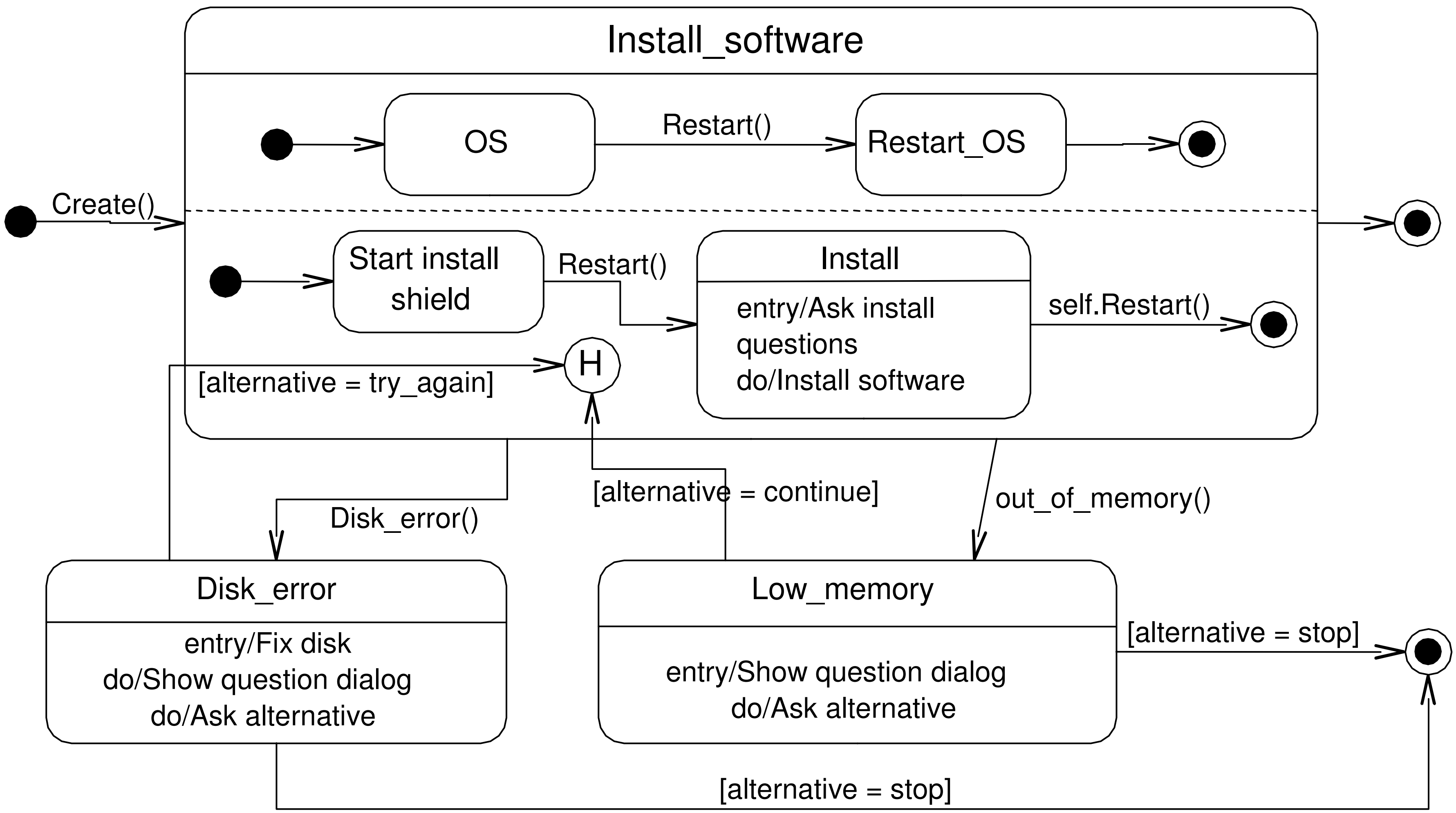
Examples of "or" and "and" sub-states



Usage Example of the UML History Indicator (1/2)



Usage Example of the UML History Indicator (2/2)



Workshop Activity -14-

A class named “campaign” is textually described as follows:

“Once a campaign is established, it is assigned a manager and staff. Authorisation in the form of a signed contract and an authorisation code is required to kick-off an established campaign. Once a campaign is started it is noted as active. On completion of an active campaign, accountability is carried out in the form of preparation of final statements. Once payment is received in full, a campaign is considered paid, is archived and any assigned personnel is released. If payment is only effected in part, the campaign is not considered paid but rather simply completed. If any payments were effected in advance of campaign completion and are in excess of the final payment request, a refund should be issued.

Draw a UML state diagram for the above system.

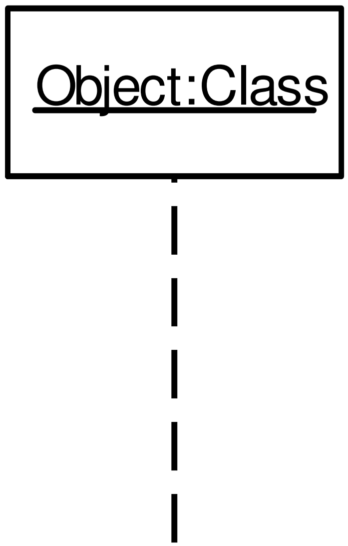
Workshop Activity -15-

One of the states in Workshop Activity -14- will probably be “Active” or “CampaignActive”. Produce a nested state diagram for this state.

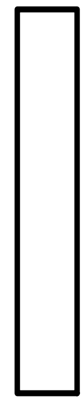
UML Sequence Diagrams

- Used to show object interaction
- Interaction takes the form of messages
- Can only model <u>single-scenario</u> situations
- Message type is one of the interaction types *(i.e. synchronous, simple, etc. – see next slide)*
- Sequence diagrams should be read starting from the top downwards
- Sequence diagrams highlight control focus in objects *(i.e. object activation)*
- Sequence diagrams can be either of "instance" or of "generic" form

Components of a UML Sequence Diagram



Object name and “lifeline”

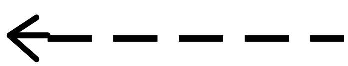


Operation duration (activation)

Call (synchronisation) message

Undefined type (uncommitted) message

Control (acknowledgement) message

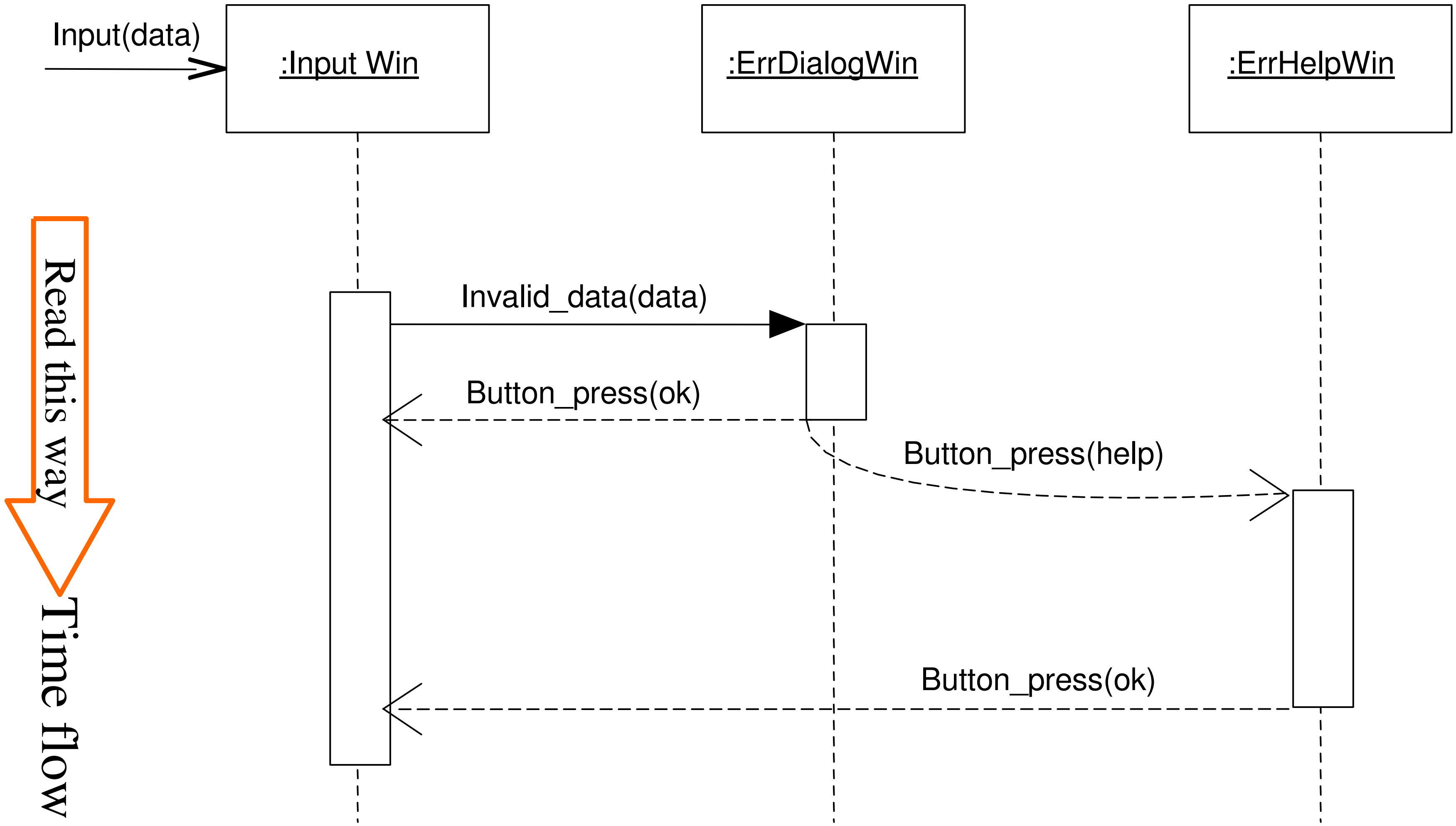


Independent (asynchronous) message

Types of Interaction

- Simple
  - Shows a control message without any particular details (often, but not solely, used as acknowledgement to messages)
- Synchronous
  - Shows an operation call message. Assumes that the called object operation must terminate before the caller can proceed. Could include implicit return.
- Asynchronous
  - Indicates independent process execution. No explicit "return-to-caller" action. Used mainly in real-time concurrent systems.
- Uncommitted
  - Shows a message of undetermined type. Can be either synchronous or asynchronous.
- Synchronous with immediate acknowledgement
  - Indicates a combination of simple and synchronous and indicates that an immediate reply happens. (Strictly, not purely UML notation)

