# Modellistica Medica

Maria Grazia Pia

INFN Genova

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

# Lezione 6

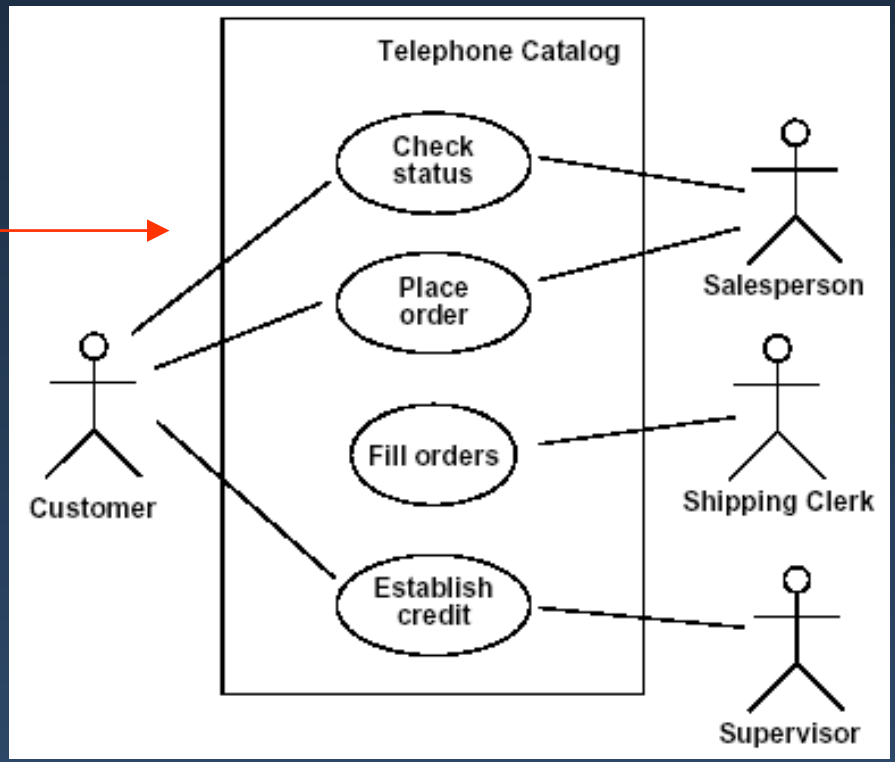
UML

Introduction

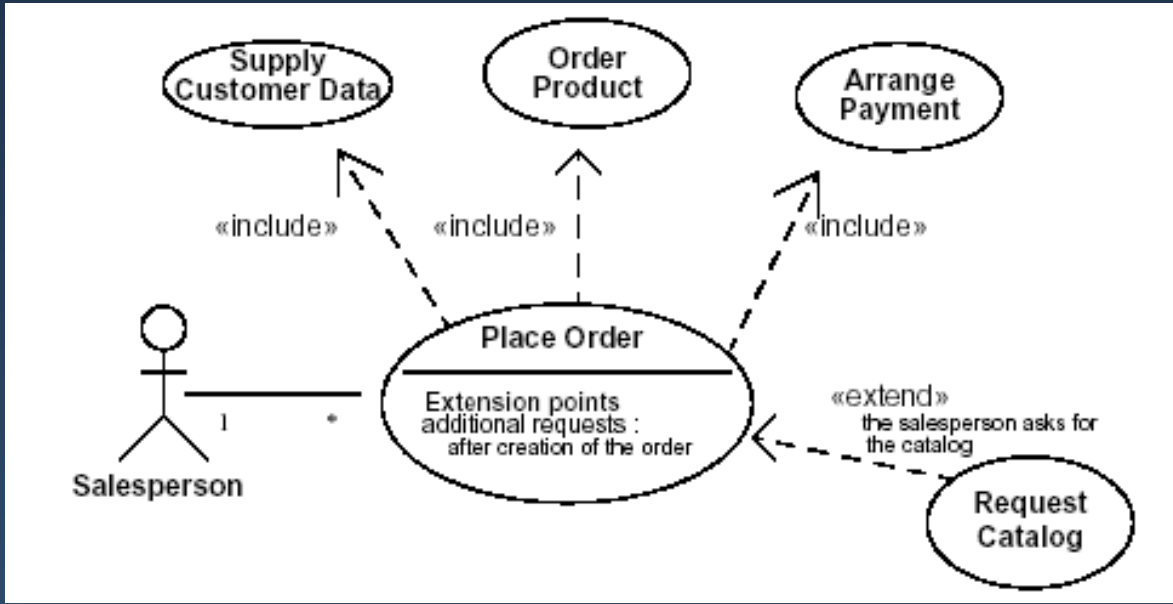
Structural diagrams

# Basics

What is?

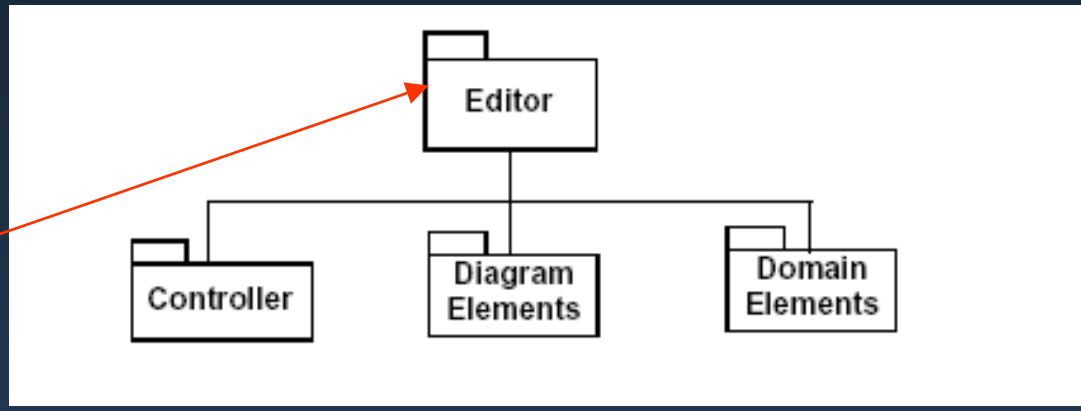


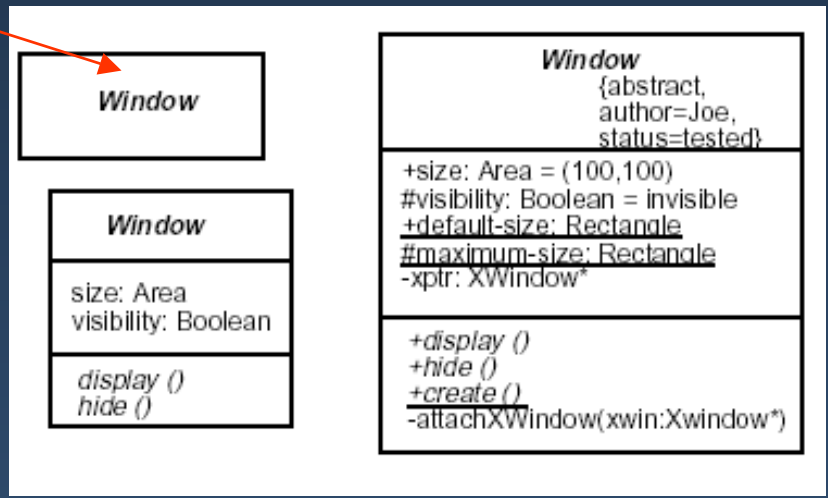
# Please explain



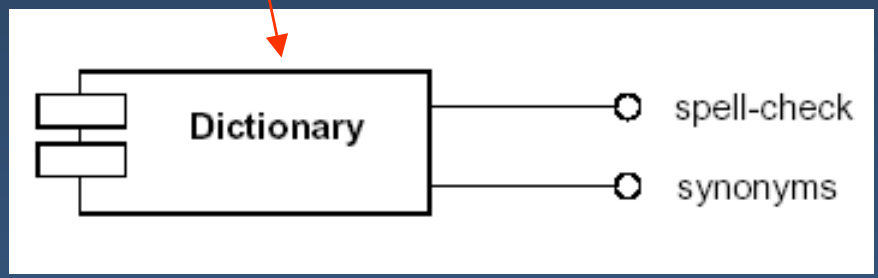
# UML

What is?



