Object Oriented Analysis & Design

Ajit K Nayak

Department of Computer Science & Engineering

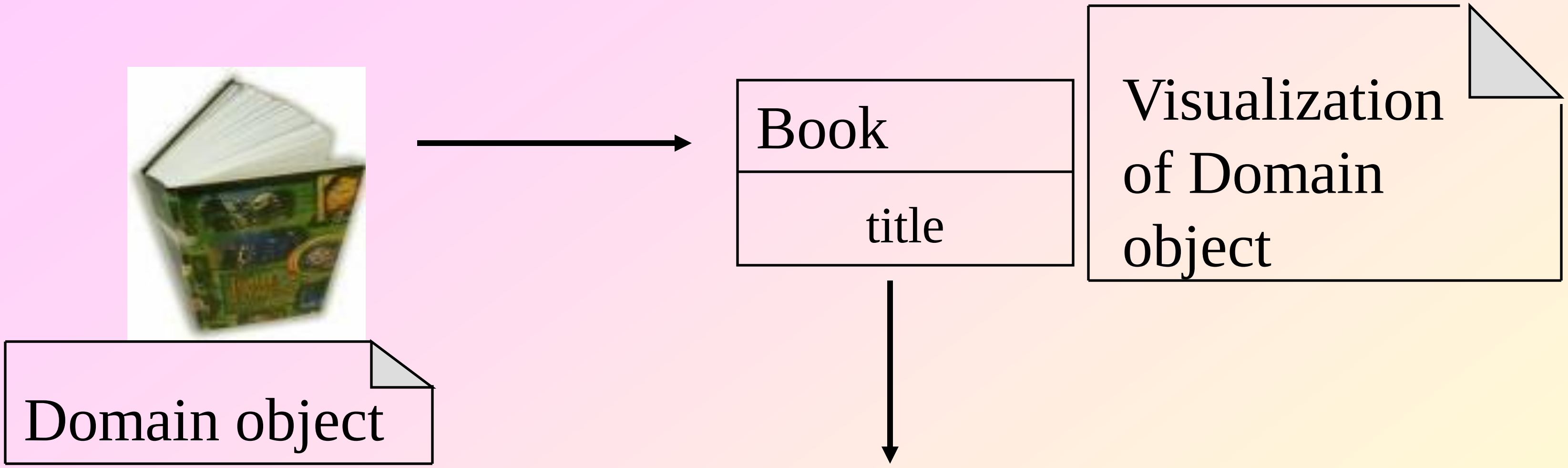
Silicon Institute of Technology, Odisha, India

Analysis & Design

- Analysis emphasizes an investigation of the problem and requirements
- i.e. *requirement analysis* is an investigation of the requirements and *object analysis* is an investigation of domain objects
- Design emphasizes a conceptual solution that fulfills requirements, ultimately the design may be implemented.

OO analysis & Design

- During object oriented analysis emphasizes on finding and describing the objects in the problem domain
- Examples: in case of library information system, some of the objects include Book, Library etc.
- During Object Oriented Design emphasizes on defining software objects and how they collaborate to fulfill the requirements
- Example: in the library system, a Book software object may have a title attribute and a getChapter () method
- Finally, during implementation the objects are implemented, such as a Book class in C++



Book

title

Visualization of Domain object

Domain object

Implementation

```
class Book{
    String title;
    public:
    chapter getChapter(int);
};
```

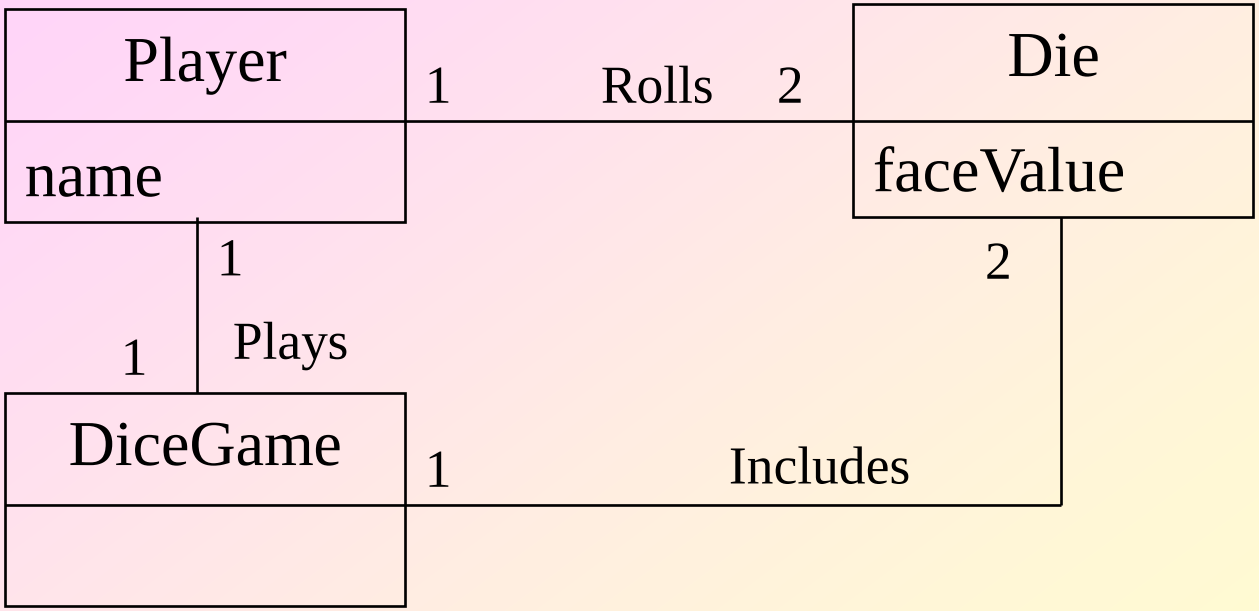
OOAD

- The steps for object oriented analysis and design may be as follows
  - Define Usecases
  - Define domain model
  - Define interaction diagram
  - Define design class diagram

Example

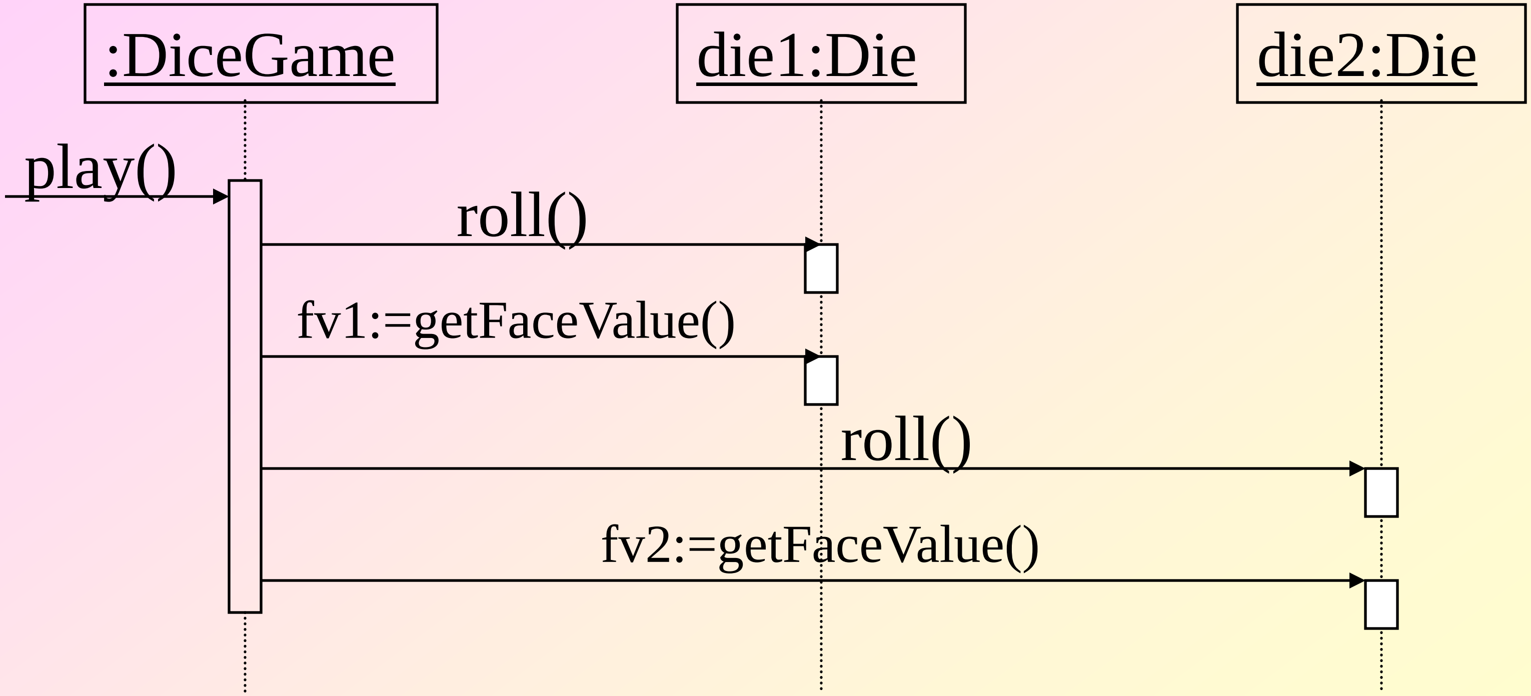
- A *“Dice Game” : A player rolls two die*
- *Usecase: description of related domain process*
  - *Play a Dice Game: A player picks up and rolls the dice. If the dice face value total seven, they win; otherwise, they lose*
- Define domain model: Decomposition of the domain involves identification of objects, attributes and associations.
  - Represented in a diagram