UML Sequence Diagram Example



Guidelines for Building a UML Sequence Diagram

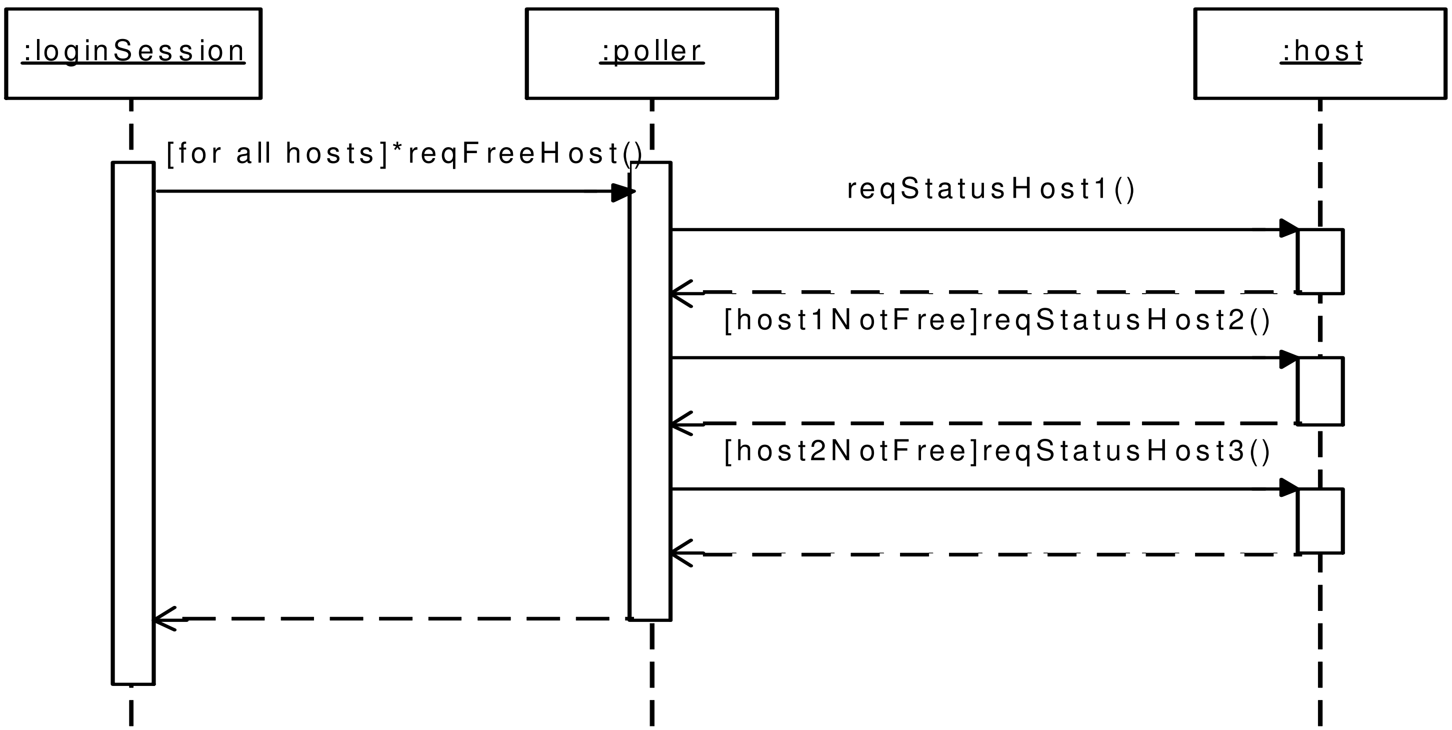
1. Set the context (i.e. scope the system)
2. Identify participating objects
3. Draw arbitrary lifelines for each class
4. Draw the duration of the objects on the class lifeline
5. Insert the object messages from top to bottom of diagram (time-based)
6. Check the diagram for completeness

Iteration Conditions in UML Sequence Diagrams

Iteration condition controlled message syntax:

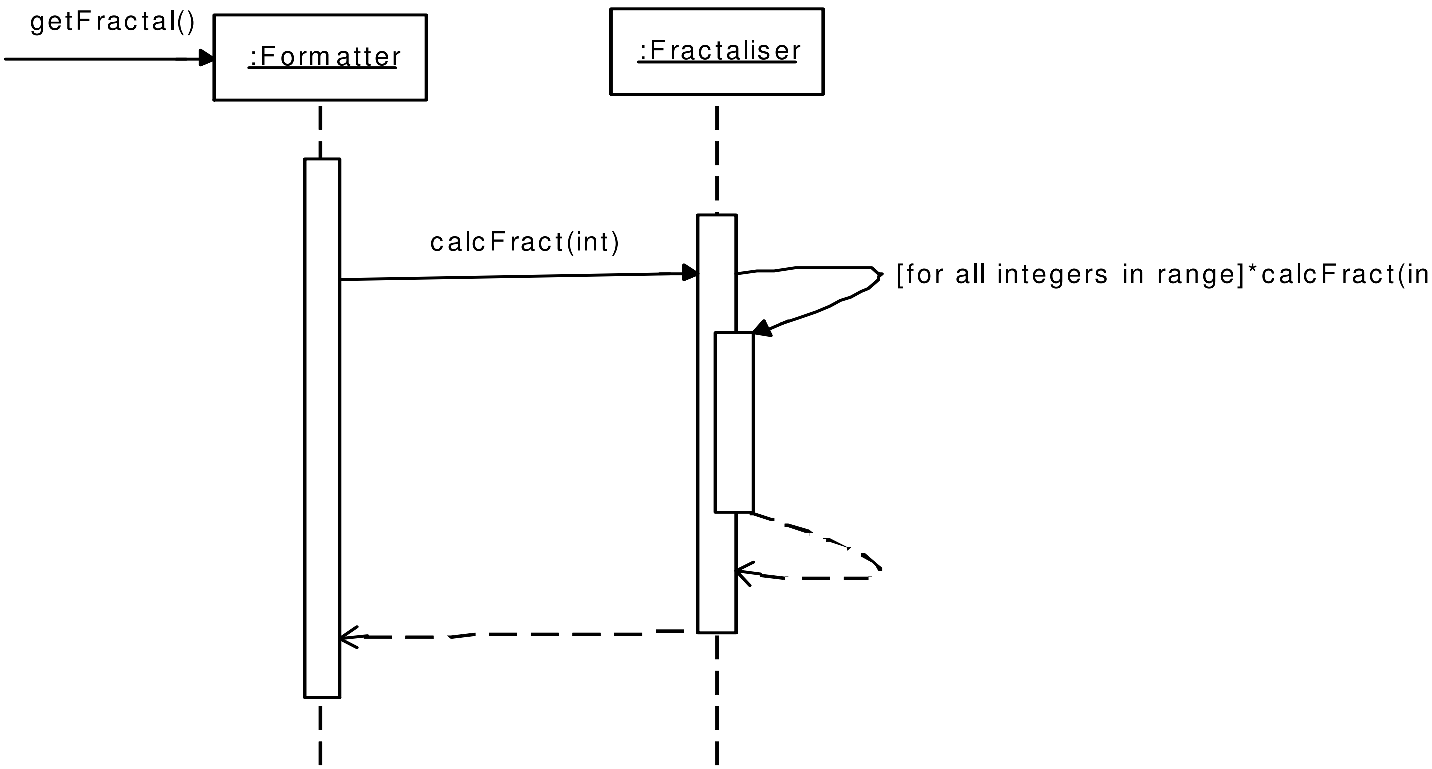
`[continuationCondition]*operation(parameter)`

An example…

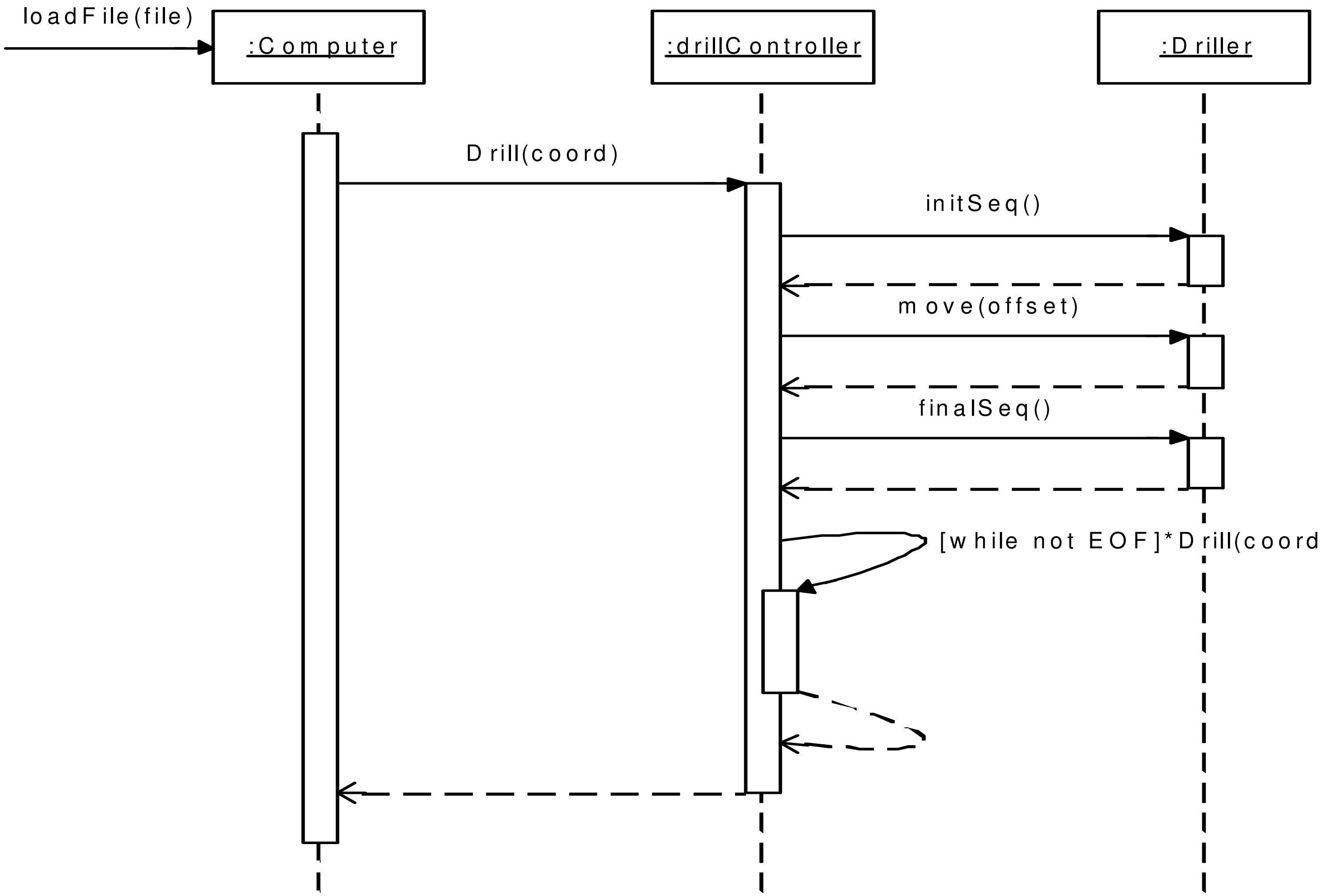


Recursion Modelling in UML Sequence Diagrams

- Recursion is always carried out by call (synchronous) messaging and is represented in UML sequence diagrams as follows:



Recursion Example in UML Sequence Diagrams

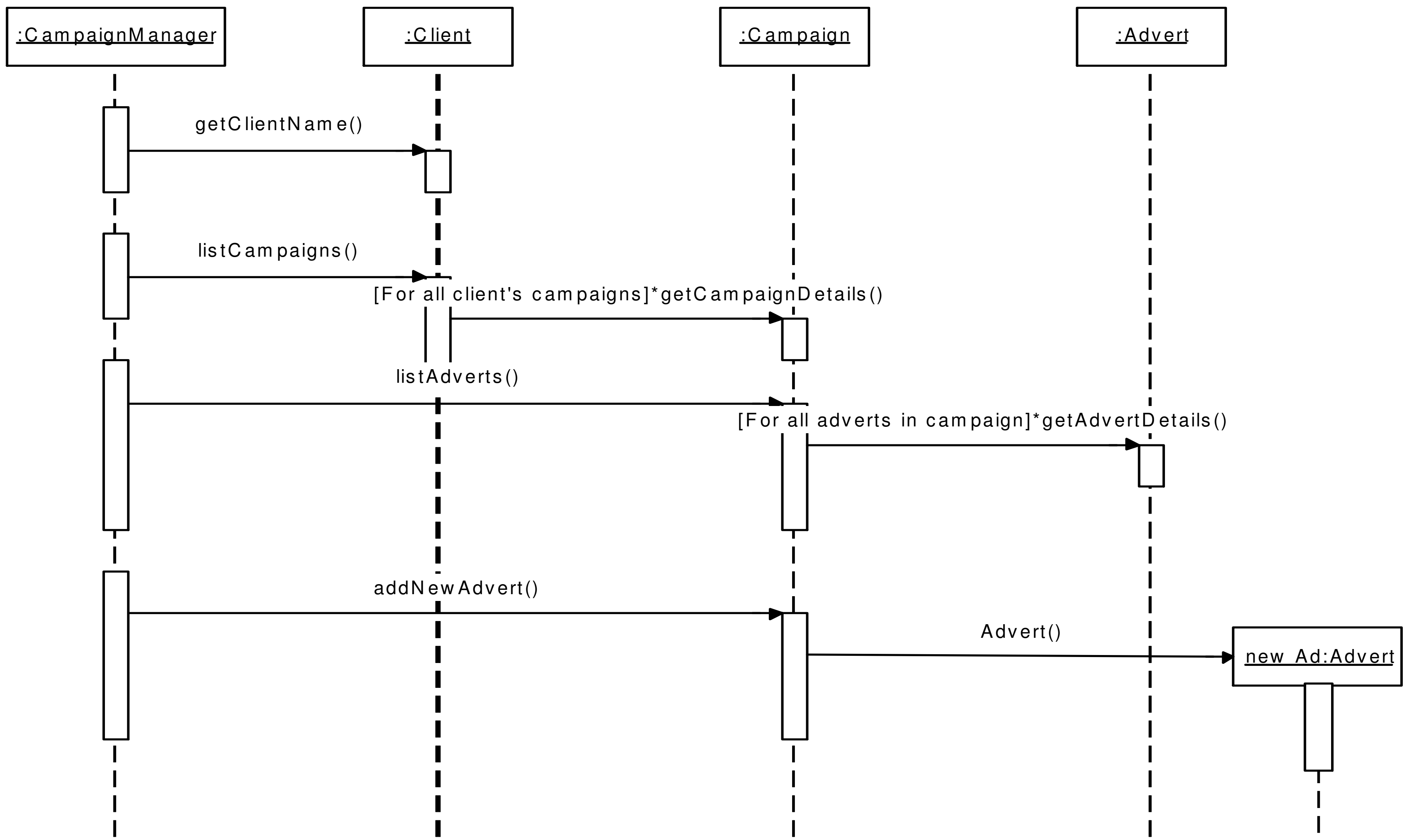


Try this yourself…

Draw up a sequence diagram modelling the case when an advert campaign manager retrieves the details of a particular client's advertising campaign and lists the details of a particular advert from the campaign. The sequence diagram should also show the case when a new advert is created. *Only call messages (synchronous) should be used in this example and use any iteration conditions as you deem necessary.*

Objects to use: "CampainManager", "Client", "Campaign", and "Advert"

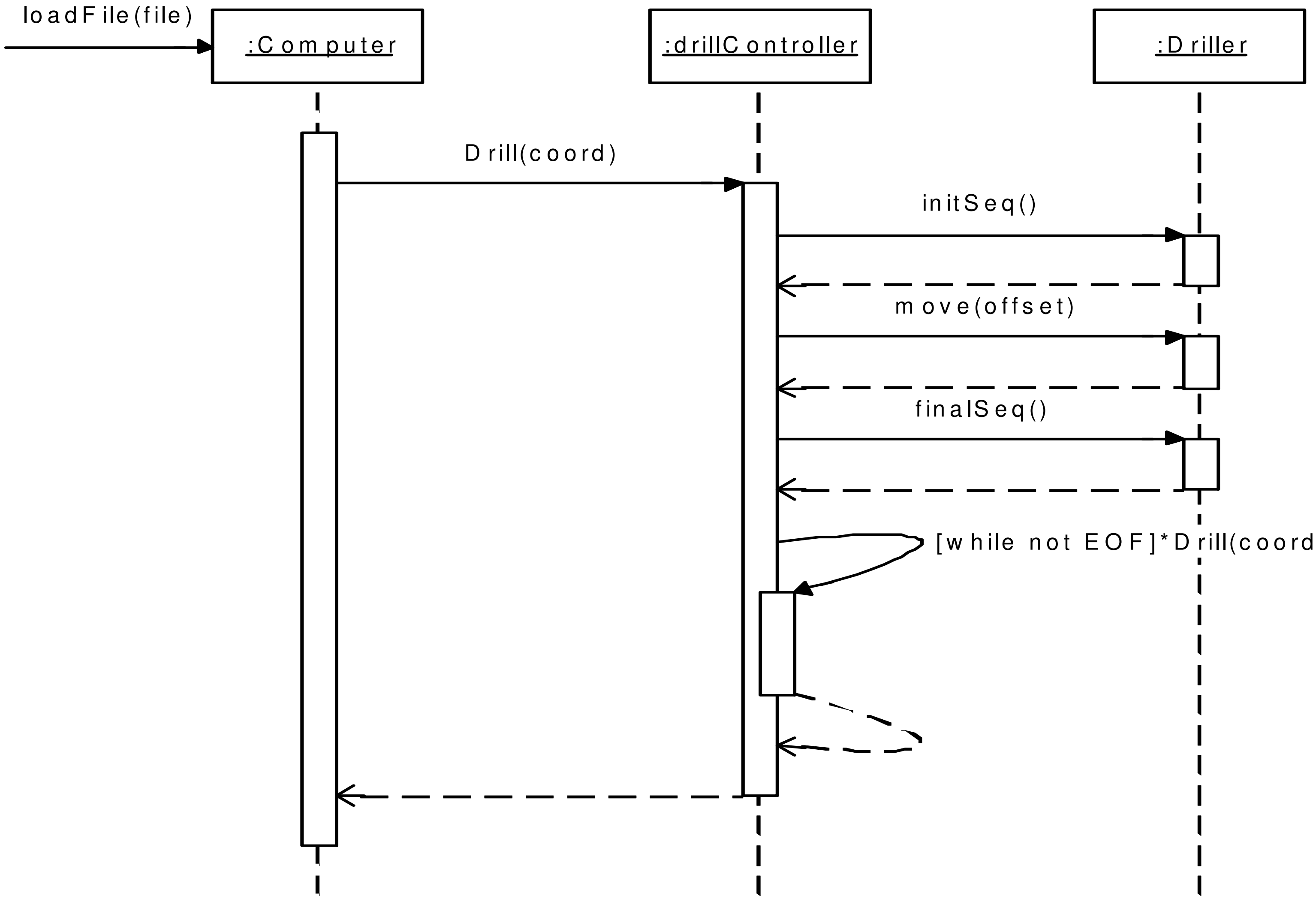
A Solution to Previous Slide



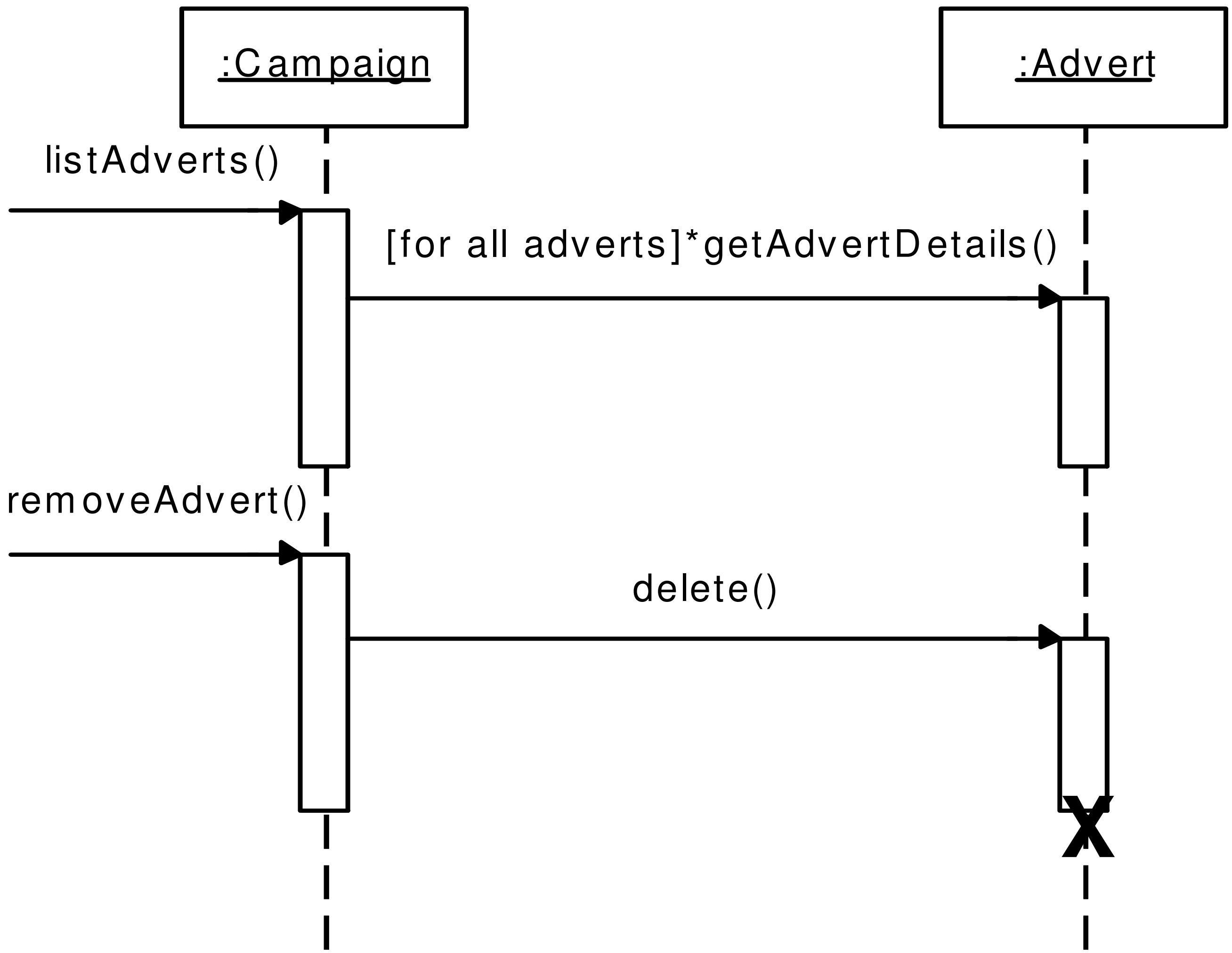
Try this too…

Create a sequence diagram modelling the behaviour of a PCB drilling machine. The machine will drill holes in a PCB of given dimensions at a set of given co-ordinates. Co-ordinates are given as a list, which must contain at least one set of co-ordinates. Drilling stops when the end of the list is reached or when a user interrupts the process.