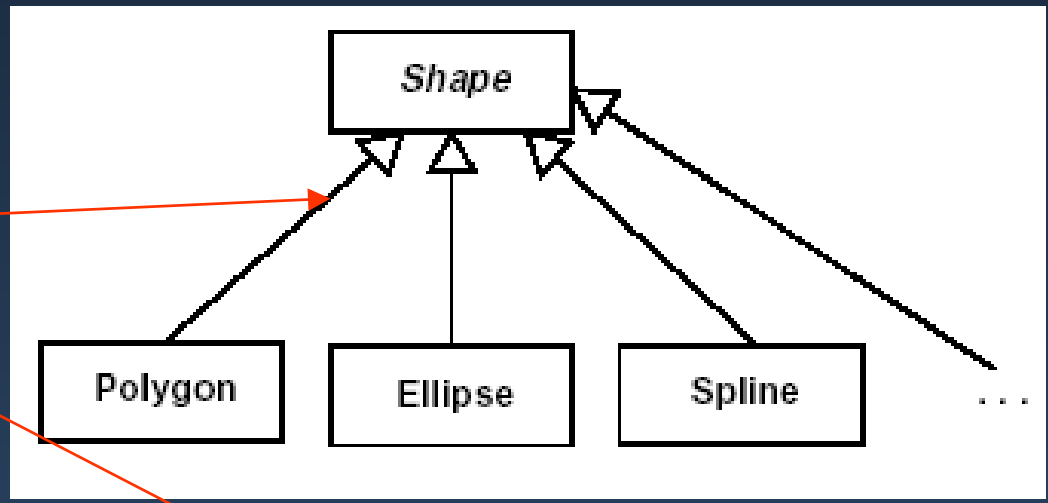



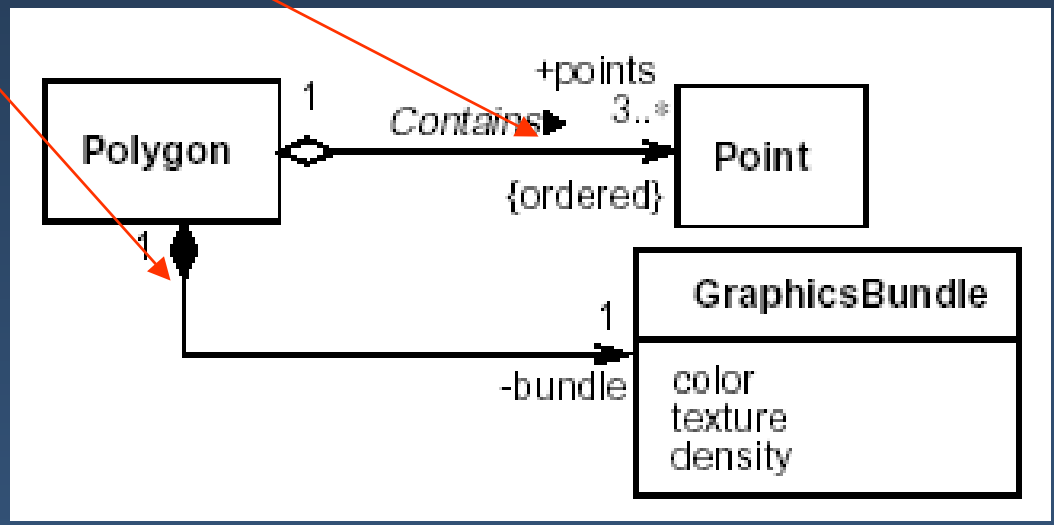

What do +,- # mean?



# Please explain

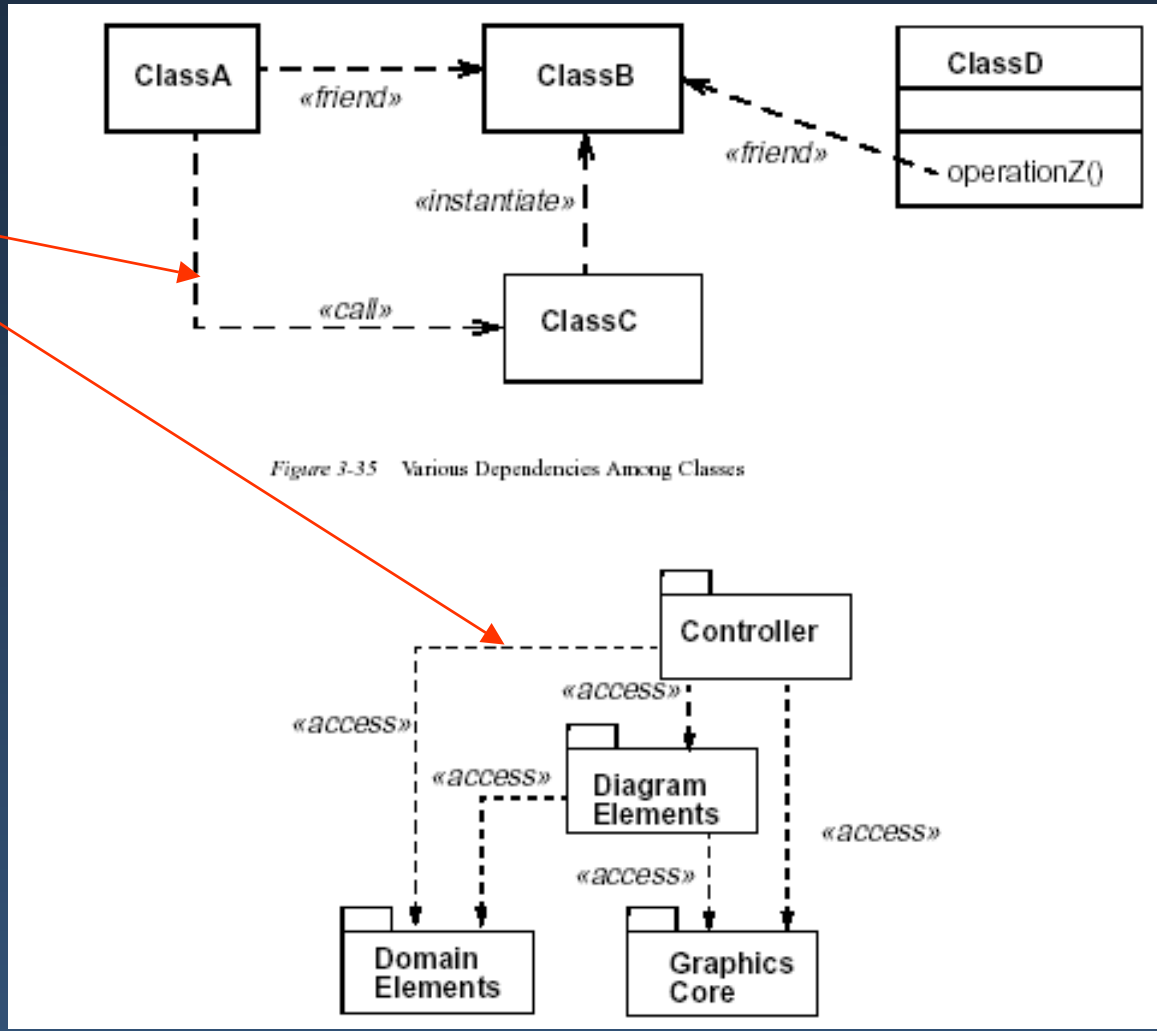
What is?