Defining Domain model



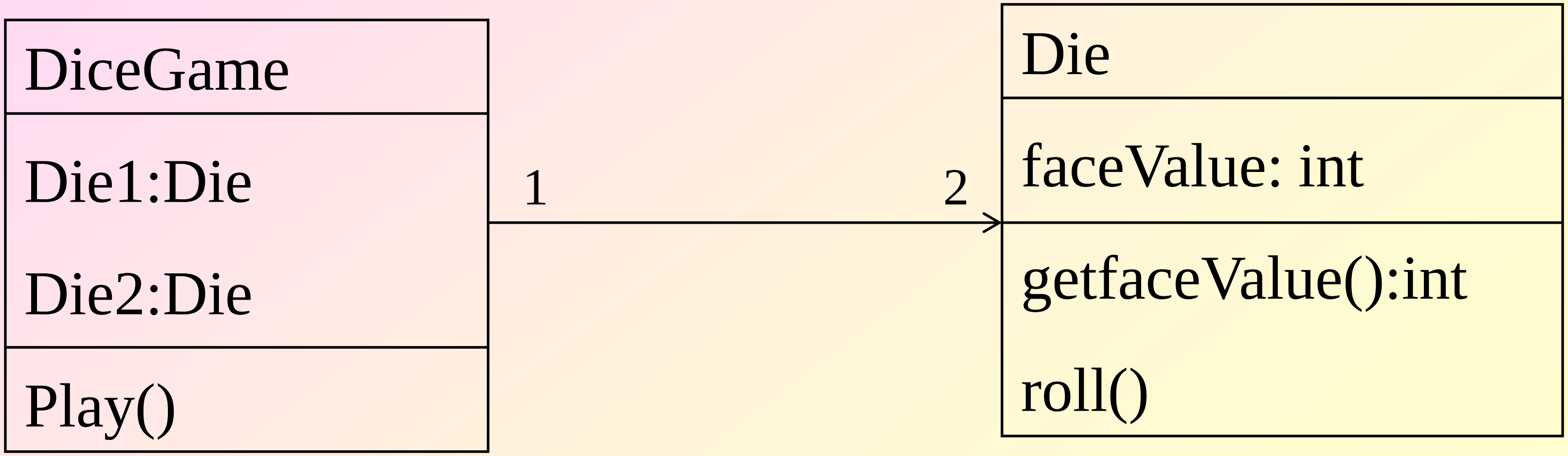
Define Interaction diagram

- Defining software objects and their collaborations.
- This diagram shows the flow of message between software objects and thus invocation of methods



Define design class diagrams

- To create a static view of the class definition with a design class diagram.
- This illustrates the attributes and methods of the classes.



- To represent the analysis and design of object oriented software systems, we use a language called Unified Modeling Language.

UML

- The Unified Modeling Language™ (UML) is the industry-standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems
- It simplifies the complex process of software design, making a "blueprint" for construction.
- The Unified Modeling Language, UML, and the UML logos are trademarks of the Object Management Group.
- www.omg.org

Purpose of Modeling?

- Developing a model for an industrial-strength software system prior to its construction or renovation is as essential as having a blueprint for large building.
- Good models are essential for communication among project teams and to assure architectural soundness.
- As the complexity of systems increase, so does the importance of good modeling techniques. There are many additional factors of a project's success, but having a rigorous modeling language standard is one essential factor.

References

- Books
  - The Unified Modeling Language, User Guide
    - Booch, Rumbaugh. Jacobson **(TEXT)**
  - The Unified Modeling Language, Reference Manual
    - Booch, Rumbaugh. Jacobson
  - The Unified Software Development Process
    - Booch, Rumbaugh. Jacobson
  - Using UML
    - Rob Pooley et al.
  - UML Distilled
    - Martin Fowler, Kendall Scott
  - Instant UML
    - Pierre-Alain Muller

Building blocks of UML

- Building blocks of UML constitutes
  - Things
  - Relationships
  - Diagrams

Things

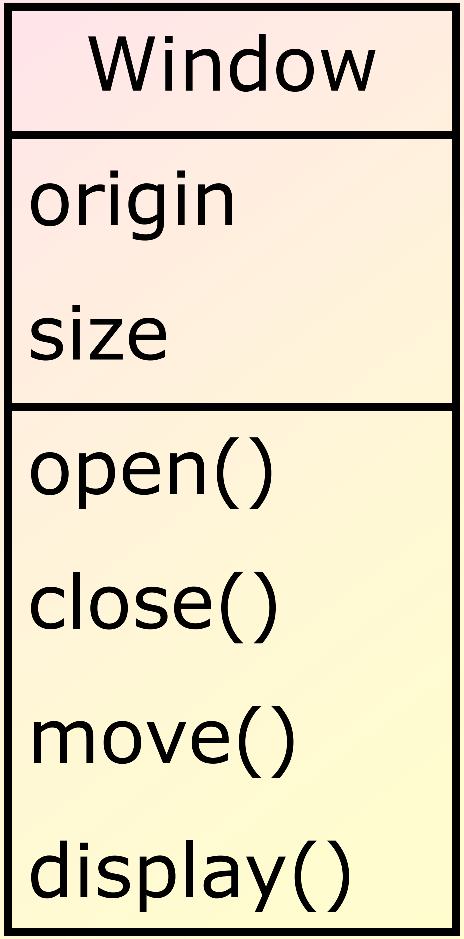
- There are four kind of things defined
  - Structural things
  - Behavioral things
  - Grouping things
  - Annotational things

A. Structural Things

- These are nouns of UML model
- These things represents elements that are either conceptual or physical.
- There are seven kinds of structural things
  - Class
  - Interface
  - Collaboration
  - Use Case
  - Active Class
  - Component
  - Node

Class

- Is a description of a set of objects that share common attributes, operations, relationships, and semantics.
- A class implements one or more instances
- It is represented by a rectangular box



Window

origin

size

open()

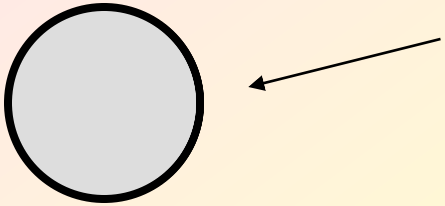
close()

move()

display()

Interface

- Is a collection of operations that specify a service of a class
- An interface might represent the complete behaviour of a class or only a part of that behaviour
- It is represented by a circle, with its name

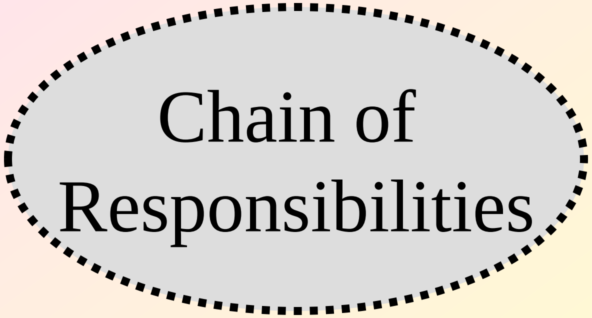


Interface spelling

ISpelling

Collaboration

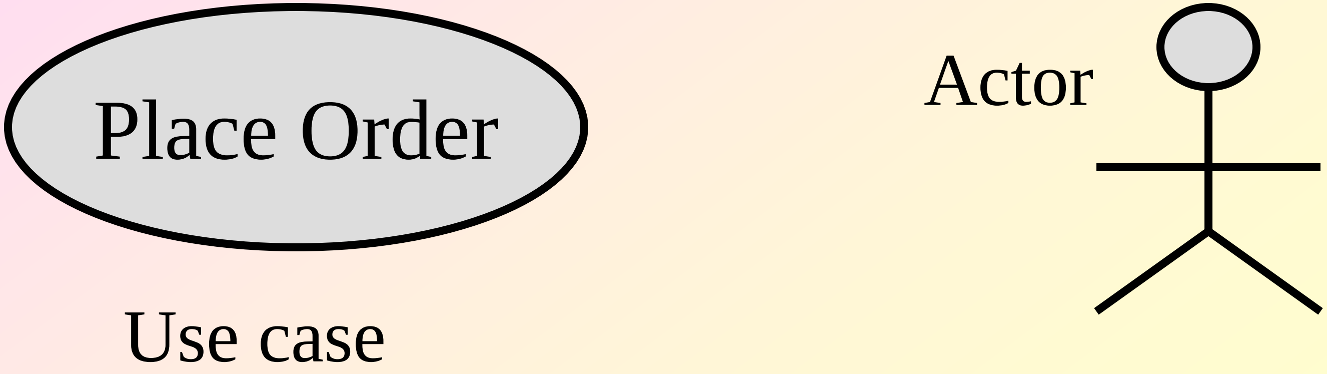
- Defines an interaction and is a society of roles and other elements that work together to provide some cooperative behaviour
- It is represented by an dashed ellipse with its name