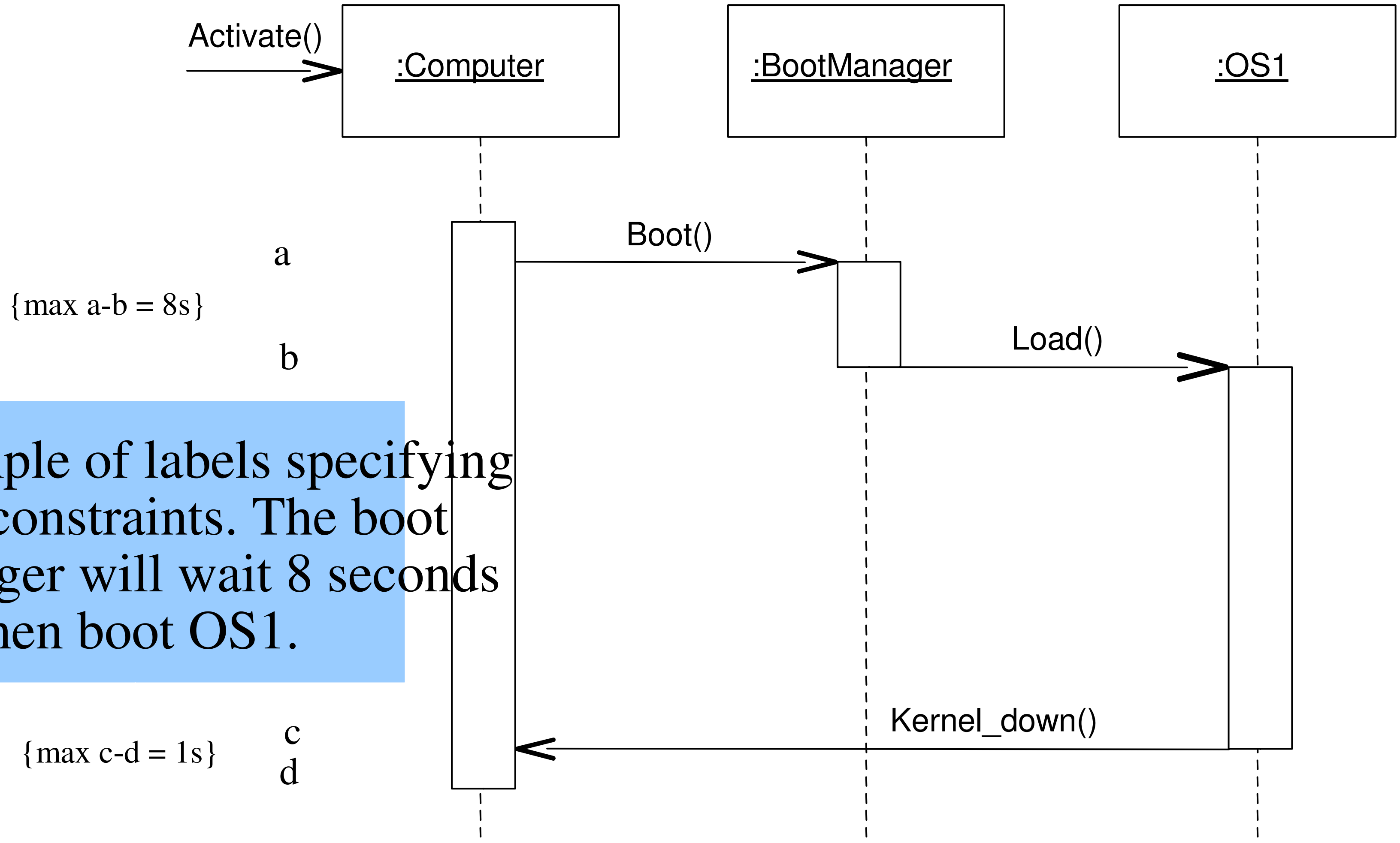
A Solution to Previous Slide



Object Destruction in UML

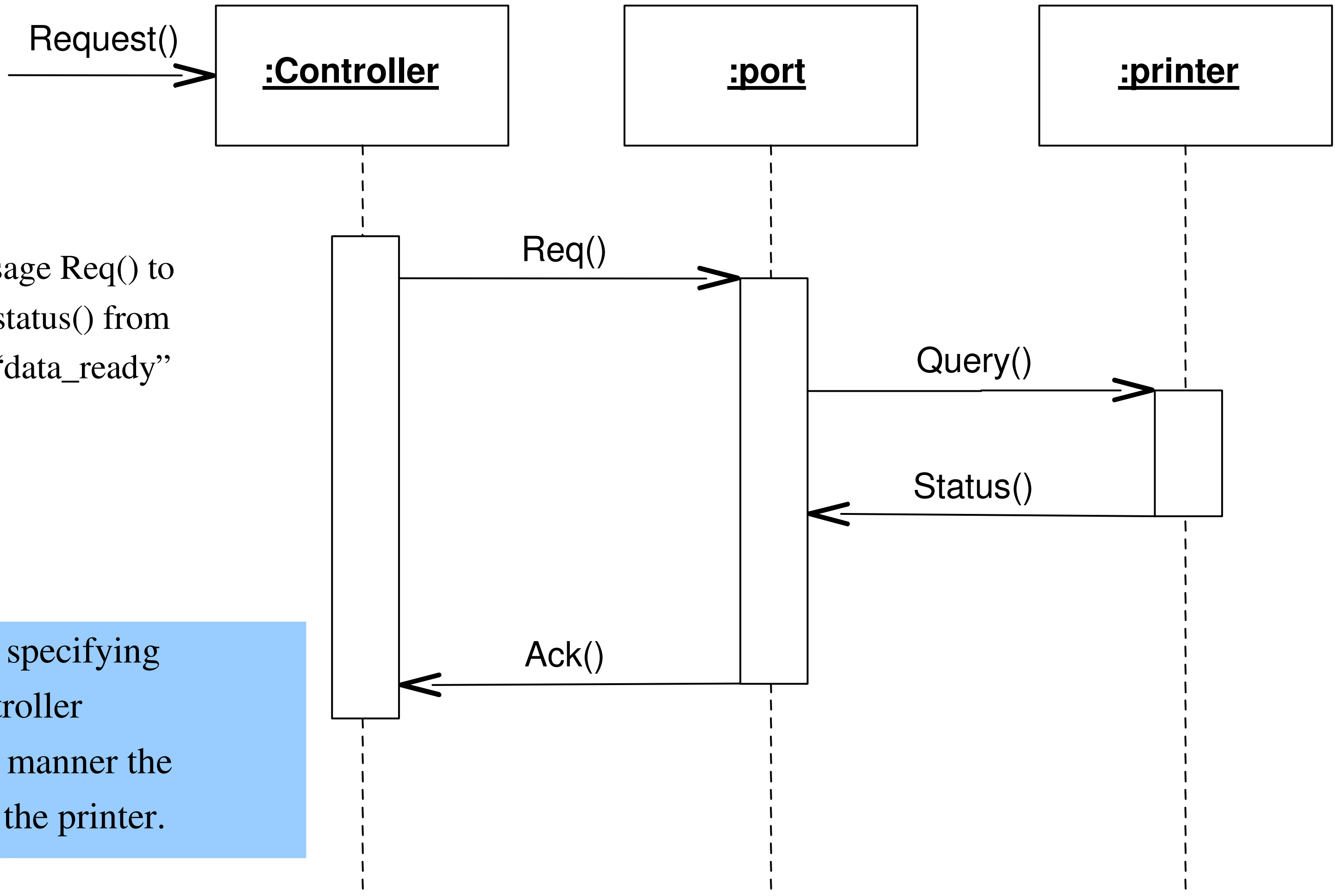


Using Labels in UML Sequence Diagrams (1/2)



Example of labels specifying time constraints. The boot manager will wait 8 seconds and then boot OS1.

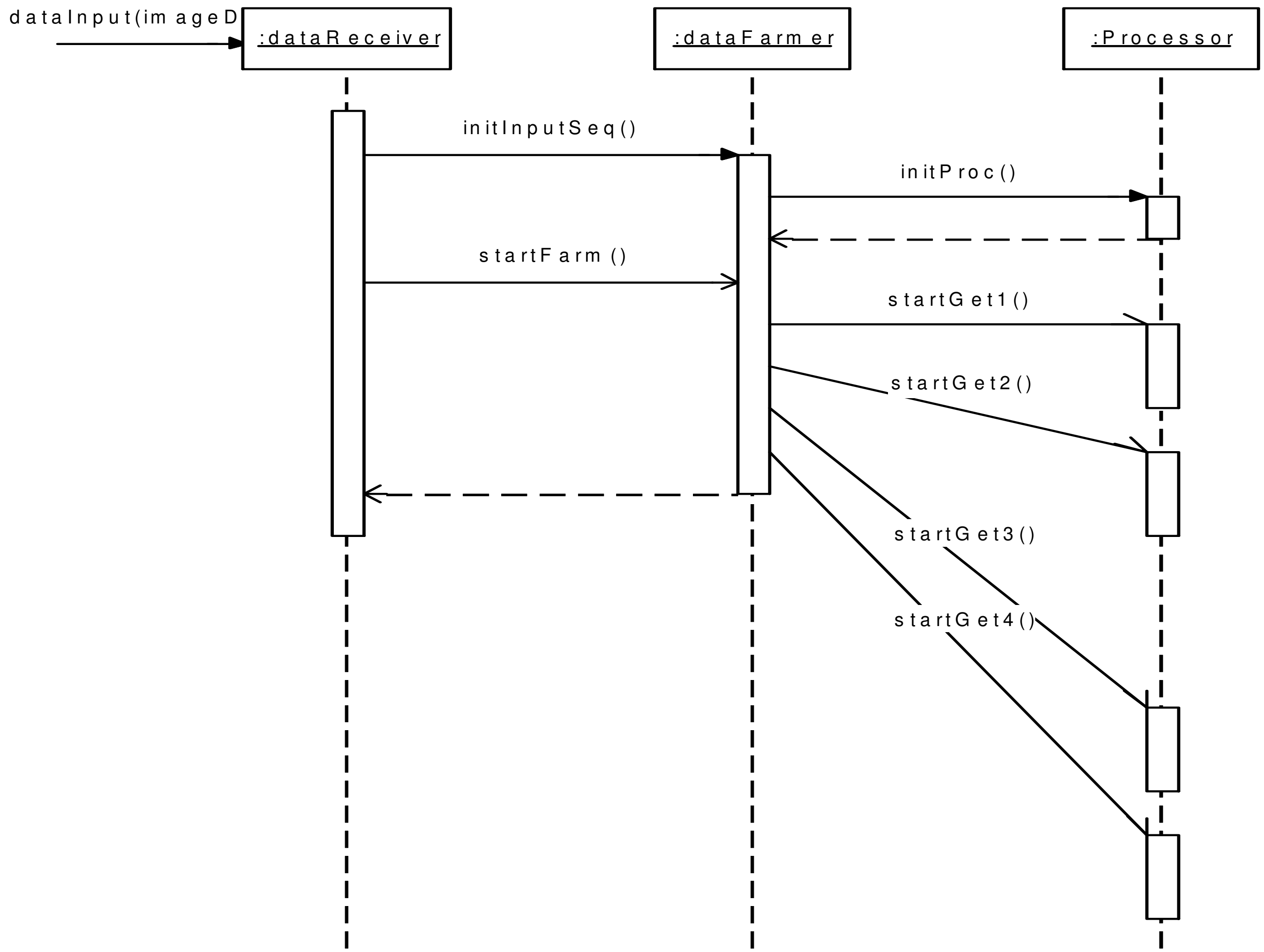
Using Labels in UML Sequence Diagrams (2/2)



Send message Req() to port until status() from printer is “data_ready”

Example of labels specifying iteration. The controller queries in a cyclic manner the port for data from the printer.

Other Message Type Examples in UML Sequence Diagram



Some Points to Ponder

1. Compare the following term-pairs:
   - State – Behaviour
   - Class – Object
   - Action – Activity
   - Use-case – Scenario
   - Method – Message
2. Do lifelines always continue down the entire page of a sequence diagram? Explain.
3. What is meant by focus of control in sequence diagrams?
4. What are the essential parts of a sequence diagram message? Give one concrete example.
5. How do synchronous and asynchronous messages differ? Give one concrete example of each case.

Workshop Activity -16-

Assuming that a microwave oven is analysed as the following objects:

- Oven
- Light
- Emission tube
- Timer

The main actions associated with the microwave oven are as follows:

- Opening the door
- Closing the door
- Using the control button
- Completion of the prescribed cooking interval

Create a <u>sequence diagram</u> for the following scenario:

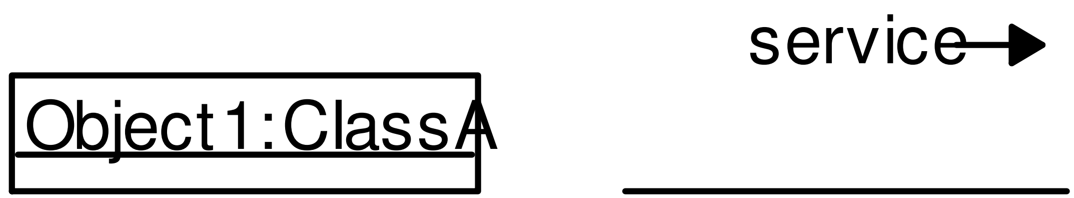
– Open the door, insert the food, close the door set the 1-minute timer, wait for cooking to complete, open the door and retrieve the food.