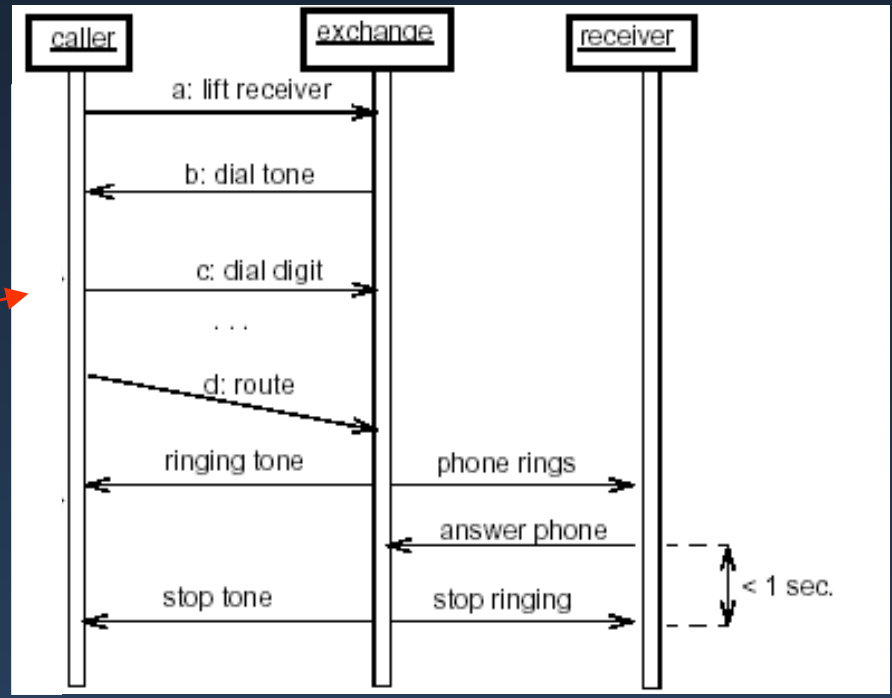
# UML

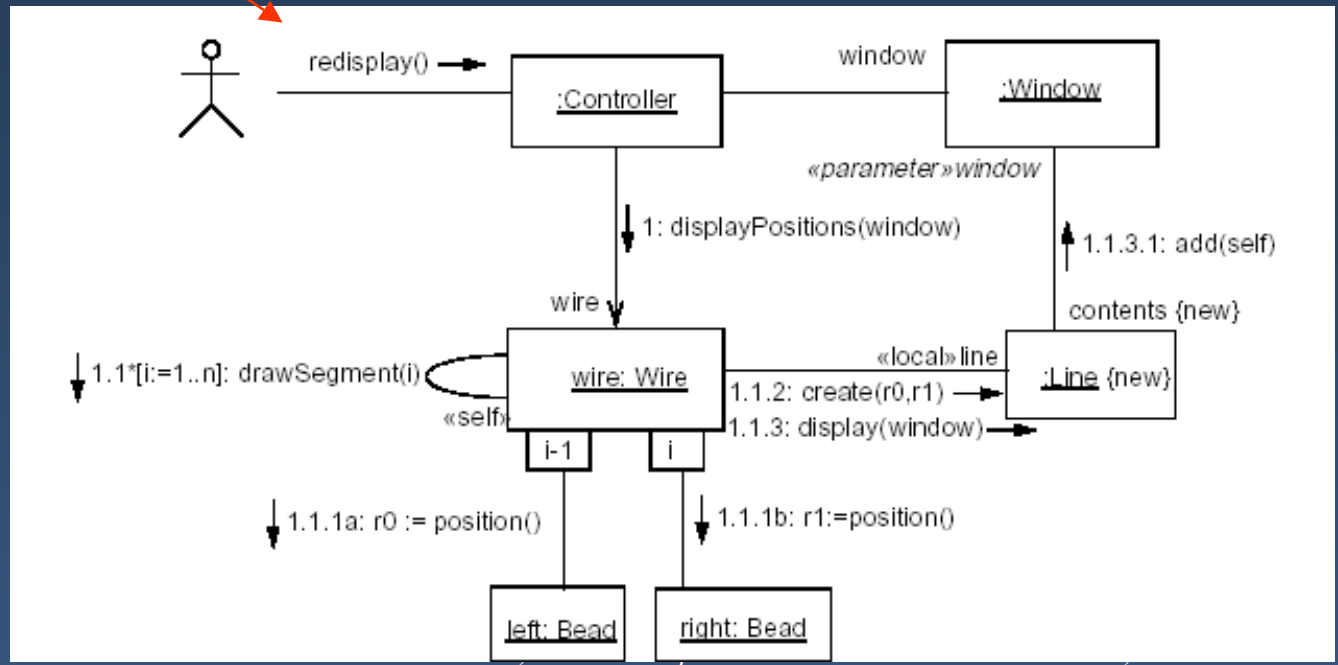
What is?


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

# Modeling

- Modeling is the designing of software applications before coding
- The OMG's Unified Modeling Language™ (UML™) helps you
  - **specify**
  - **visualize**
  - **document**

  models of software systems, including their structure and design

# Reasons to Model

- To communicate the desired structure and behavior of the system
- To visualize and control the system's architecture
- To better understand the system and expose opportunities for simplification and reuse
- To manage risk

- Every model may be expressed at different levels of precision
- No single model is sufficient

# UML in One Sentence

The UML is a graphical language for

- visualizing
- specifying
- constructing
- documenting

artifacts of software systems

*(can be used also in non-software systems)*

# History

- The UML got its initial start from the combined efforts of
  - Grady Booch, with his Booch method
  - James Rumbaugh, with his Object Modeling Technique (OMT)
  - Ivar Jacobson, with his Object-Oriented Software Engineering (OOSE) method
- It represents the evolutionary unification of their experience with other industry engineering best practices
- Beginning in 1994, under the auspices of Rational Software Inc., the UML began to take shape
- The Object Management Group (OMG) adopted the UML in November 1997 as the official object-oriented modeling notation for describing application domains

# Elements

A modeling language must include:

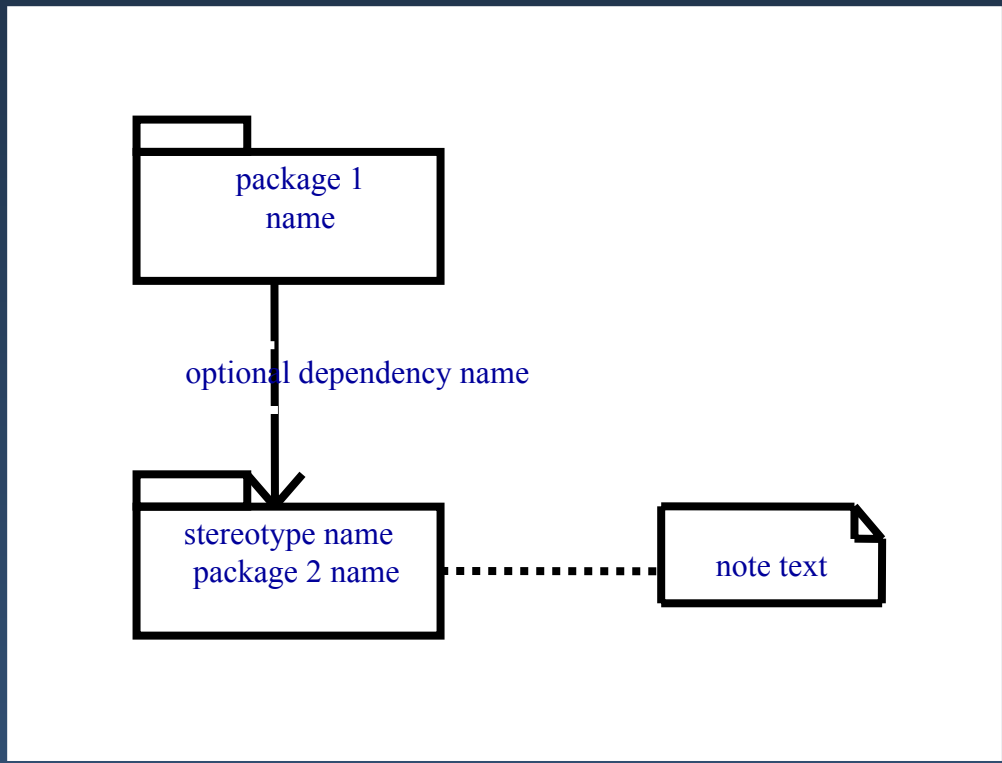
- Model elements
  - fundamental modeling concepts and semantics
- Notation
  - visual rendering of model elements
- Guidelines
  - idioms of usage

# What one can model

- Structural Diagrams
  - Class Diagram
  - Object Diagram
  - Component Diagram
  - Deployment Diagram
- Behavior Diagrams
  - Use Case Diagram
  - Sequence Diagram
  - Activity Diagram
  - Collaboration Diagram
  - Statechart Diagram
- Model Management Diagrams
  - Packages
  - Subsystems
  - Models

# General purpose concepts

Can be used in various diagram types



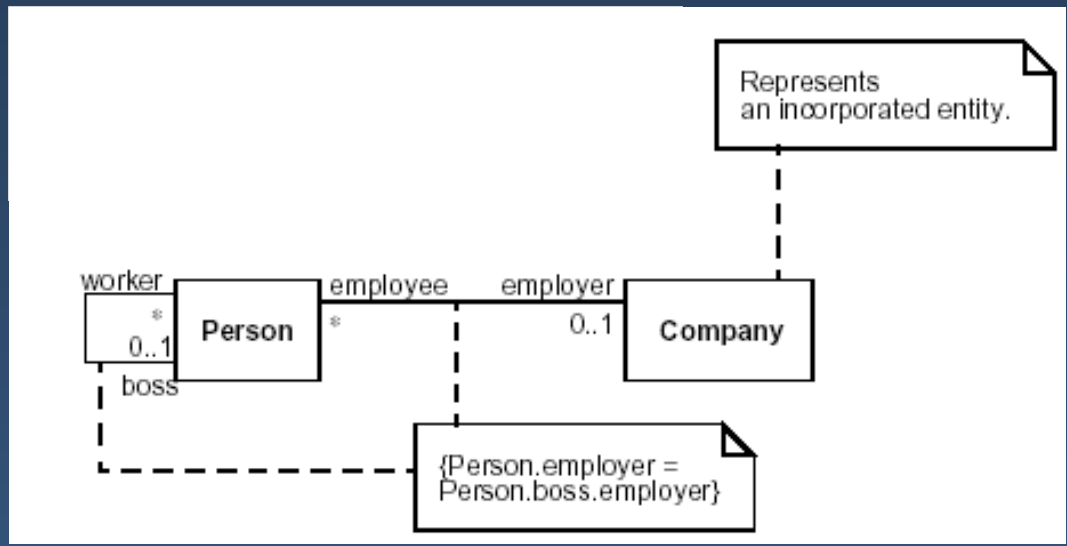
# Notes

A note is a graphical symbol containing text and/or graphics that offer(s) some comment or detail about an element within a model.