Chain of Responsibilities

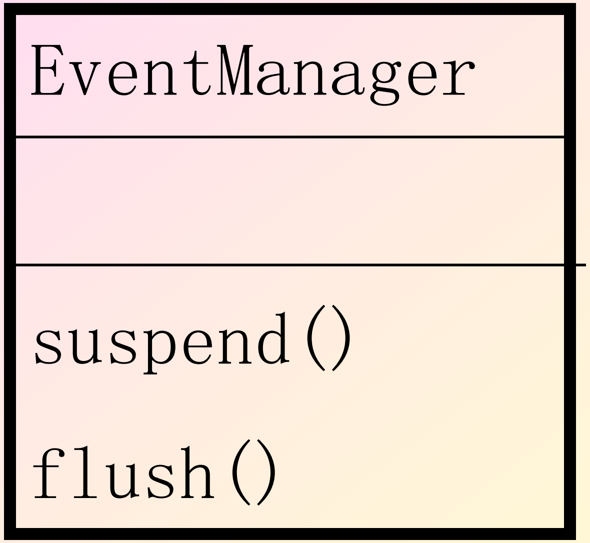
Use Case

- It is a description of sequence of actions that a system performs that yields an observable result of a value to a particular actor
- It is represented by a solid ellipse with its name



Active Class

- It is a class, whose objects own one or more processes or threads and therefore can initiate control activity
- It is represented by a heavy lined box



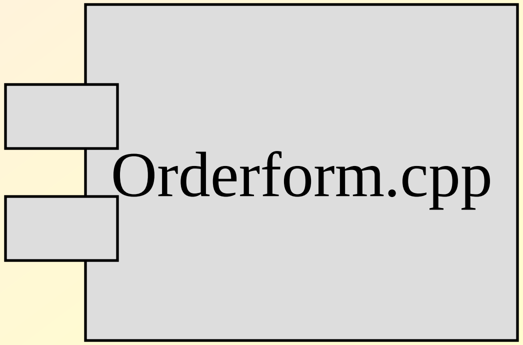
EventManager

suspend()

flush()

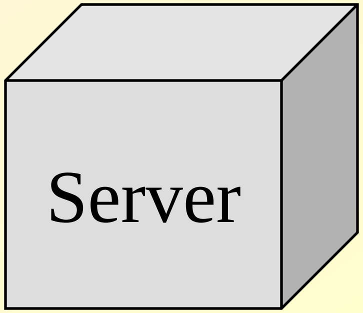
Component

- It is a physical and replaceable part of a system that confirms to and realization of a set of interfaces.
- i.e. physical packaging of classes, interfaces and collaborations



Node

7. It is a physical element that exists in run time and represents a computational resource, generally having some memory and processing capability
8. A set of components may reside as a node and may also migrate from node to node
9. It is represented as a cube

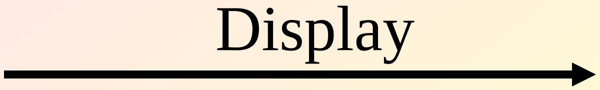


B. Behavioural Things

- These are dynamic parts of UML model
- These are verbs of a model, representing behaviour over time and space
- There are two types of Behavioural things
  - Interaction
  - State Machine

Interaction

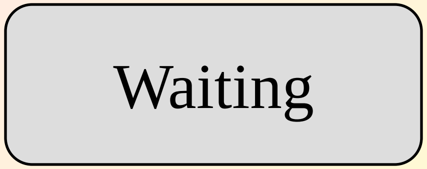
1. It is a behaviour that comprises a set of messages exchanged among a set of objects to accomplish a specific purpose.
2. An interaction involves messages, action sequences and links. (connection between objects)
3. It is represented by a solid arrow



Display

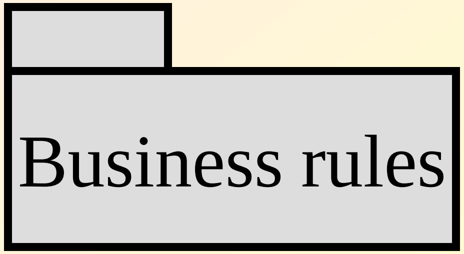
State Machine

- It is a behaviour that specifies the sequence of states of an object or an interaction goes through during its life time in response to events, to gether with its responses to these events.
- A state machine involves
  - States
  - Transitions (flow from one state to another)
  - Events (things that trigger a transition)
  - Activities (response to transitions)
  - It is represented with a rectangular box having rounded corners



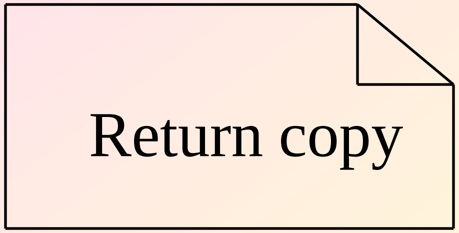
C.Grouping Things

- These are organizational part of UML models
- There is one primary kind of grouping thing called packages
- Package
  - It is a general purpose mechanism for organizing elements in to groups
  - It is rendered as a tabbed folder



D. Annotational Things

- These are explanation parts of UML
- Note is a annotational thing called
- Note
  - It is simply a symbol for rendering constraints and comments attached to an element or a collection of elements.



Some explanatory text

Relationships

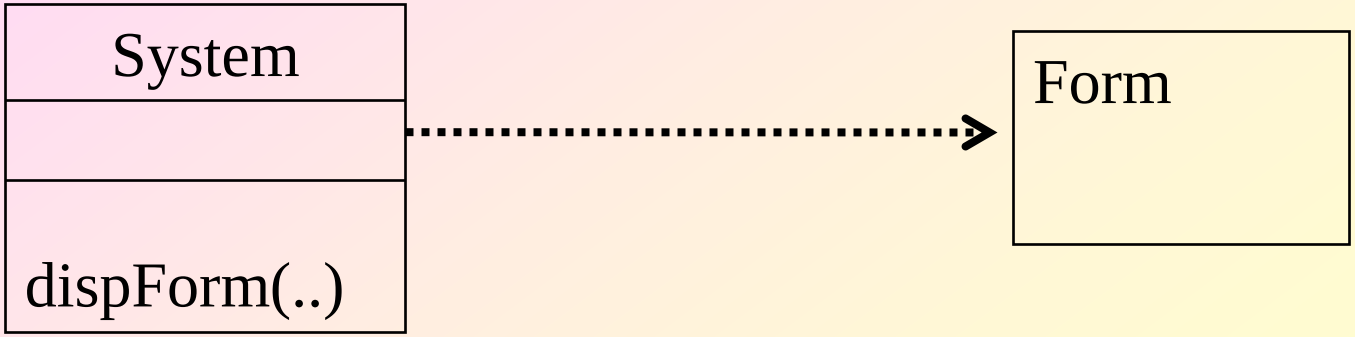
- A relationship is a connection among things
- It is rendered as a path, with different kind of lines used to distinguish the kind of relationships

Relationships

- There are 4 kinds of relationships
  - Dependency
  - Association
  - Generalization
  - Realization

A. Dependency

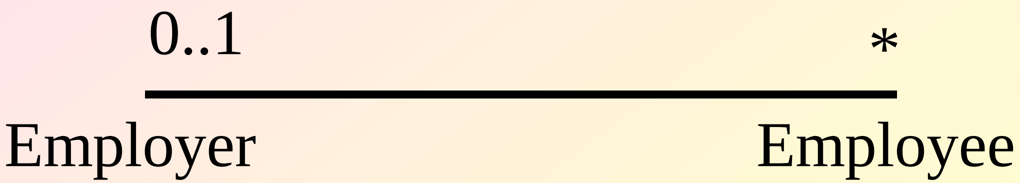
- It is a semantic relationship between two things in which a change to independent thing may affect the semantic of the other thing
- It is rendered as a directed dashed line



- The form the system displays obviously depends on which form the user selects

B. Association

- It describes a set of links
  - **Aggregation** is a special kind of association, representing a structural relationship between a whole and its parts
  - It is rendered as a solid line, occasionally including a label and other specifications



0..1

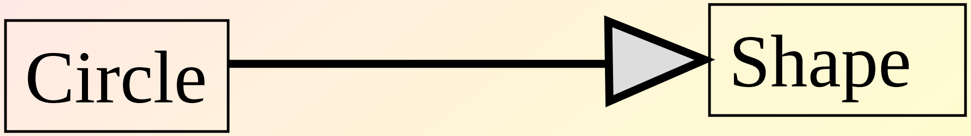
*

Employer

Employee

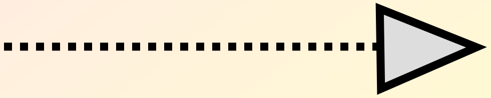
C. Generalization/Specialization

- Child shares the structure and behaviour of the parent
  - ✓ Objects of specialized elements are substitutable for the objects of generalized elements
  - ✓ It is rendered as a solid line with a hollow arrow head pointing to the parent



D. Realization

- It is a relationship between classifiers. (polymorphism)
  - ✓ One classifier specifies a contract that another classifier guarantees to carry out
  - ✓ It is rendered as a dashed line with a hollow arrow head.
  - ✓ The base class provides an interface for which the derived class writes an interface



Diagrams

- A Graphical representation of a set of elements, most often rendered as a connected graph of vertices (things) and arcs (relationships).