Create a <u>state diagram</u> for the above scenario.

UML Collaboration Diagram

- Show interactions between collaborating (interacting) objects.
- A bit like a cross between a class and a sequence diagram (object diagram with msgs)
- Are mainly a design tool
- Collaboration diagrams are not time-ordered
- Can serve as basis for a sequence diagram
- Like sequence diagrams, collaboration diagrams employ message passing.

Basic Components of a UML Collaboration Diagram



Service:
`Sequence-number:[condition]:message(parameter)`

A return value can be shown as an assigned ":=" expression

Sequence-number is a decimalised value (e.g. 1, 1.3, 2.3.1, etc.)

Collaboration Diagram Messages

Syntax is:

`predecessor guard sequence return-value:=signature`

Comma-separated list of message path numbers

A condition clause

An integer (sometimes with a trailing letter) followed by a recurrence

Variable

An operation call

Examples of Collaboration Diagram Messages

```
1:display()

[mode=display]1.2.1:redraw()

[balance > 0]5:debit(amount)

2*[n:=1..m]curr:=nextSeq(n)

2.3[x<0]:doodle()

3.1[y=>]:write()

1.1a,1.1b:doMore()

2.2a,2.2b/3:playback()
```

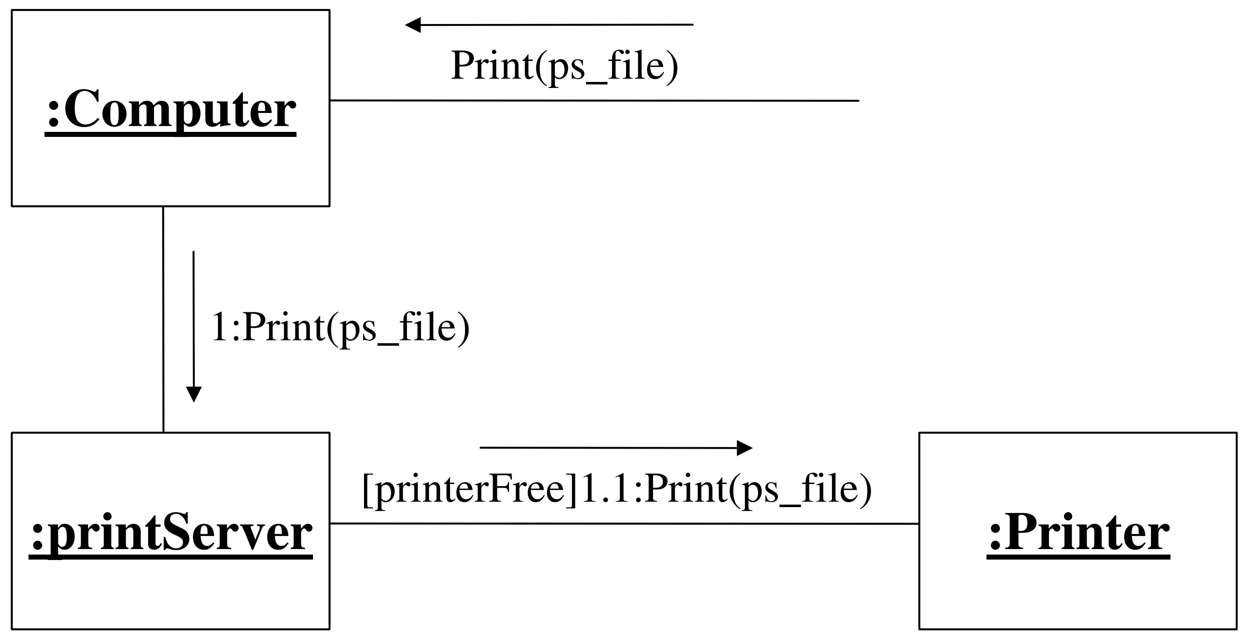
Predefined Stereotyped Links in Collaboration Diagrams

- Global
  - (link role) Instance is available as a name known throughout the system
- Local
  - (link role) Instance is available as a local variable in an operation
- Parameter
  - (link role) Instance is available as a parameter in an operation
- Self
  - (link role) Object can send messages to itself
- Vote
  - (group of messages) Return value is chosen by majority vote of all returned values from a given collection
- Broadcast
  - (group of messages) No message ordering in given group

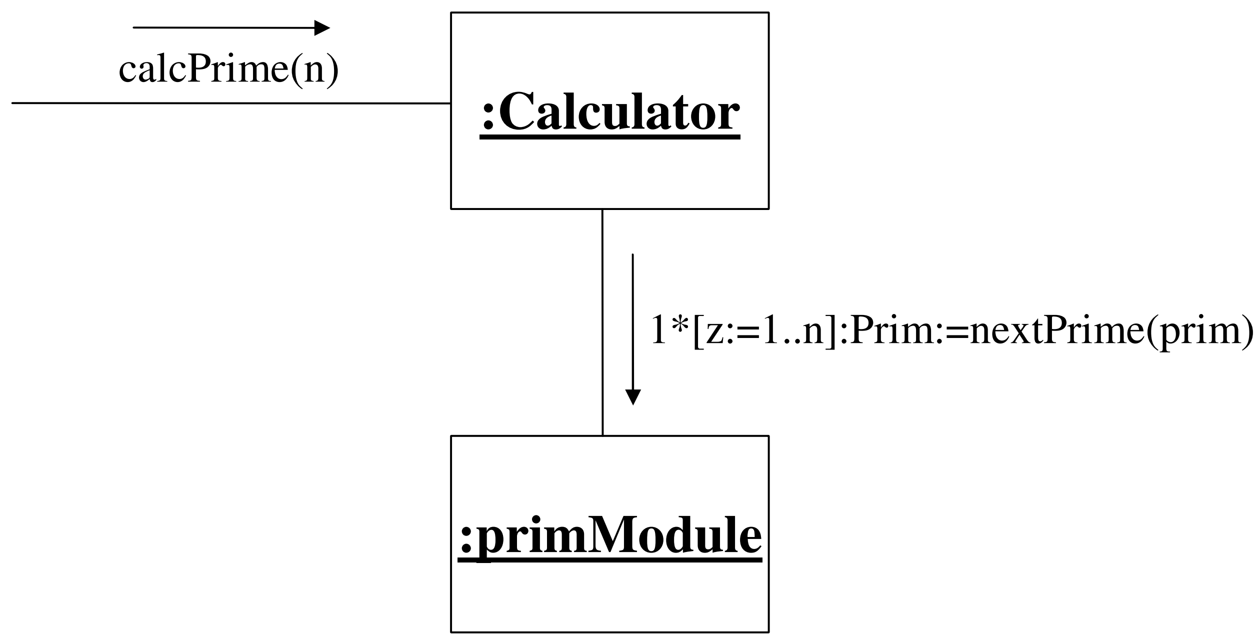
Collaboration Diagram Object Lifeline Predefined Stereotypes

- New
  - Object created during a collaboration
- Destroyed
  - Object destroyed during a collaboration
- Transient
  - Object created *and* destroyed during the same collaboration