Check with Mike on this.

See encrypt.doc

See http://www.softdocwiz.com



# Adornments and Extensibility

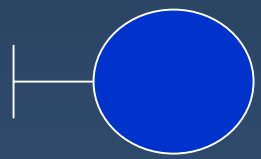
- An adornment is an item, such as a note, that adds text or graphical detail to a model

The UML offers various mechanisms that one can use to extend the “official” language

- stereotypes
- tagged values
- constraints

# Stereotypes

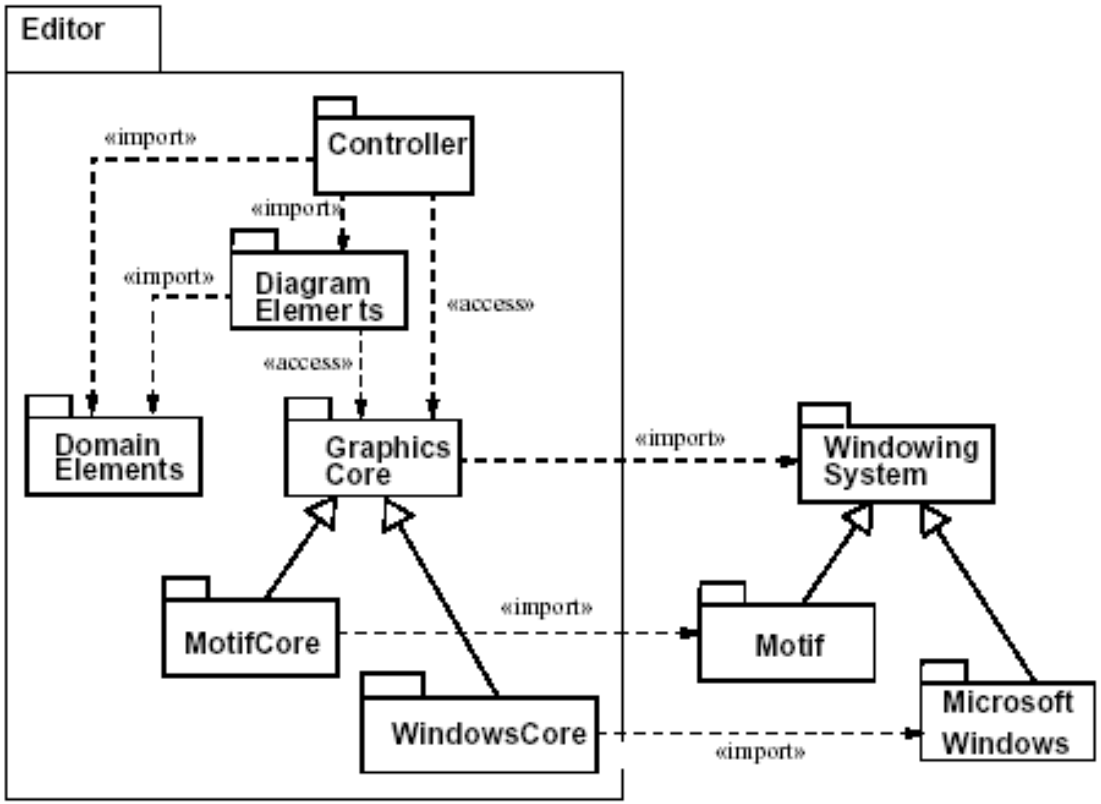
A stereotype is an extension of the vocabulary of the UML that allows you to create a new kind of “building block” that’s specific to the problem you’re trying to solve.

Humidity Sensor

# Packages

- A package is a general-purpose mechanism for organizing elements of a model, such as classes or diagrams, into groups Packages themselves may be nested within other packages.
- A package may contain both subordinate packages as well as other kinds of model elements.

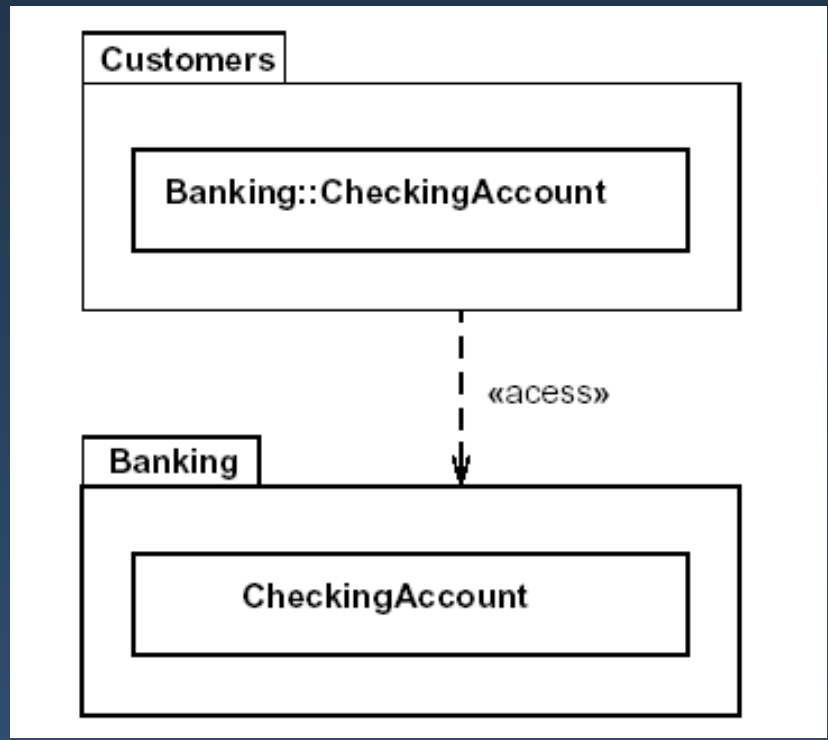



- All kinds of UML model elements can be organized into packages.
- Every element within a model is uniquely owned by one package

# Access dependency among packages

The access dependency is displayed as a dependency arrow from the referencing (client) package to the target (supplier) package containing the target of the references.

This dependency indicates that elements within the client package may legally reference elements within the supplier.



# Structural Diagrams

**Used to visualize, specify, construct, document static aspects of system**

- class diagram
- package diagram [not standard UML]
- object diagram
- component diagram
- deployment diagram

# Class diagrams

Class diagrams show the static structure of the model, in particular, the things that exist (such as classes and types), their internal structure, and their relationships to other things.