Diagrams

- There are 9 types of Diagrams
  - Class Diagram
    - Shows a set of classes, interfaces, and collaborations and their relationships.
  - Object Diagram
    - Shows a set of objects and their relationships
  - Use Case Diagram
    - Shows a set of use cases and actors
  - Sequence Diagram & Collaboration Diagram
    - These two are interaction kind of diagram. They show a set of objects and their relationships

Diagrams

- Statechart Diagram
  - Shows a state machine, consisting of states, transitions, events and activities
- Activity Diagram
  - It is a kind of statechart diagram that shows the flow from activity to activity with a system
- Component Diagram
  - Shows the organisation and dependencies among a set of componets. It addresses the static implementation views of a system
- Deployment Diagram
  - Shows the configuration of run-time processing nodes and the components that live on them.

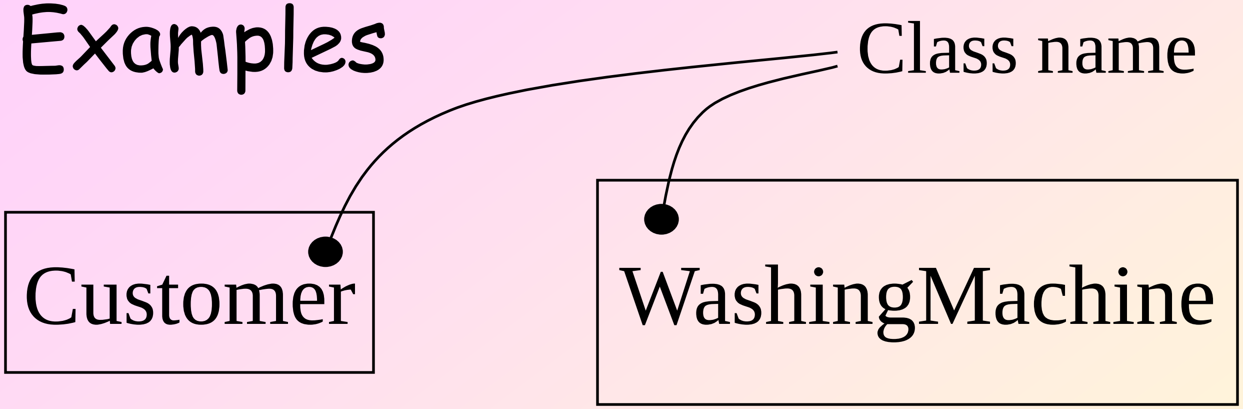
Classes

- A class is a descriptions of a set of objects, that share attributes, relationships and semantics.
- Graphically a class is rendered as a rectangle
- Name of a class
  - In practice class names are short nouns or noun phrases drawn from the vocabulary of the system to be modeled
  - Every class has a name that distinguishes it from other classes

Name of a class

- – Name is a textual string, with first letter of every word capitalized

- Examples



Class name

Customer

WashingMachine

•A path name is the class name prefixed by the name of the package in which that class lives

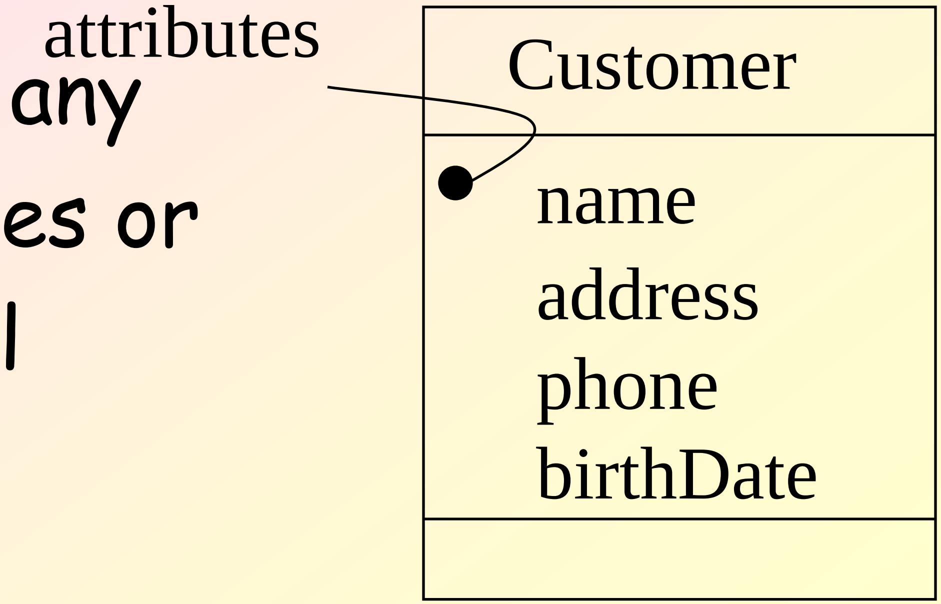
HouseholdAppliances : : WashingMachine

Attributes of a Class

- An attribute is an abstraction of the kind of data or state an object of the class might encompass
- An attribute is a named property of a class
- A class may have any

number of attributes or

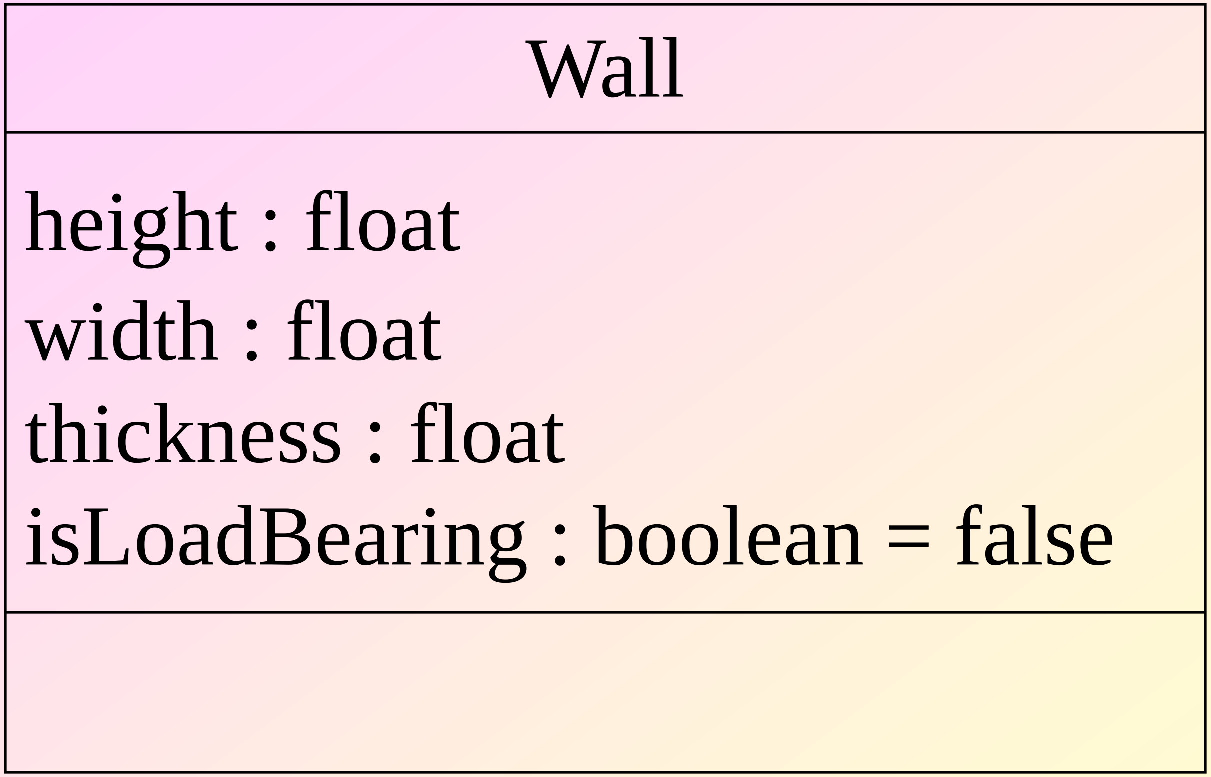
no attributes at all



Attributes of a Class

- Attributes are listed in a compartment just below the class name
- First letter of every word is capital in an attribute name expect the first word
- An attribute may be specified by stating its class and possibly default initial value
- In practice an attribute name is a short noun or noun Phrase that represents some property of its enclosing class