UML Collaboration Diagram Example

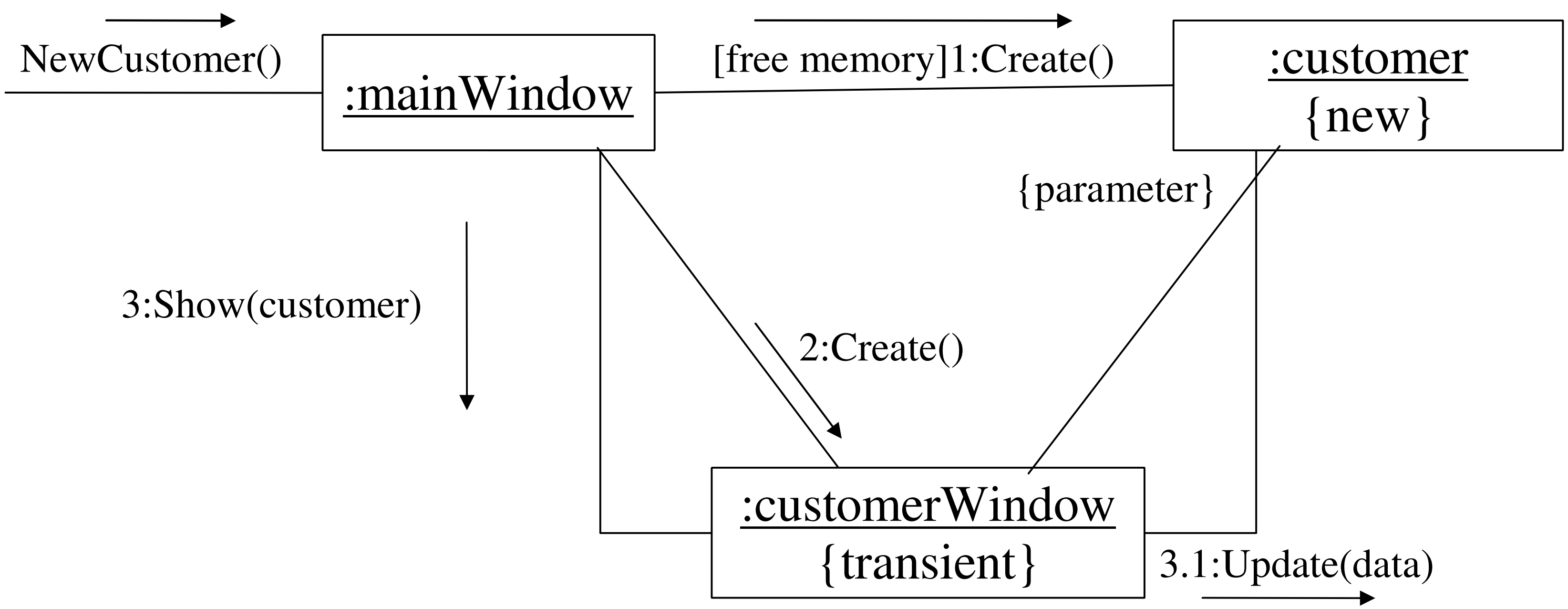


UML Collaboration Diagram Example with Return-Value

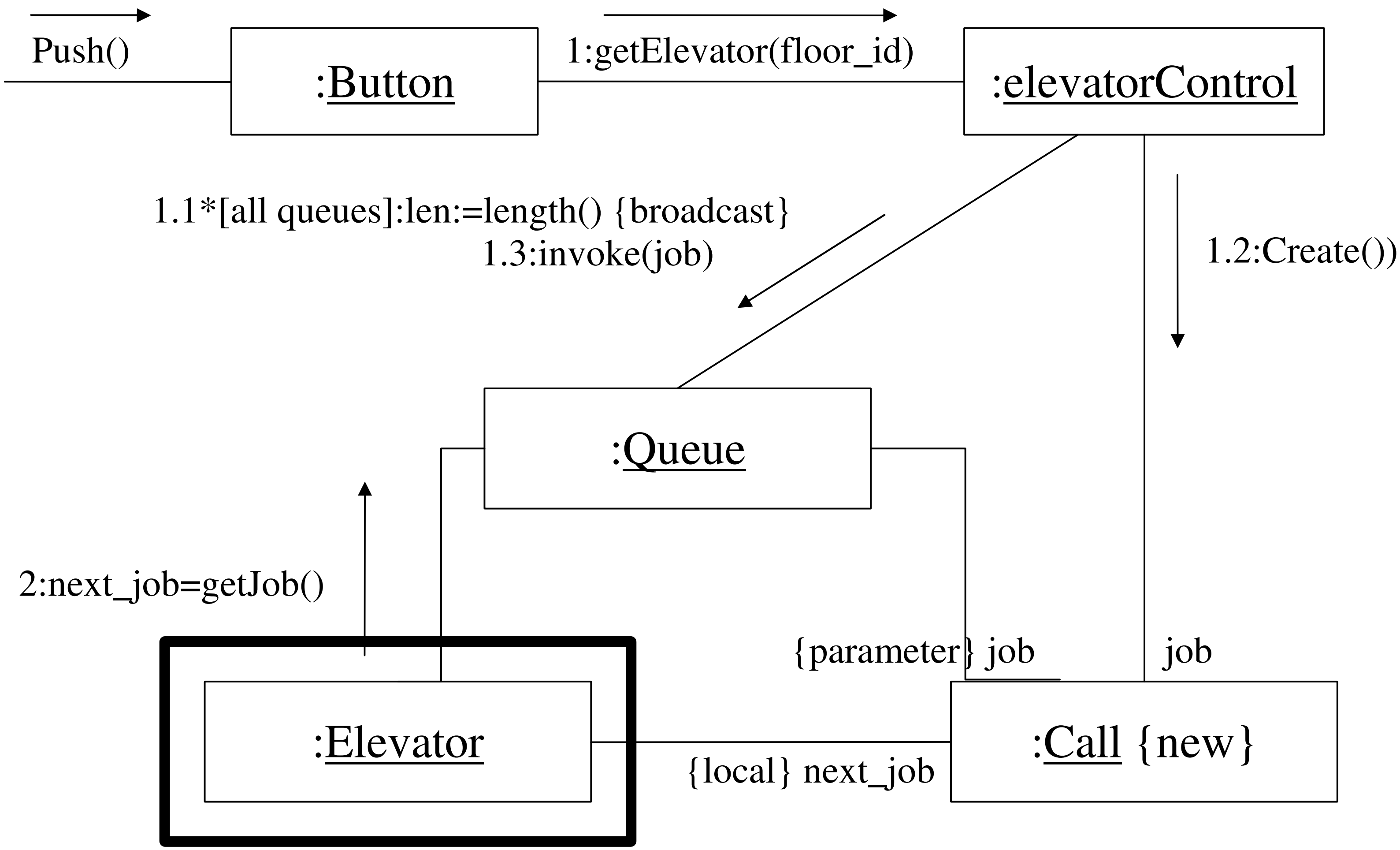


Try This Yourself…

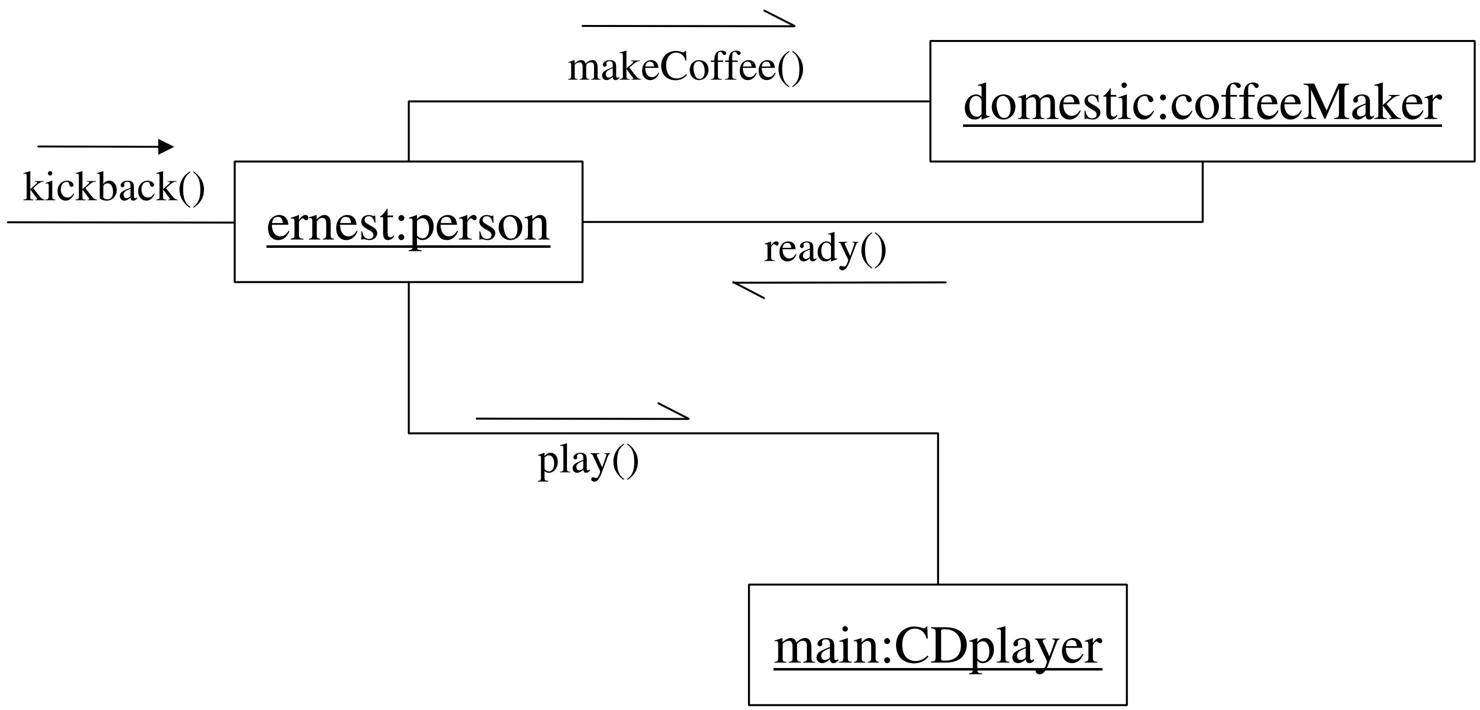
- Explain in plain English, what the following collaboration diagram depicts:



…and This One Too Please…



Example of Asynchronous Collaboration Diagram



Steps in Building a UML Collaboration Diagram

1. Set the context
2. Identify which objects and which associations between them participate in the collaboration
3. Draw the classes/objects and link them
4. Insert the messages
5. Validate the diagram

Workshop Activity -17-

Assuming the following 4 classes:

- CampaignDialogue
- Client
- Campaign
- Advert

Produce a collaboration diagram modelling the addition of a new advert to an existing campaign. Stereotypes need not be used.

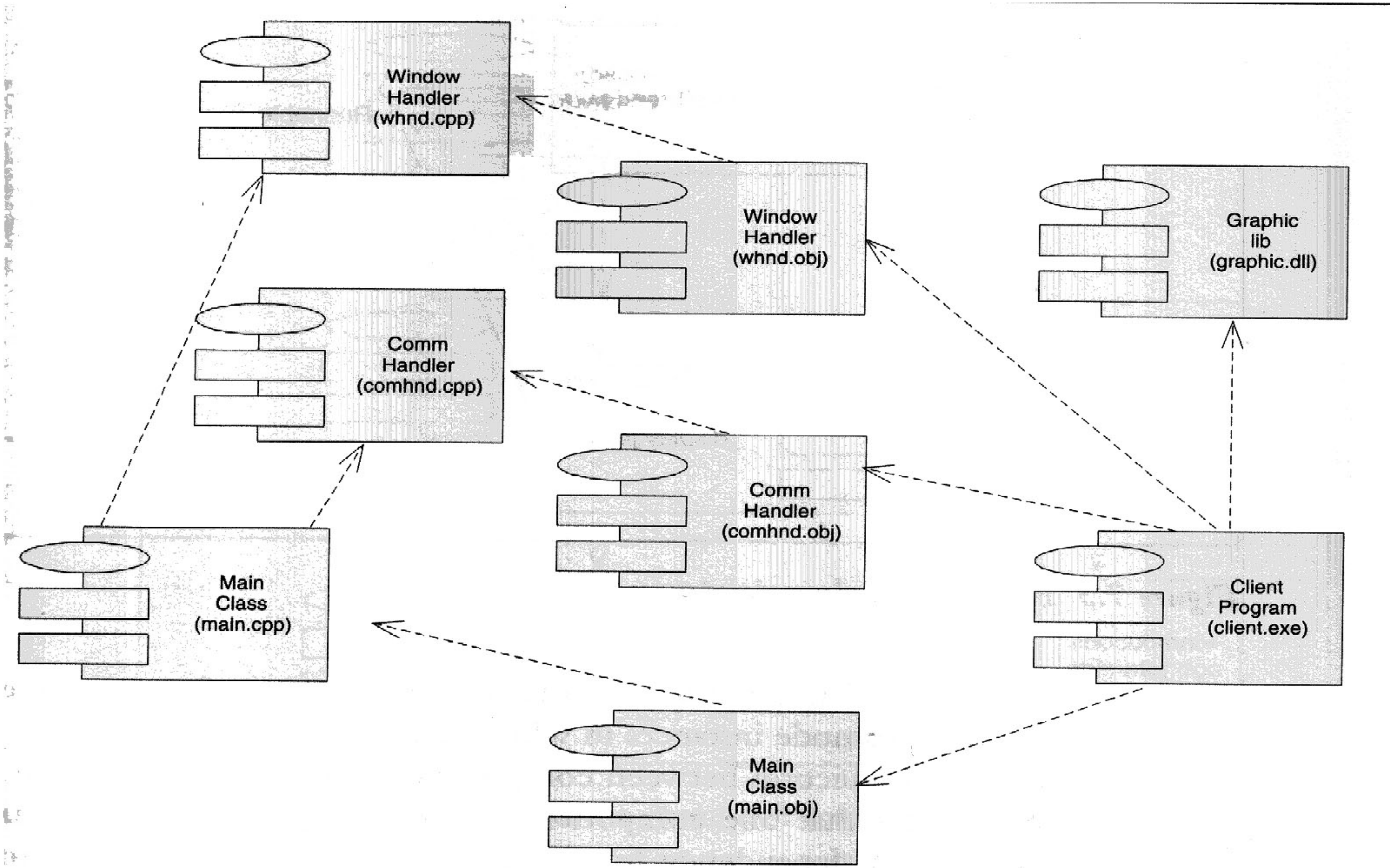
UML Component Diagrams

- Models parts of software and their inter-dependencies
- Represents code structure
- Are generally implemented in physical terms as “files”
- Come as either “source”, “binary”, or “executable” components
- ONLY executable type components can be instantiated