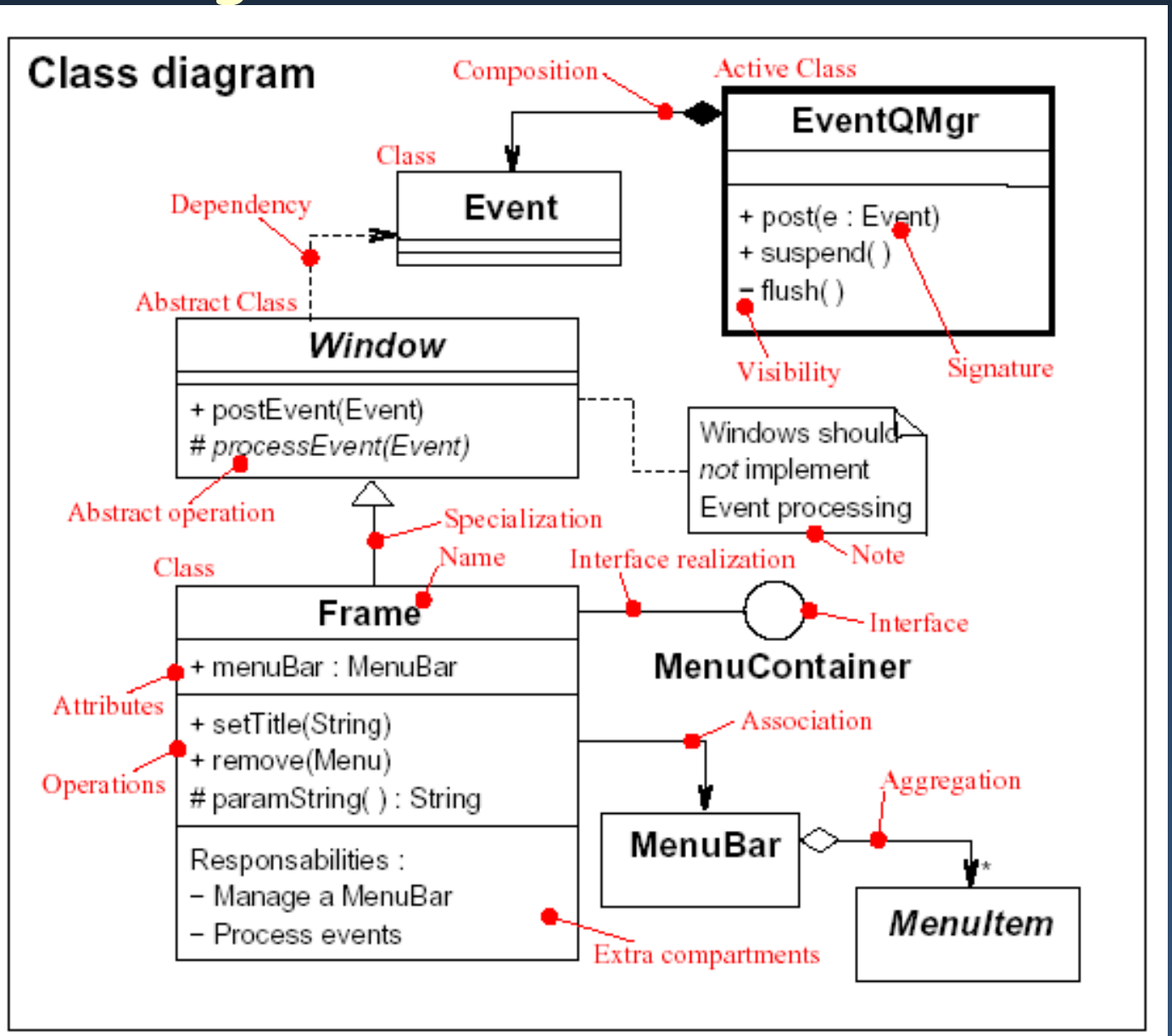
Perhaps a better name would be **"static structural diagram",** but "class diagram" is shorter and well established.

Class diagrams *do not show temporal information*, although they may contain reified occurrences of things that have or things that describe temporal behavior.

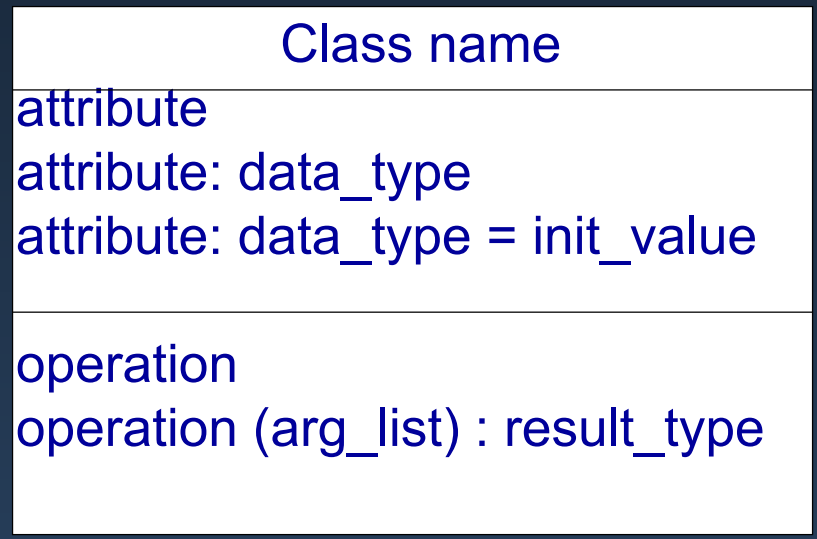
# Common Uses of Class Diagrams

- to model vocabulary of the system, in terms of which abstractions are part of the system and which fall outside its boundaries
- to model simple collaborations (societies of elements that work together to provide cooperative behavior)
- to model logical database schema (blueprint for conceptual design of database)

# Class diagram



# Class

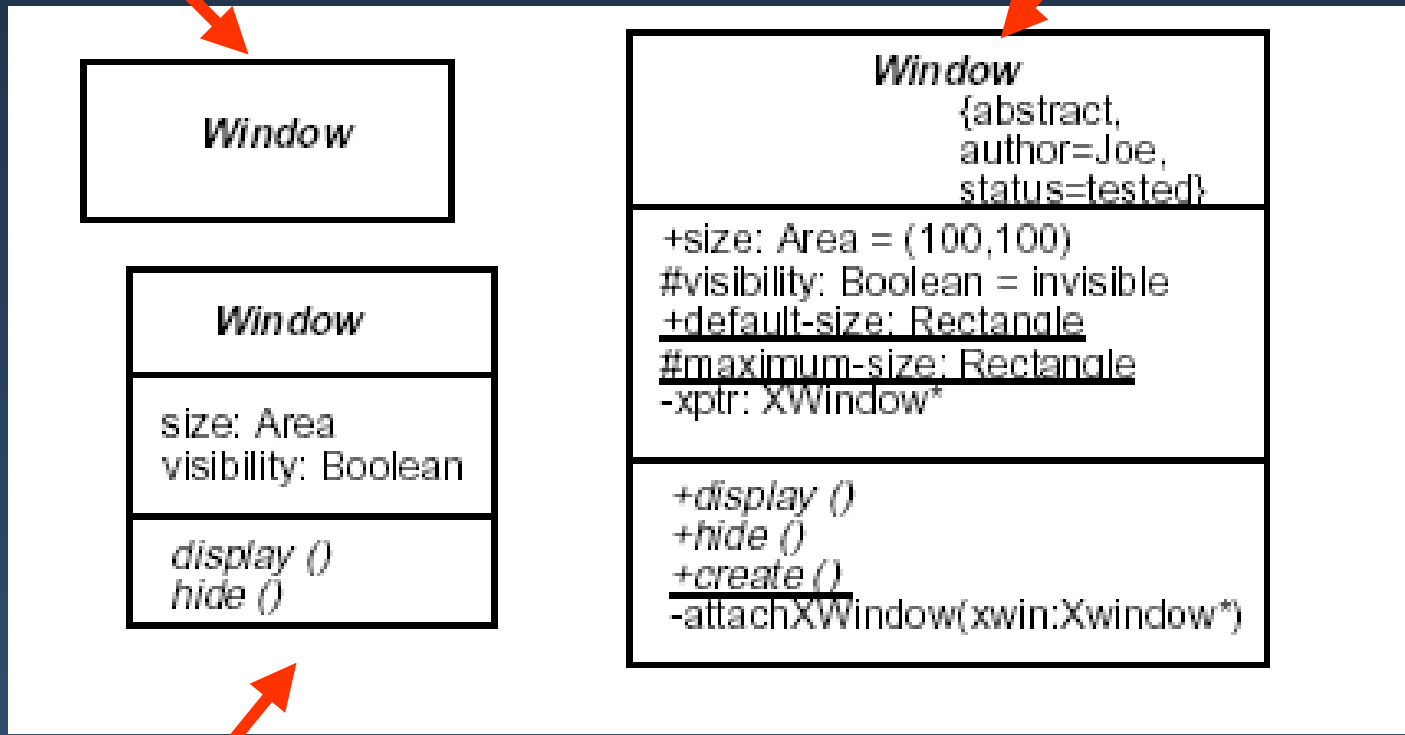




- A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics
- An attribute is a named property of a class that describes a range of values that instances of the property may hold
- An operation is a service that can be requested from an object to affect behavior

# Level of detail

Details suppressed

Implementation-level details




Analysis-level details

# Attribute

An attribute is semantically equivalent to a composition association; however, the intent and usage is normally different.