Attribute Example



Wall

height : float
width : float
thickness : float
isLoadBearing : boolean = false

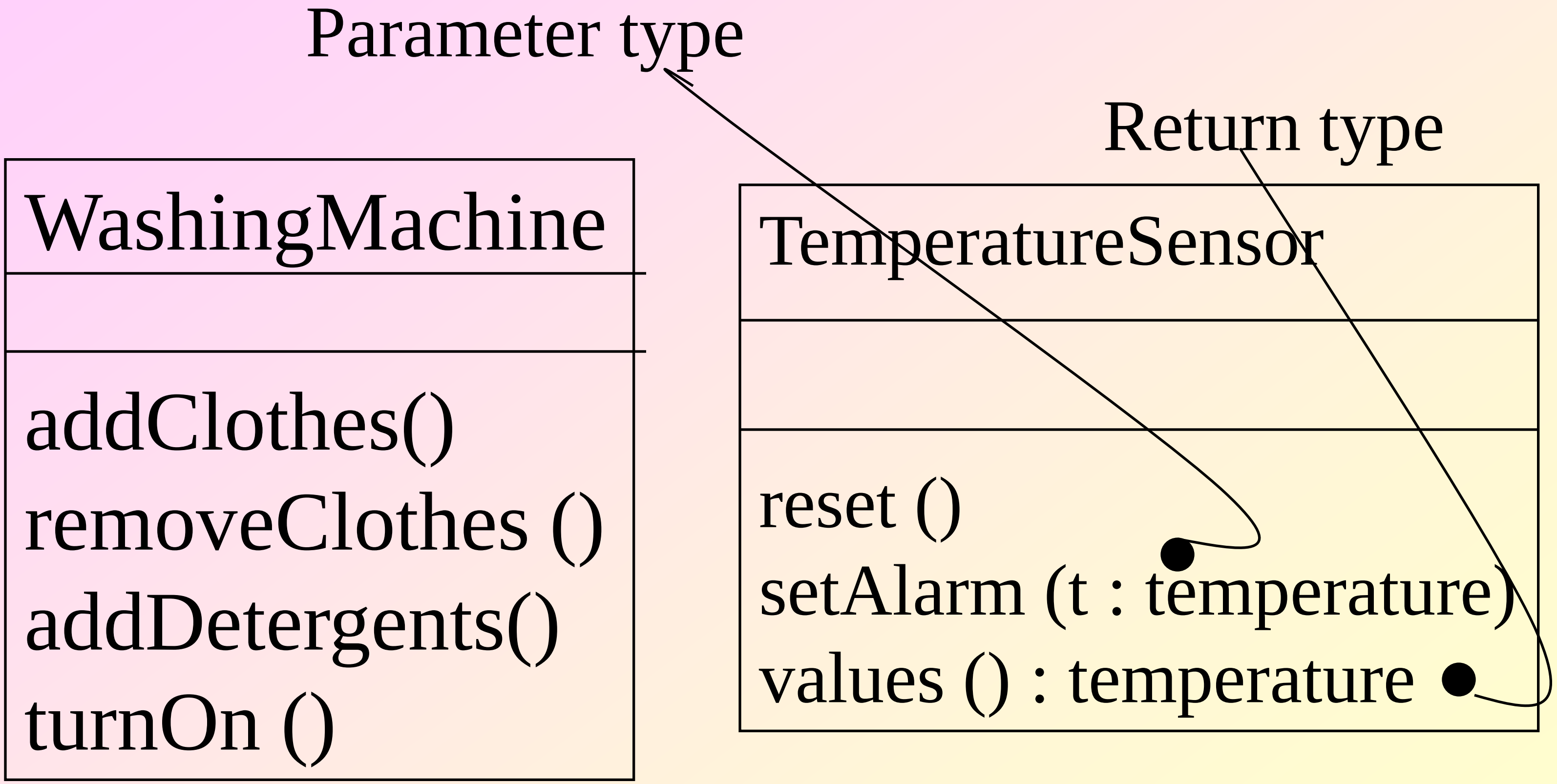
Operations in a Class

- An operation is an abstraction of something, one can do to at an object and that is shared by all objects of that class
- A class may have any no. of operations or no operations at all
- Operation are listed in a Compartment just below The class attribute

Operations in a Class

- The first letter of every word in an operations name is capital, except the first letter of first word
- An operations can be specified By stating its signature, covering the name, type and default values of all parameters and a return type
- In practice, an operation name is a short verb or verb phrase that represents some behavior of its enclosing class

Operations Example



Parameter type

Return type

WashingMachine

addClothes()
removeClothes ()
addDetergents()
turnOn ()

TemperatureSensor

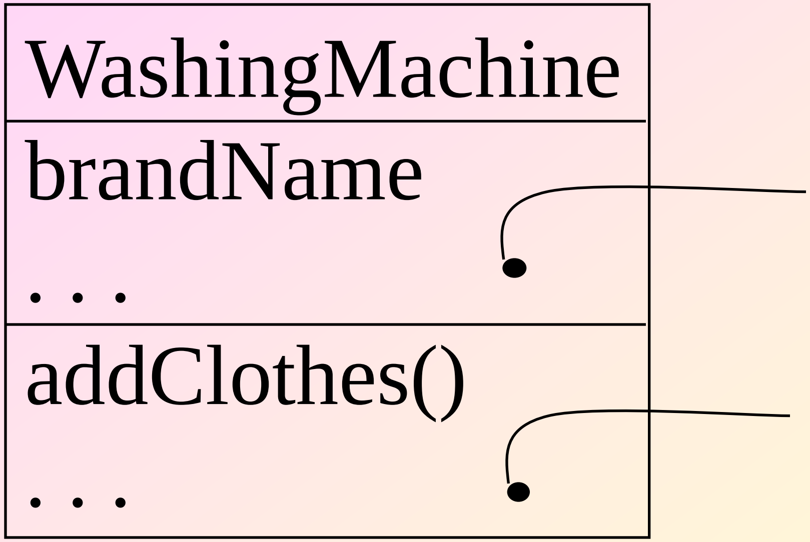
reset ()
setAlarm (t : temperature)
values () : temperature

Organising Attributes and Operations

- It may not be possible to list all attributes and operations at once
- Therefore, a class can be elided, I.e. it is possible to show only some or none of attributes and operations
- An empty compartment does not necessarily mean that there are no attributes or operations
- It can be explicitly specified that there are more attributes or operations than shown, by ending each list with an ellipsis.