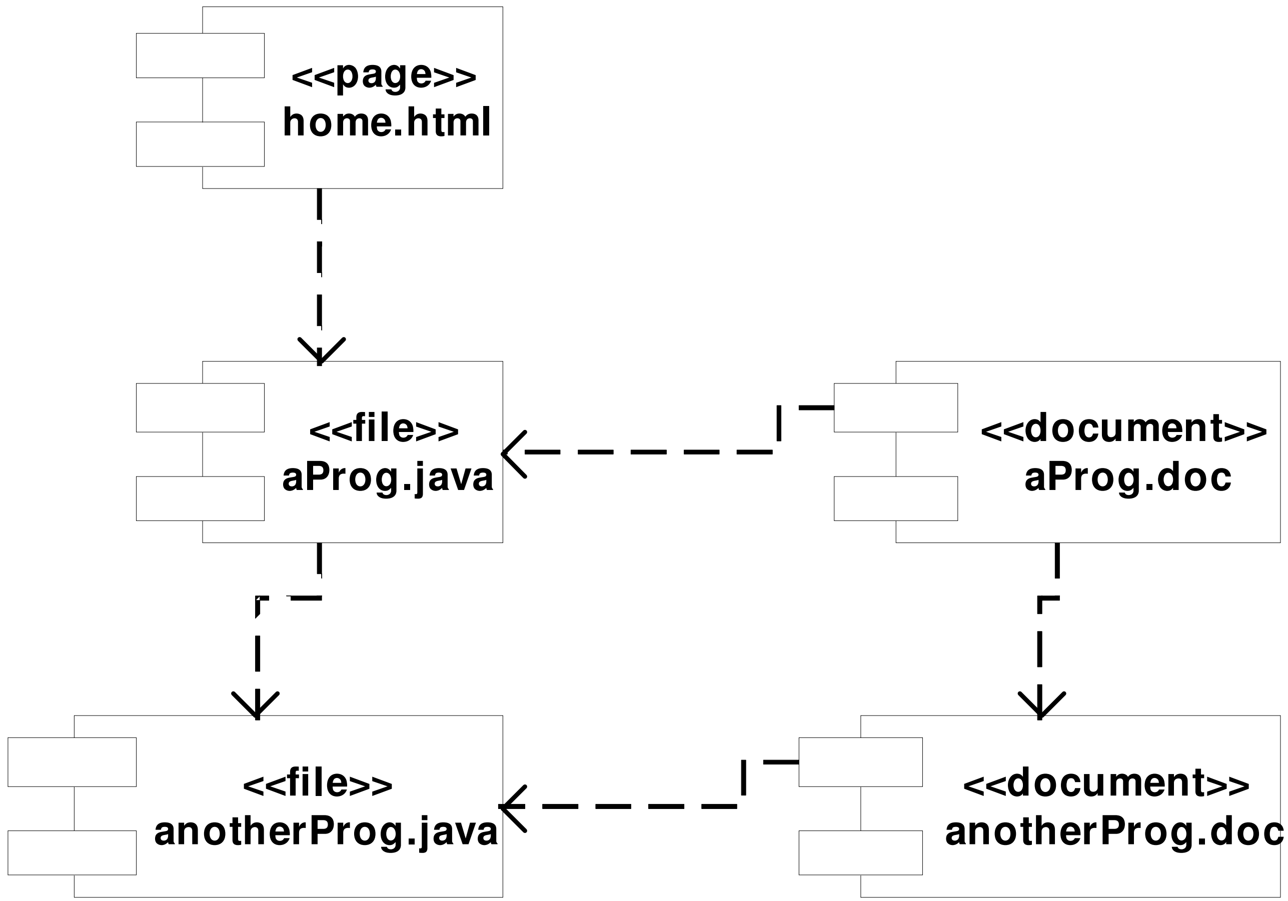
Predefined Component Diagram Stereotypes

- File
  - Usually a source code file
- Binary / Library
  - Usually a compiled segment directly linkable into other compilations
- Executable
  - Usually a directly executable module
- Table
  - Usually a database table
- Page
  - Usually a Web page
- Document
  - Usually a documentation file (as opposed to compilable code)

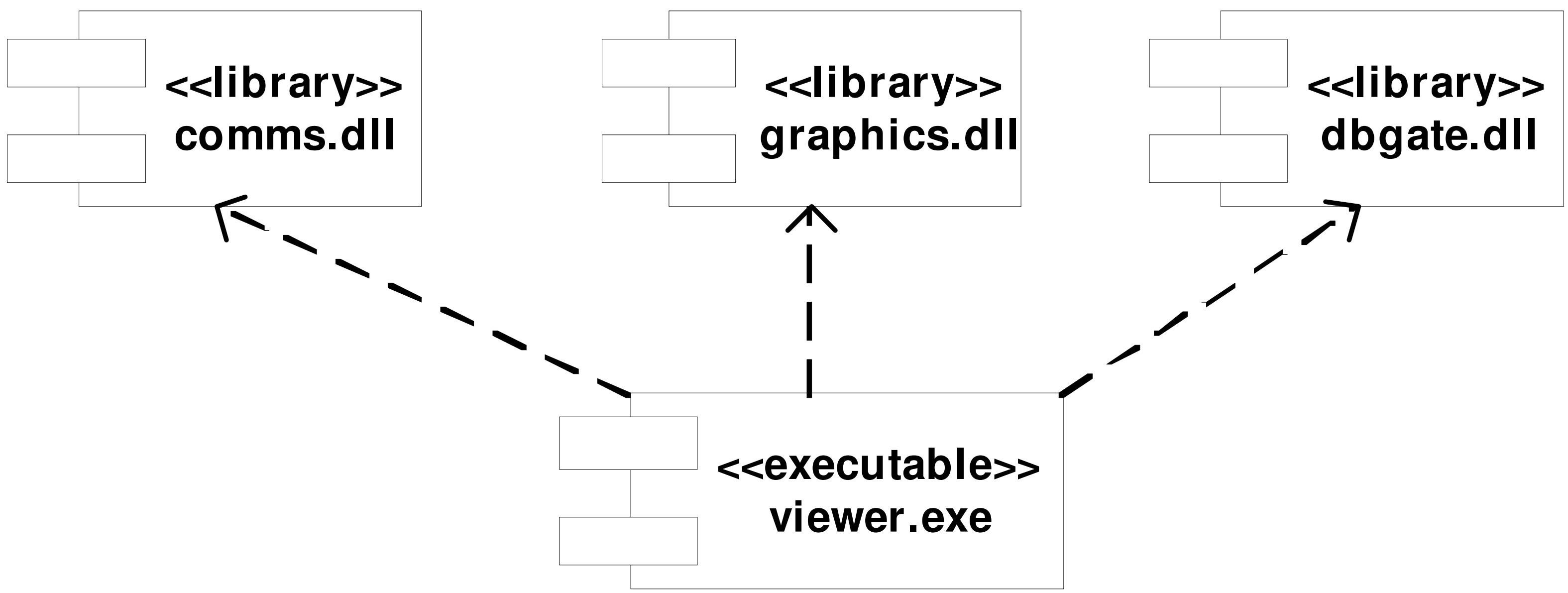
Example of a UML Component Diagram



Example of Source Code Dependencies



Runtime Component Example

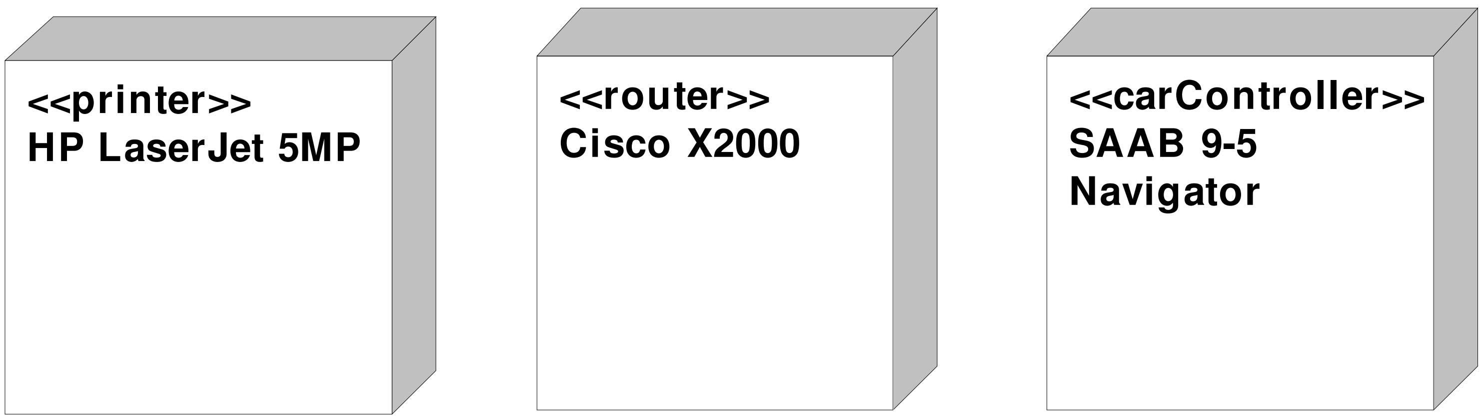


UML Deployment Diagrams

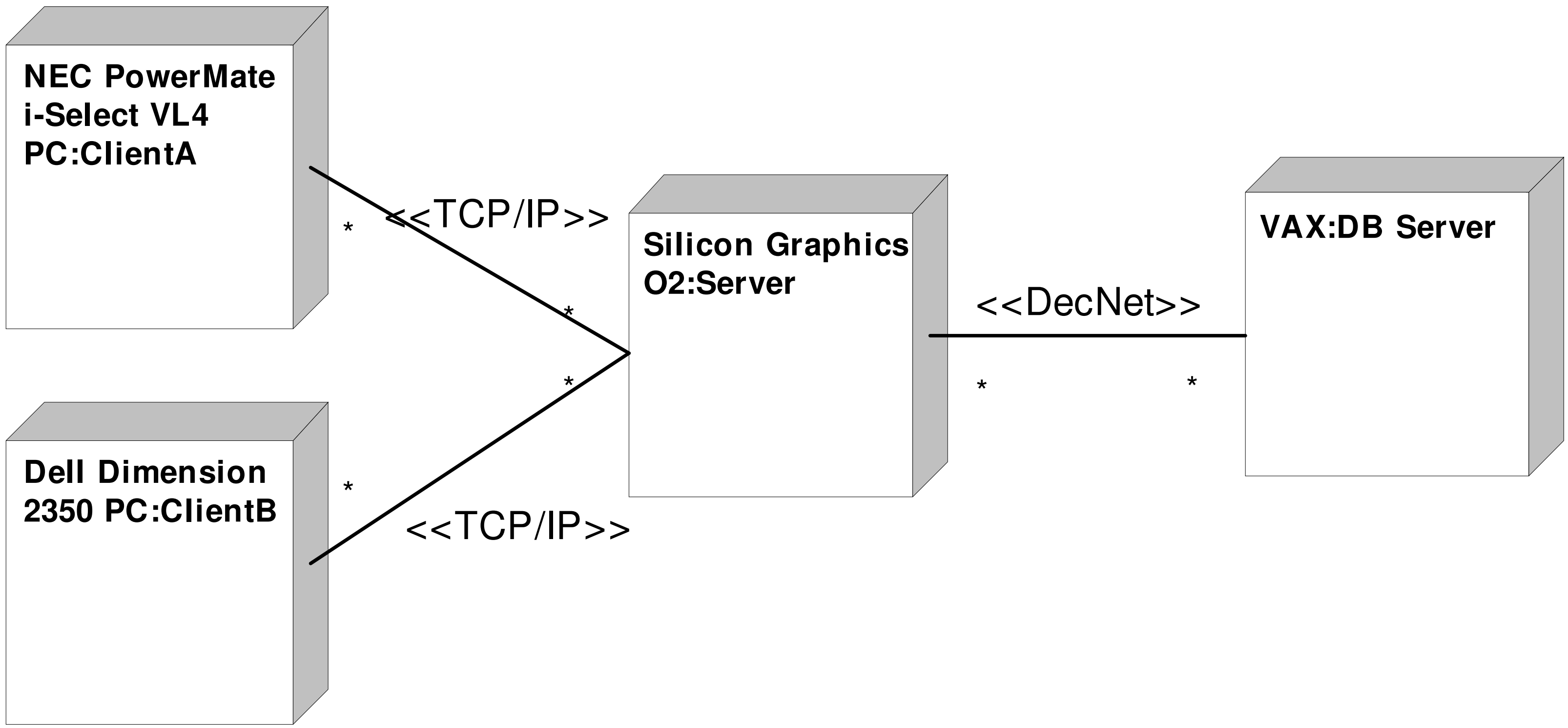
- Models the run-time architecture (topology) of:
  - Processors
  - Devices
  - Software components
- Is ultimately traceable to initial requirements

Topology node → Software component → Classes, objects, Collab., etc. → Use-case requirement