The default syntax is:

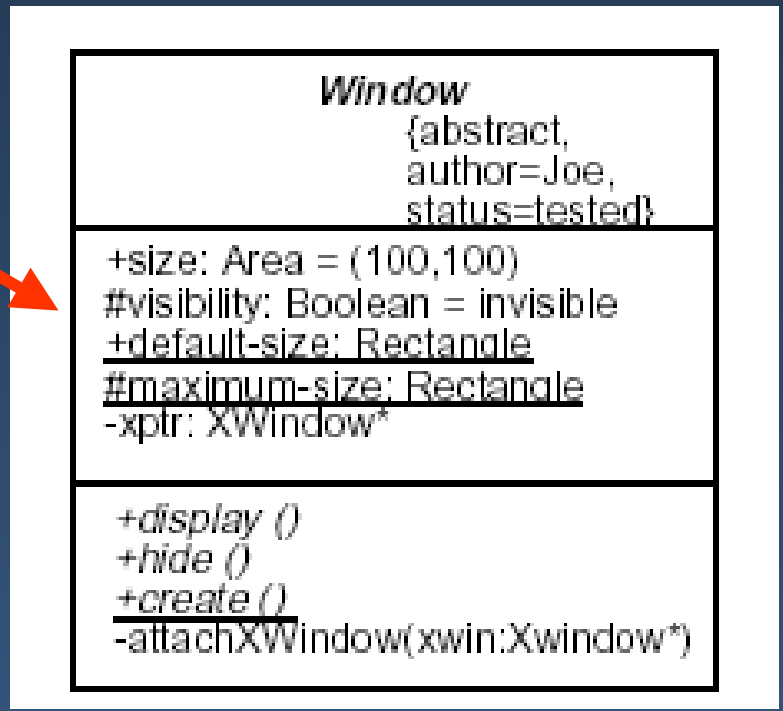
*visibility name* [ *multiplicity* ] : *type-expression* = *initial-value* { *property-string* }

where *visibility* is one of:

+ public visibility

# protected visibility

- private visibility



# Operation

An operation is a service that an instance of the class may be requested to perform. It has a name and a list of arguments.

An operation is shown as a text string that can be parsed into the various properties of an operation model element.

The default syntax is:

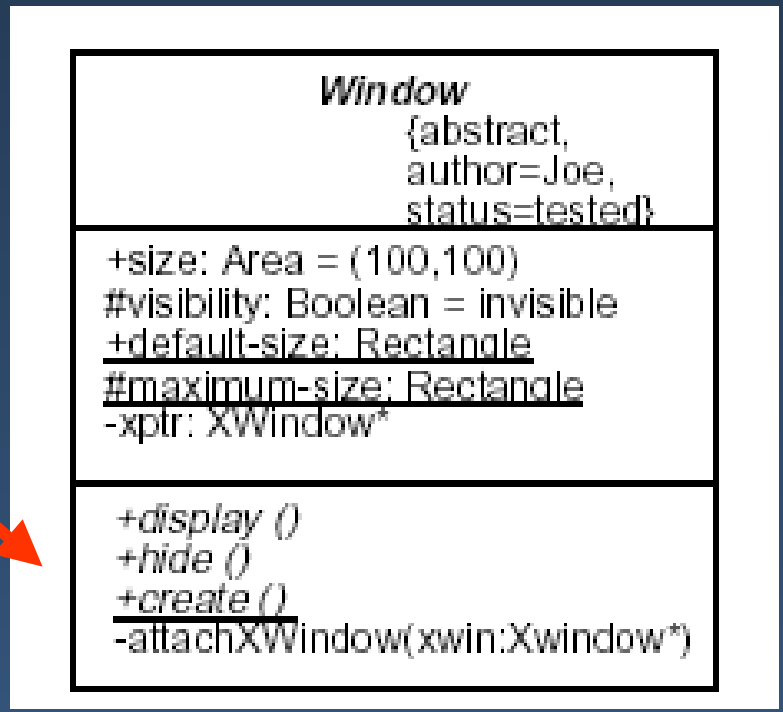
*visibility name* ( *parameter-list* ) : *return-type-expression* { *property-string* }

where *visibility* is one of:

+ public visibility

# protected visibility

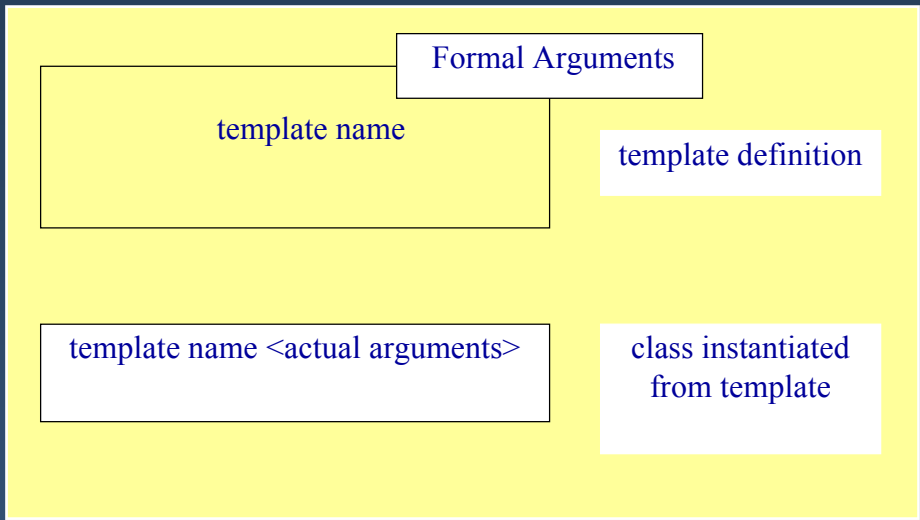
- private visibility

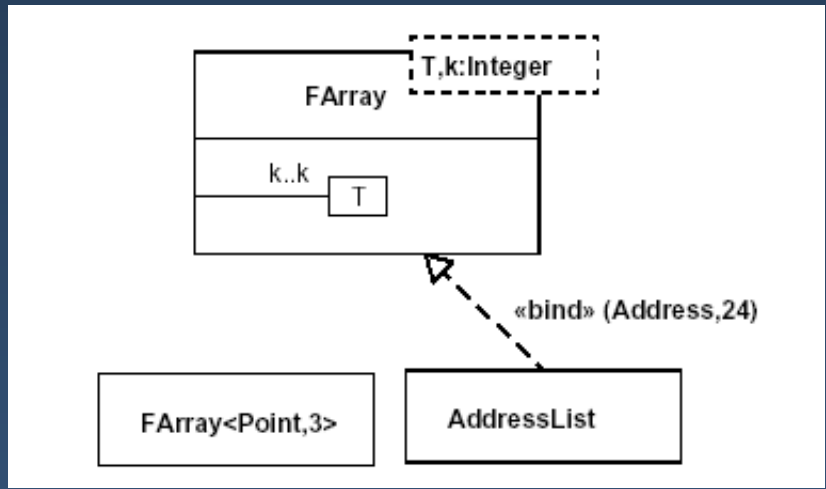


# Parameterised class (template)

A template is the descriptor for a class with one or more unbound formal parameters.

It defines a family of classes, each class specified by binding the parameters to actual values.



# Relationships

Connections between classes

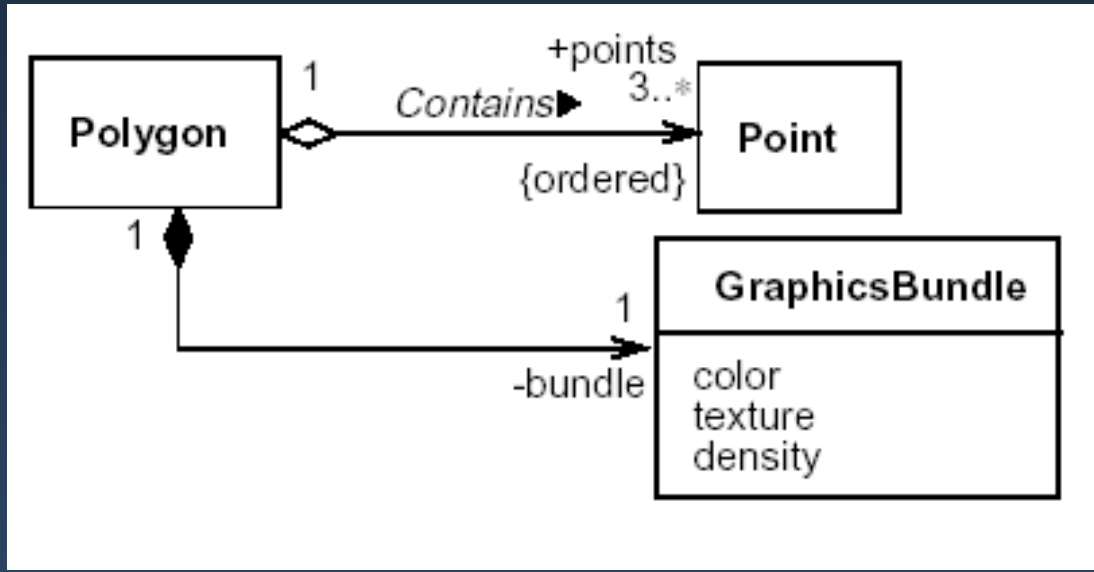
- dependency
- generalization
- association

# Association

An association is a structural relationship within which classes or objects are connected to each other

# Association adornements

- name
- role
- multiplicity
- aggregation
- composition




A hollow diamond is attached to the end of the path to indicate aggregation.

If the diamond is filled, then it signifies the strong form of aggregation known as *composition*.