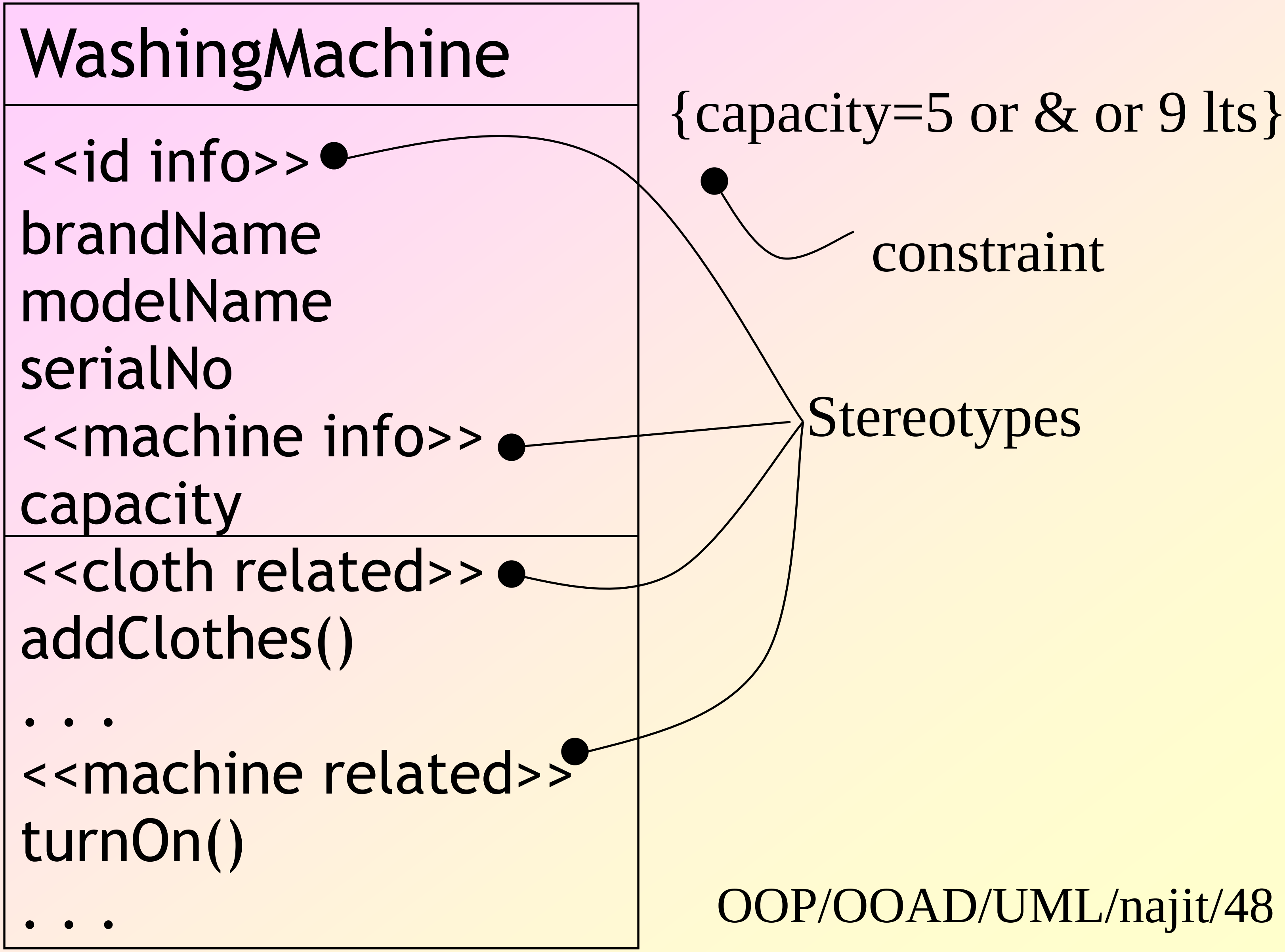
Attributes & Operations

- To better organise long lists of attributes and operations, those can be grouped.
- Each group is prefixed with a descriptive category by using stereotypes



Ellipsis shows there are some more attributes and operations

Attributes & Operations



WashingMachine

<<id info>>
brandName
modelName
serialNo
<<machine info>>
capacity

<<cloth related>>
addClothes()
. . .
<<machine related>>
turnOn()
. . .

{capacity=5 or & or 9 lts}

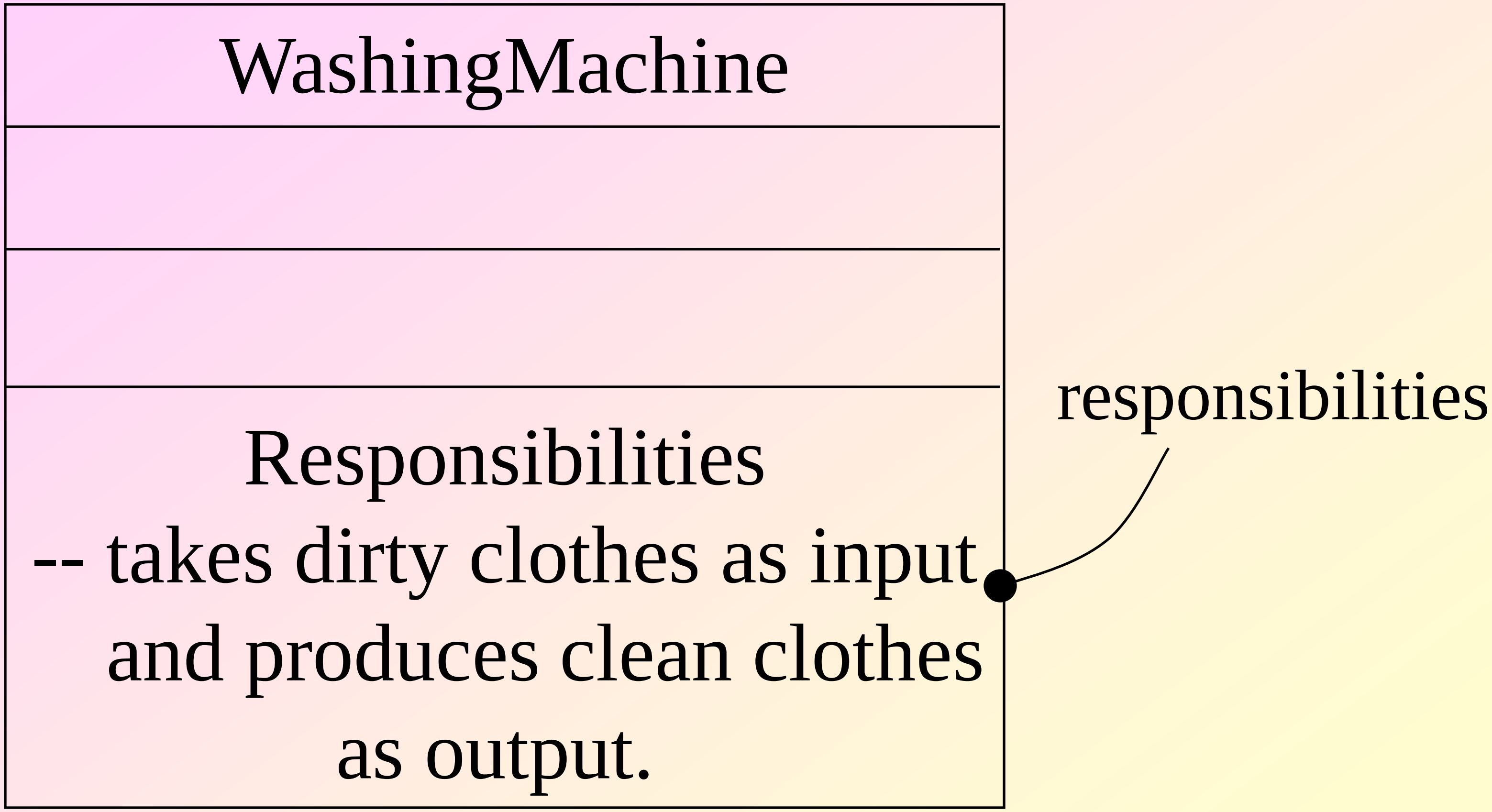
constraint

Stereotypes

Responsibilities

- It is a contract or an obligation of a class. i.e it is a description of what the class has to do
- Responsibilities can be drawn in a separate compartment at the bottom of the class icon
- A single responsibility is written as a phrase, a sentence, or (at most) a short paragraph (free-form text)

Responsibilities



WashingMachine

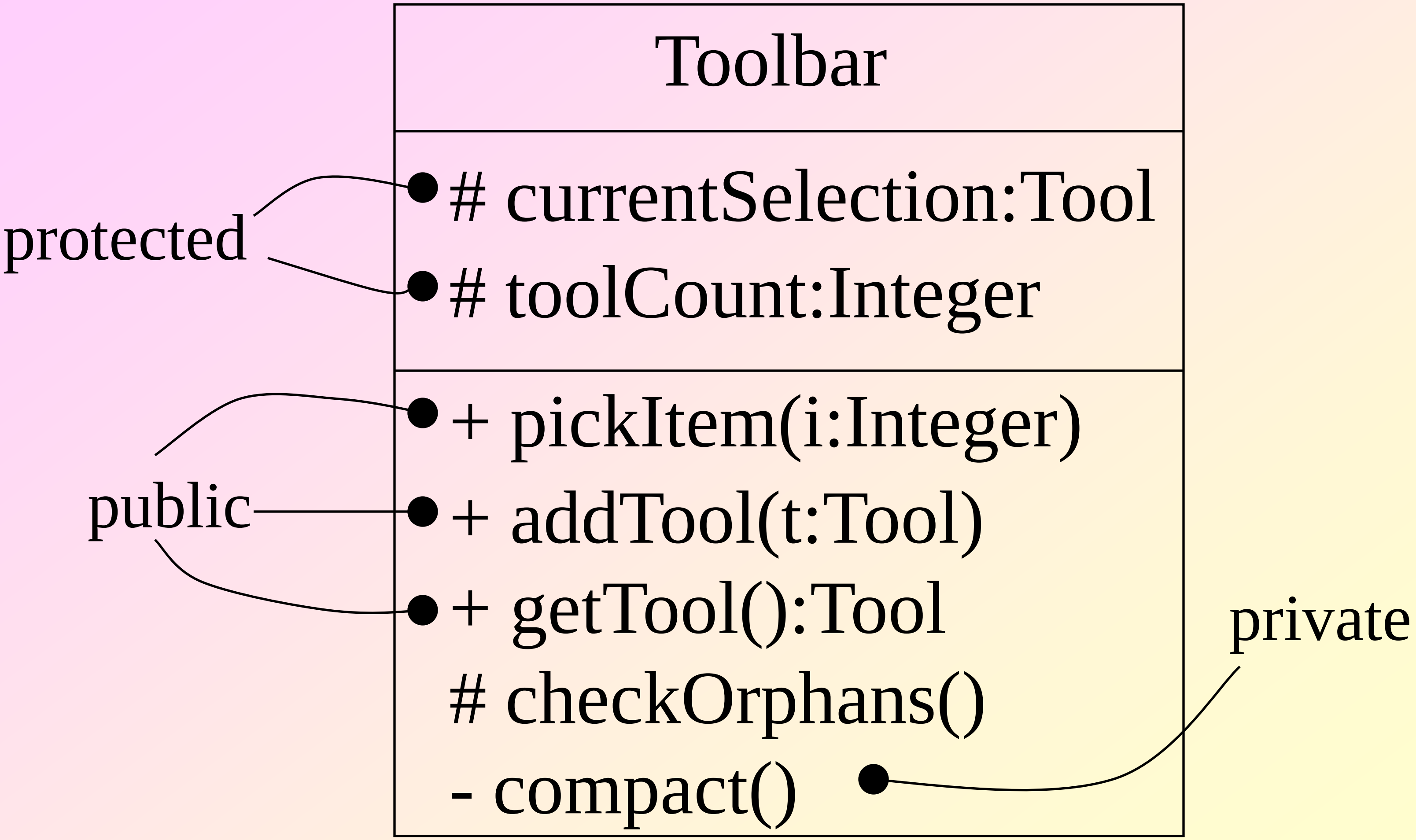
Responsibilities
-- takes dirty clothes as input
and produces clean clothes
as output.