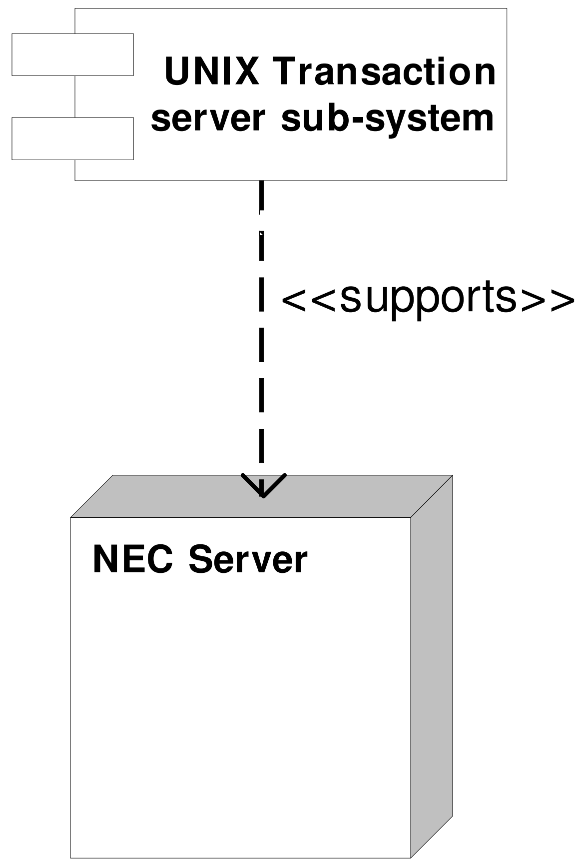
Stereotype Examples in Deployment Diagrams



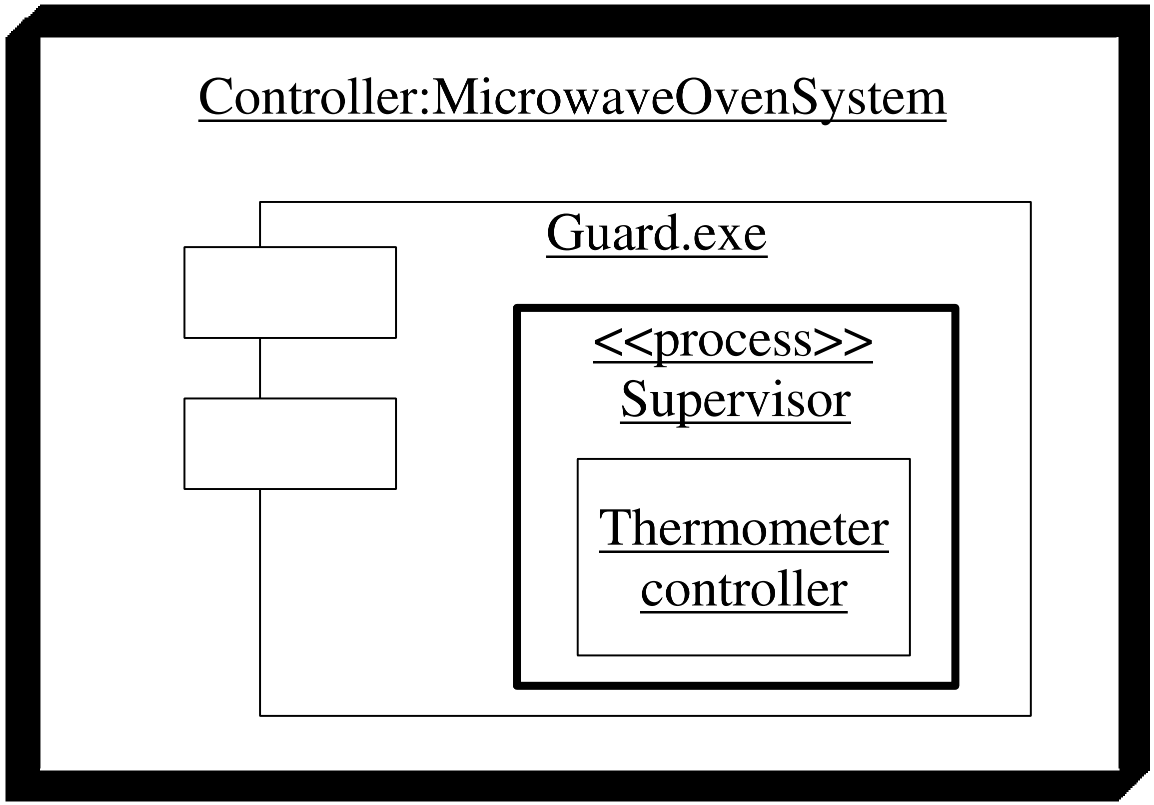
Communication Associations in Deployment Diagrams



Component Support in UML Deployment Diagrams

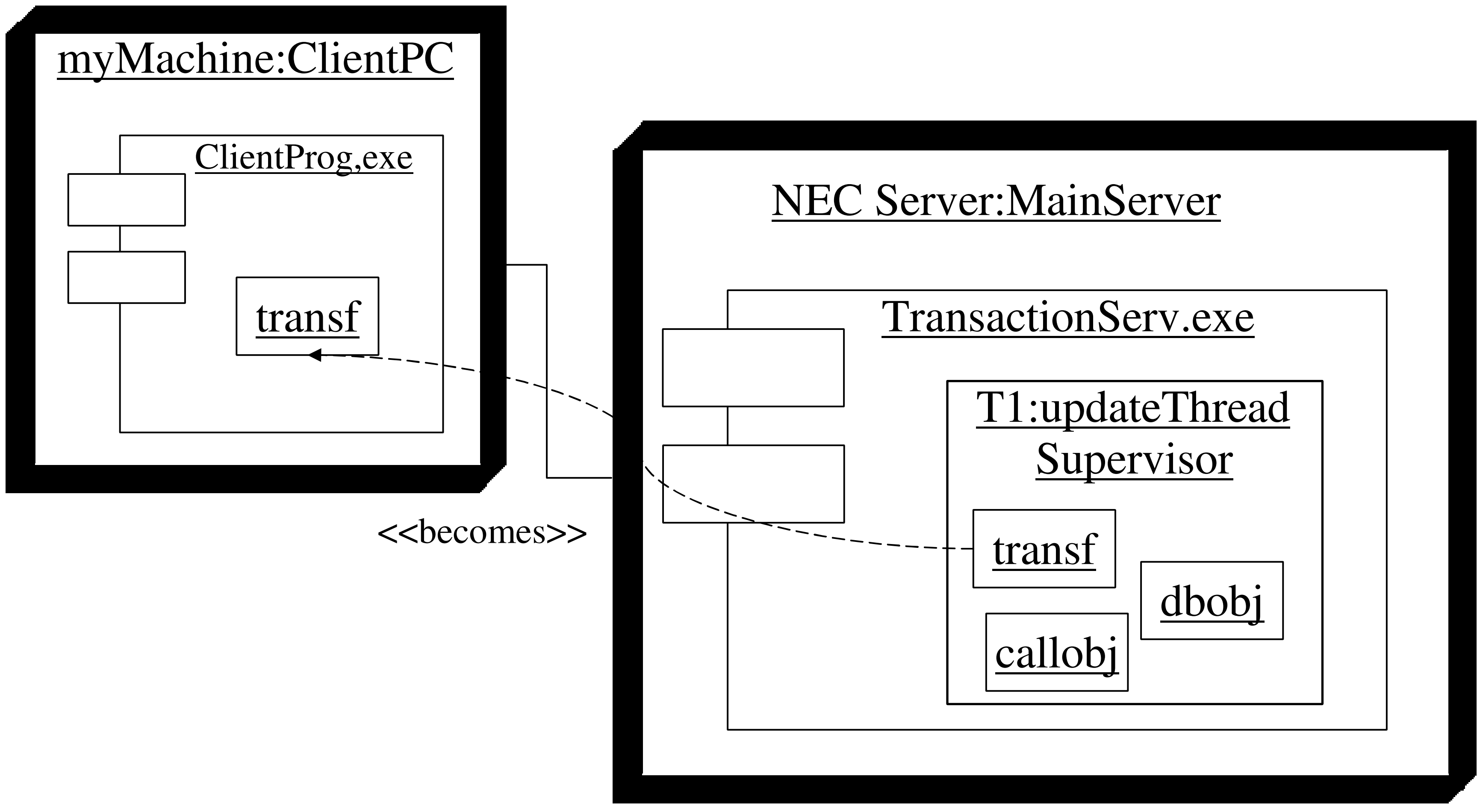


Object Allocation in Deployment Diagrams

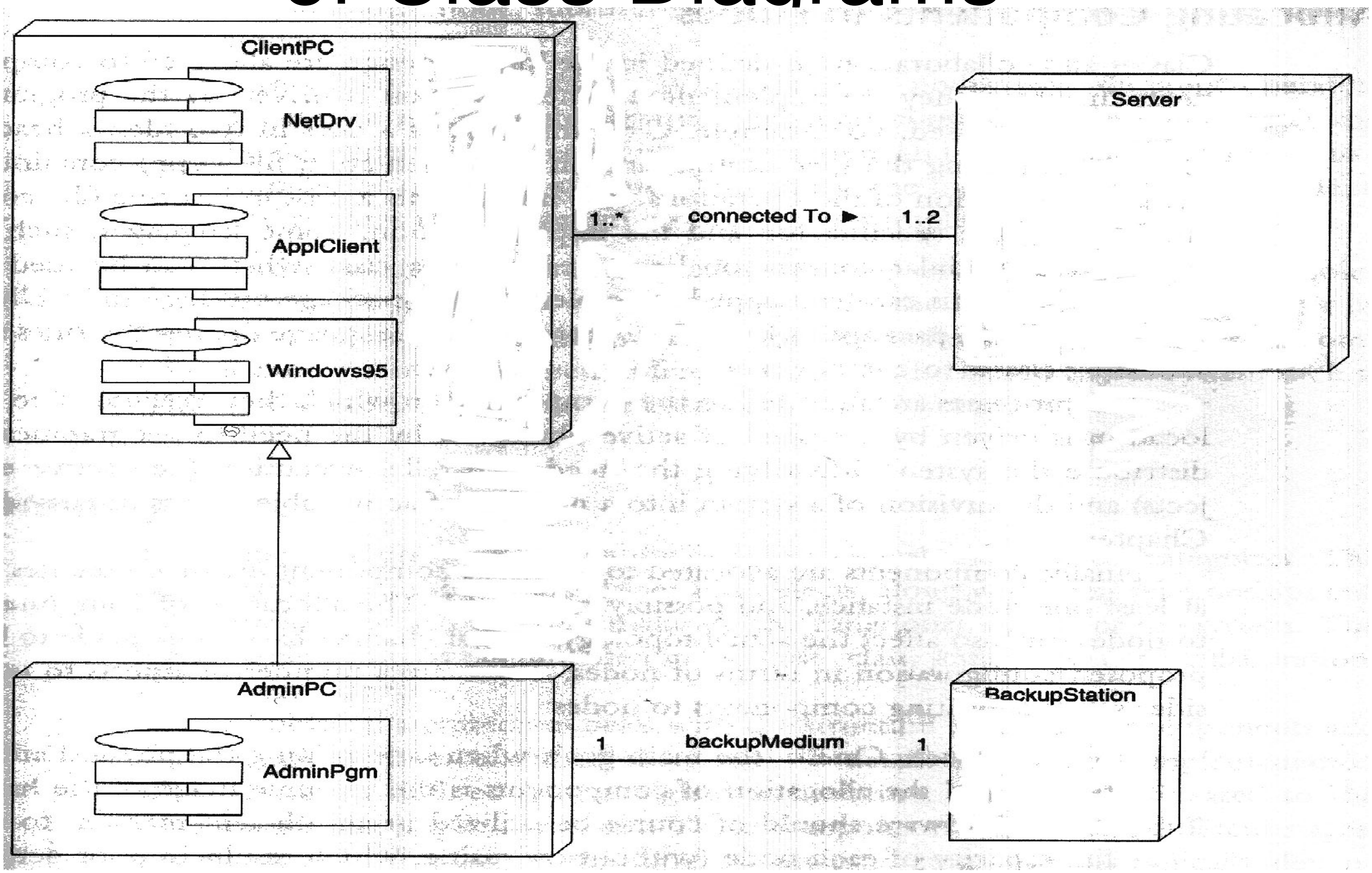


Bold object border indicates active object
(i.e. <<process>> or <<thread>>)

Object Transferability in Deployment Diagrams



Deployment Diagrams in the form of Class Diagrams



The End

THANK YOU ALL FOR YOUR TIME
AND EFFORT