# Association Name

Describes the nature of relationship:


Can also show the direction to read name:

# Association Roles

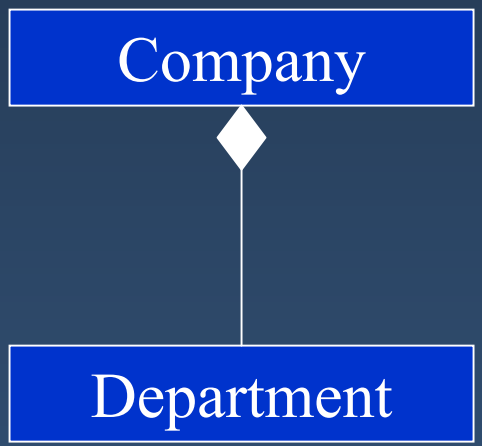
- describe “faces” that classes present to each other within association
- class can play same or different roles within different associations

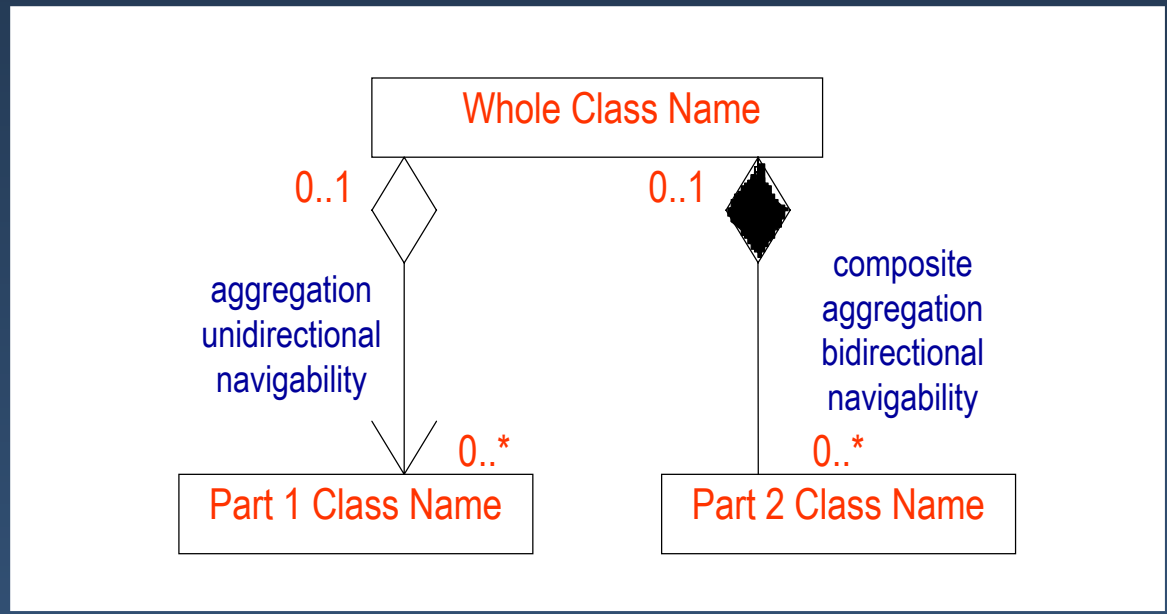
# Association Multiplicity

- possible values same as for classes: explicit value, range, or * for “many”
- Example: a Person is employed by one Company; a Company employs one or more Persons

# Aggregation

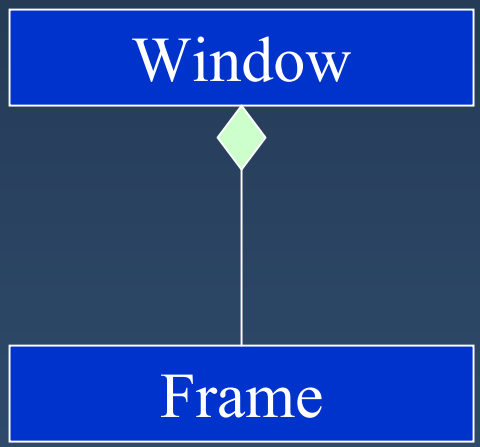
Aggregation is a “whole/part” or “has a” relationship within which one class represents a larger thing that consists of smaller things