responsibilities

Visibility of a feature

- It allows to specify the visibility for a classifier's features.
- Three levels of visibility can be specified using UML
  - Public : any outside classifier with visibility to the given classifier can use the feature; specified by prepending the symbol +
  - Protected : Any descendant of the classifier can use the feature; specified by prepending the symbol #
  - Private : only the classifier itself can use the feature; specified by prepending the symbol -

Example Visibility



Toolbar

# currentSelection:Tool

# toolCount:Integer

+ pickItem(i:Integer)

+ addTool(t:Tool)

+ getTool():Tool

# checkOrphans()

- compact()

protected

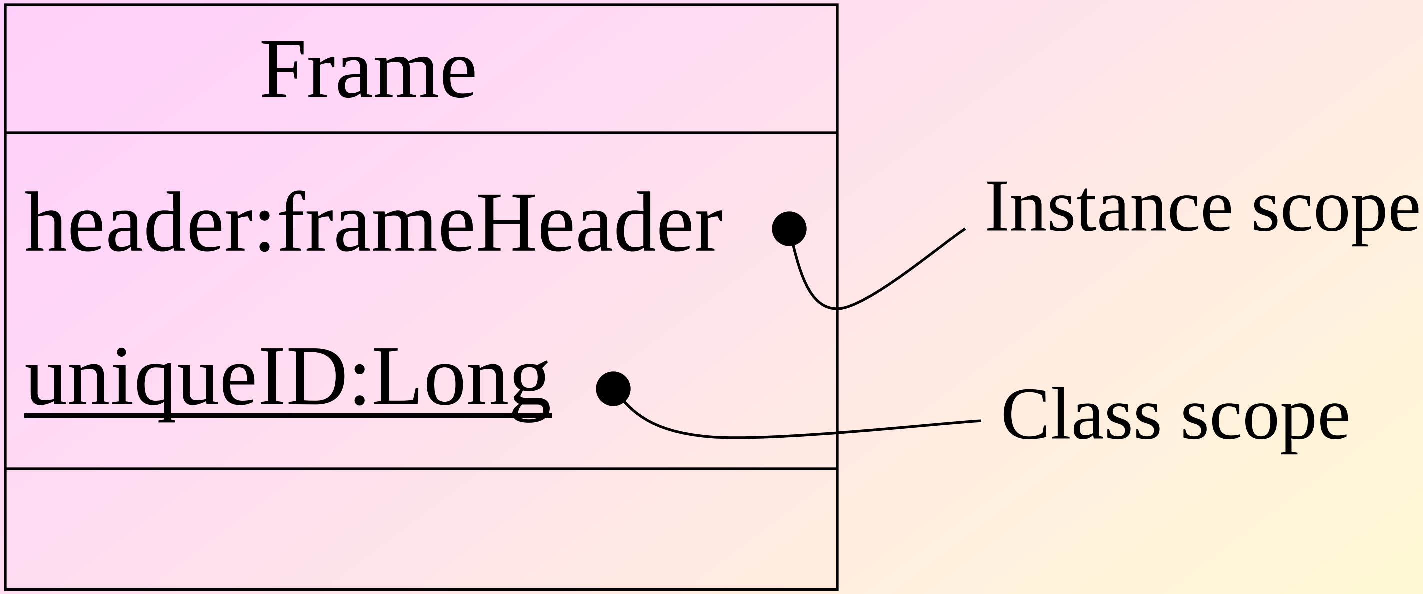
public

private

Scope of a feature

- The owner scope of a feature specifies whether the feature appears in each instance of the classifier or whether there is just a single instance for all instances of the classifier
- UML provides two kinds of scope
  - Instances : Each instance of the classifier holds its own value for the feature
  - Classifier : There is just one value of the feature for all instances of the classifier.

Scope (Contd.)


Frame

header:frameHeader

uniqueID:Long

Instance scope

Class scope

- Classifier scope is rendered by underlining the feature
- In C++ we call the class scoped as static members of a class

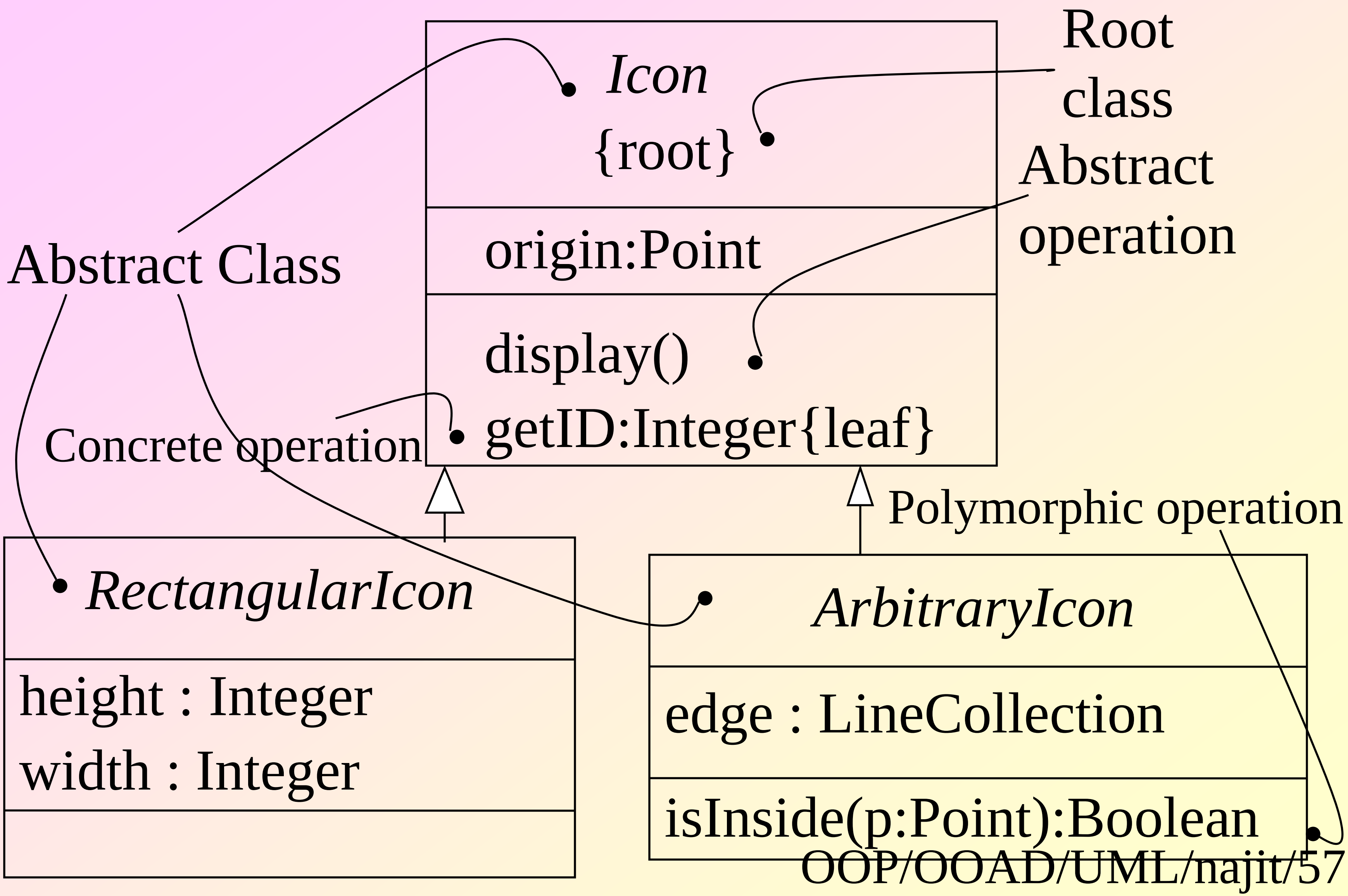
Polymorphic Elements

- Abstarct Class
  - Class having no direct instances (italics)
- Concrete Class
  - Class that may have direct instances
- Root Class
  - Class having no parent but may have children {root}
- Leaf Class
  - Class having parent but no children {leaf}