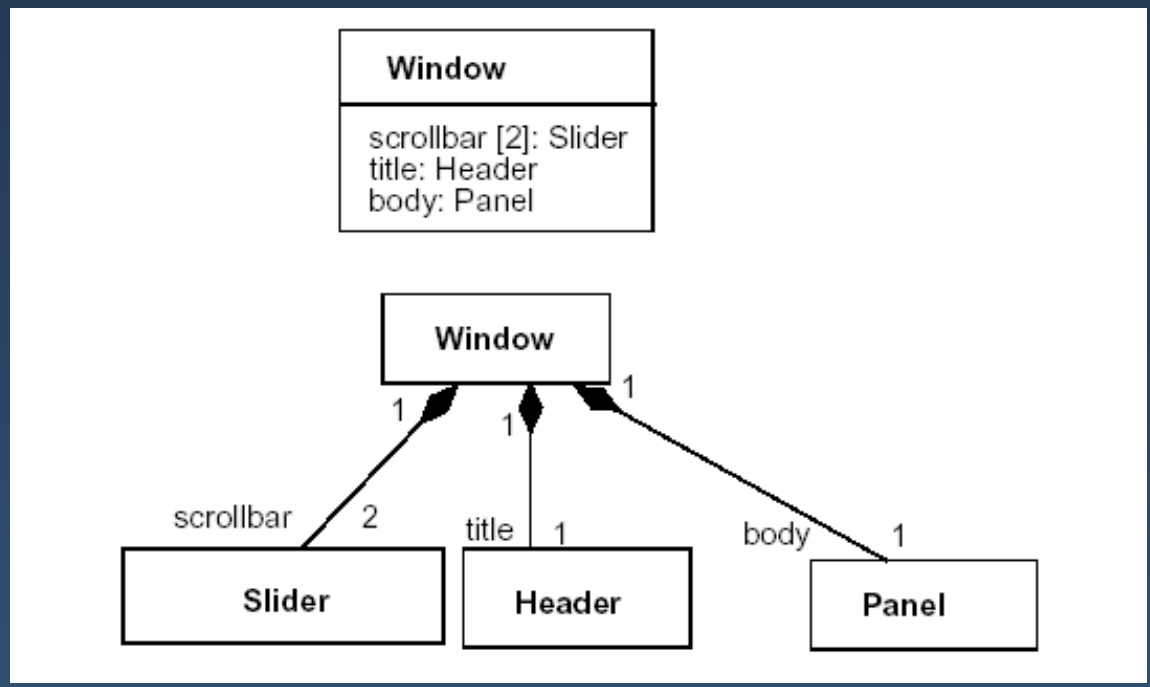


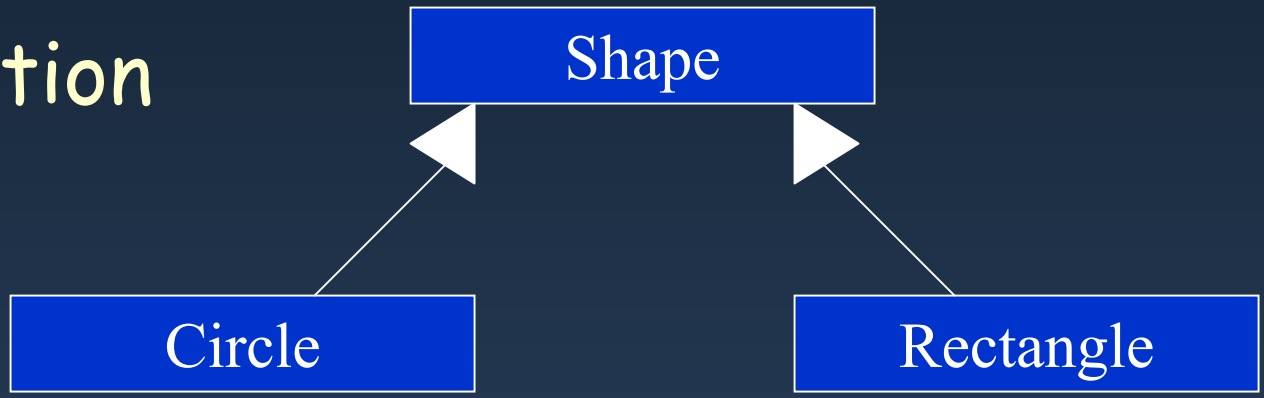
# Composition

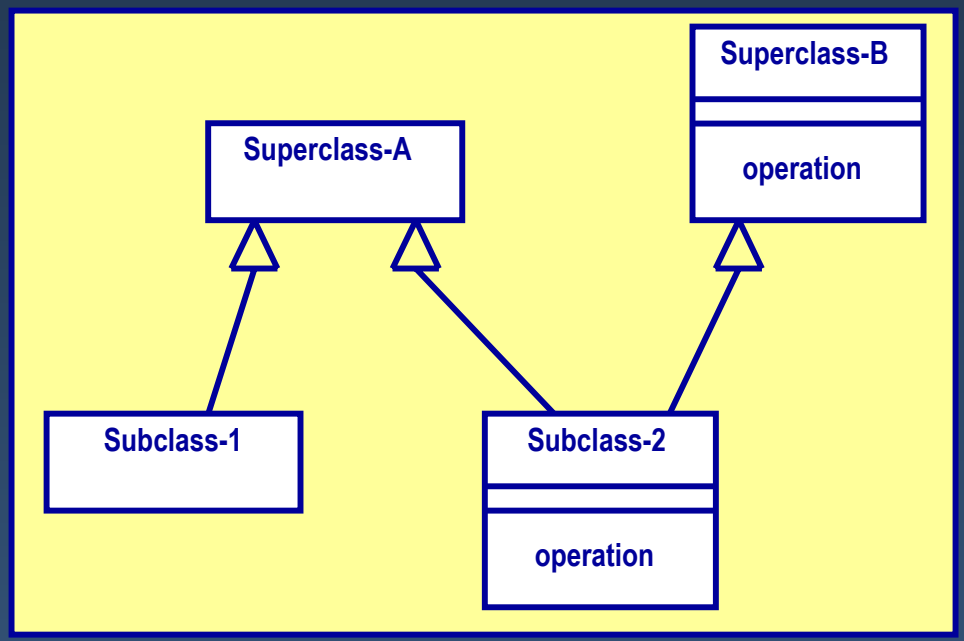
Composition is a form of aggregation with strong ownership and coincident lifetime of part with the whole



# Generalization



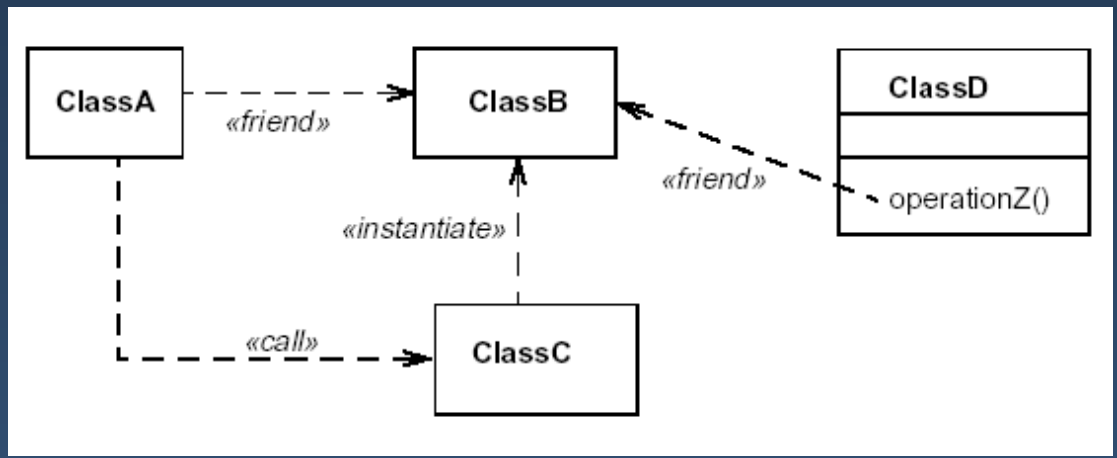




A generalization is a **"kind of"** or **"is a"** relationship between a general thing *(superclass or parent)* and a more specific thing *(subclass or child)*

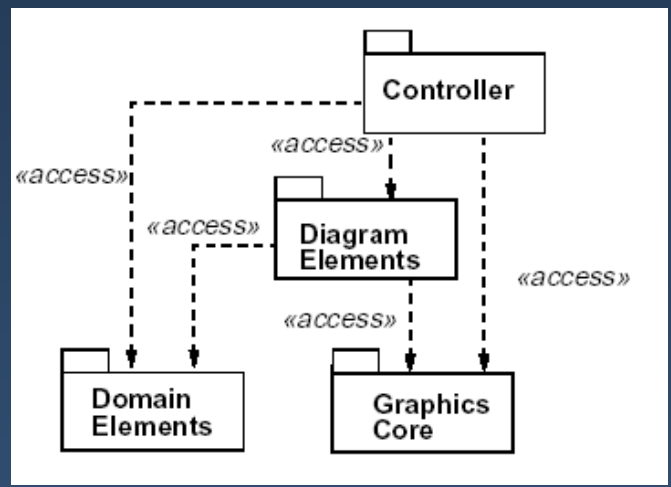
# Dependency

A dependency indicates a semantic relationship between two (or more) model elements

It indicates a situation in which a change to the target element may require a change to the source element in the dependency




among classes

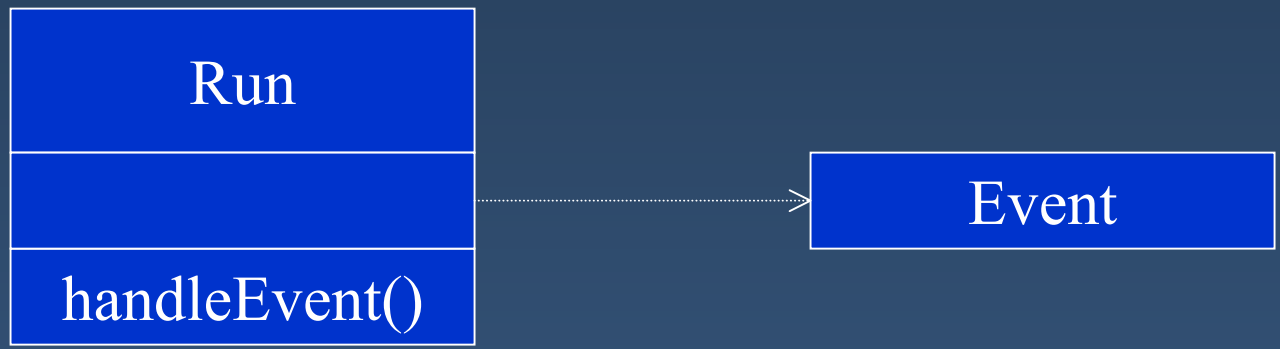



among packages

# Dependency

A dependency is a “using” relationship within which the change in the specification of one class may affect another class that uses it

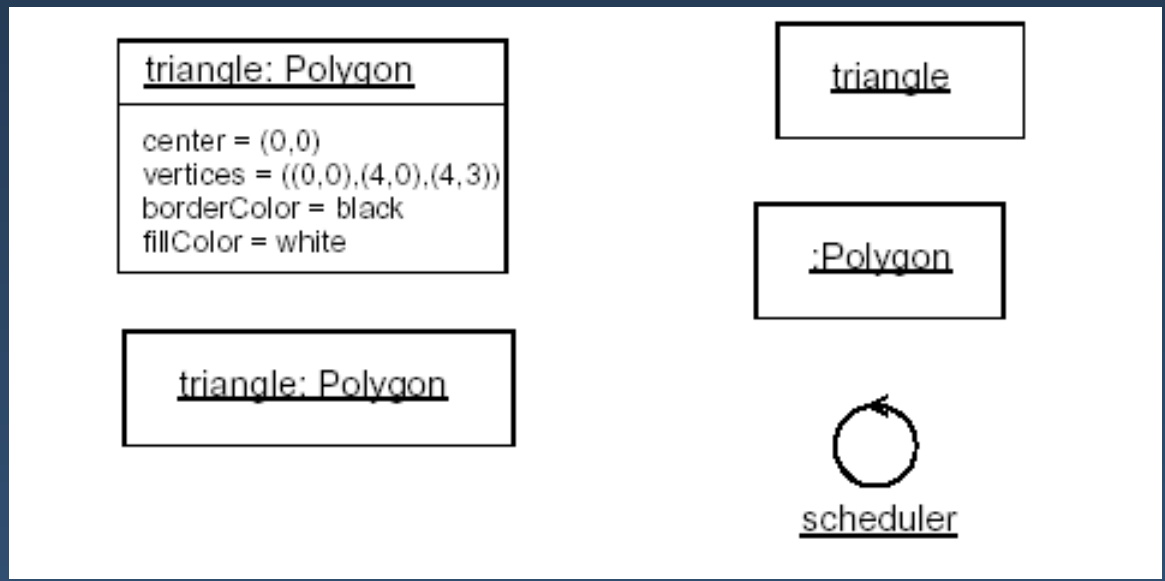
Example: one class uses another in operation



# Objects

An object represents a particular instance of a class

It has identity and attribute values



# Object Diagram

An object diagram shows a set of objects, and their relationships, at a specific point in time