Polymorphic Elements

- Polymorphic Operations
  - An operation is polymorphic, if it can be specified with the same signature at different points of hierarchy
  - When a message is dispatched at run time, the operation in the hierarchy that is invoked is chosen polymorphycally. i.e. a match is determined at run time according to the type of the object
- Abstract Operation
  - This operation is incomplete and requires a child to supply an implementation. In UML it is specified by writing its name in italics.
- Leaf operation
  - Operation is not polymorphic and may not be overridden

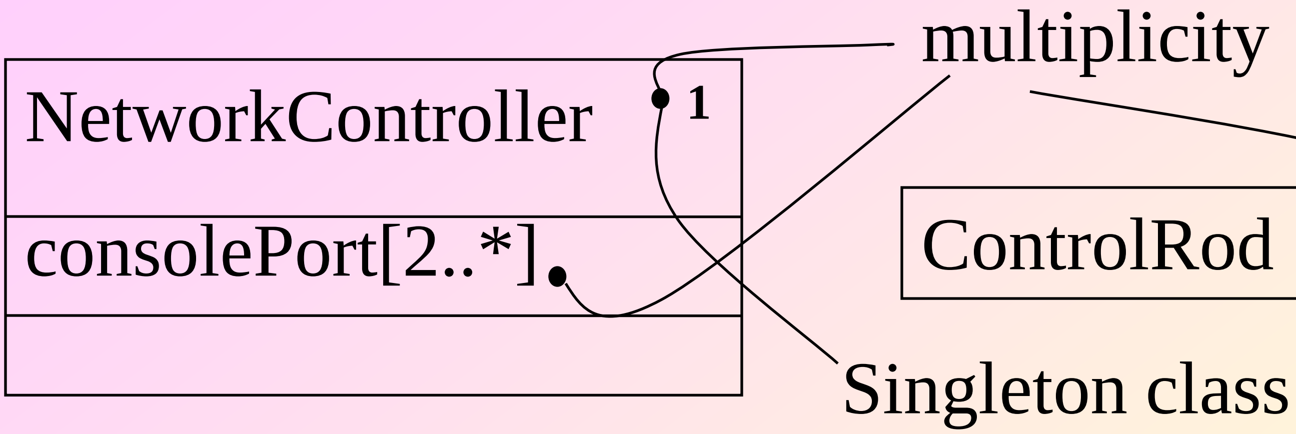
Example Polymorphism



Multiplicity

- It is a specification of the range of allowable cardinalities an entity may assume
- Multiplicity of a class
  - It is used to restrict the no of instances of a class
  - i.e. It tells the no of instances a class may have
- Multiplicity of an attribute
  - It is used to represent a set of elements for an attribute
  - i.e. It maps to representing an array variable in C++

Example Multiplicity



Notes:

- Abstract operations maps to pure virtual functions in C++
- Leaf operations maps to non-virtual functions in C++

Attributes

- In its full form, the syntax of an attribute in UML is

[visibility] name [multiplicity] [: type]

[= initial-value] [{property-string}]

Examples :

origin	Name Only
+origin	Visibility and name
origin : Point	Name and type
head : *Item	Name and complex type
name [0..1] : String	Name, multiplycity, and type

Attributes

- There are three defined properties, that can be used with attributes
  - changeable
    - There are no restrictions on modifying the attribute value
  - addonly
    - For attributes with a multiplicity greater than one, but once created, a value may not be removed or altered
  - frozen
    - The attribute’s value may not be changed after the object is initialized(constants)

Operations

- In its full form, the syntax of an operation in UML is

[visibility] name [(parameter-list)] [: return-type]
[{property-string}]

Examples

display	Name Only
+display	Visibility and name
set (n : Name, s : String)	Name and parameters
getID ( ) : Integer	Name and return type
restart () {guarded}	Nameand property

Operation Parameters

- An operation again can be provided with zero or more parameters with following syntax

[direction] name : type [= default-value]

- Direction may be any of the following
  - in : An input parameter, may not be modified
  - out : An output parameter, may be modified to communicate information to the caller
  - inout : An input parameter, may be modified

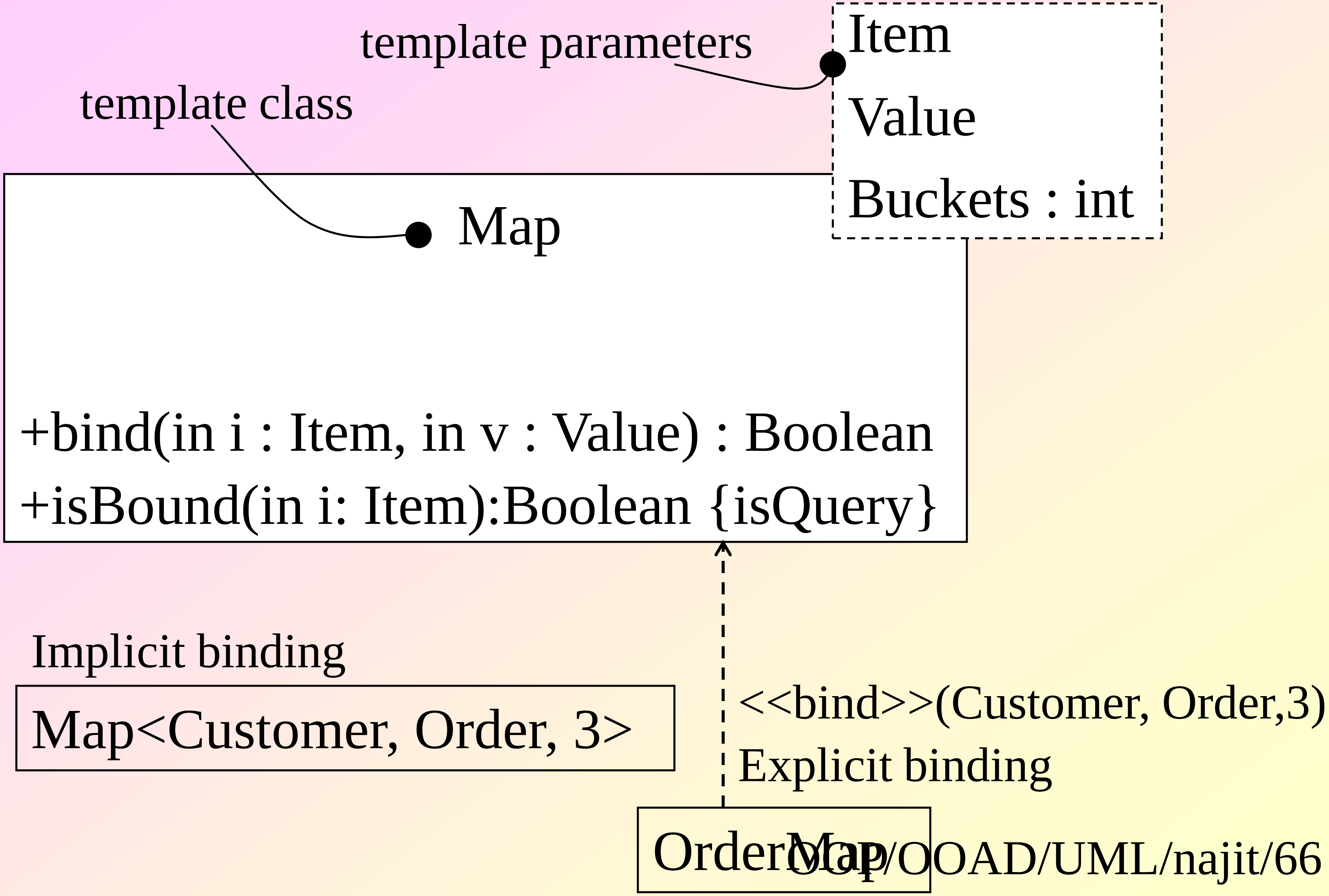
Operation Properties

- In addition to leaf property described earlier there are four other properties defined as follows
  - isQuery : It is a pure function that has no side effects (states unchanged)
  - sequential : callers must coordinate outside the object so that only one flow is in the object at a time
  - guarded : integrity of the object is guaranteed in the multiple flow of control by sequentializing all calls
  - concurrent : integrity of the object is guaranteed in the presence of multiple flows of control by treating the operation as atomic

Template Classes

- Template is a parameterized element
- Template class define a family of classes
- A Template includes slots for classes, objects and values, and these slots serve as the template parameter
- Instance of a template class is a concrete class that can be used just like any ordinary class
- The most use of template class is to specify containers that can be instantiated for specific elements.

Template Classes



template parameters

template class

Item

Value

Buckets : int

Map

+bind(in i : Item, in v : Value) : Boolean

+isBound(in i: Item):Boolean {isQuery}

Implicit binding

Map<Customer, Order, 3>

<<bind>>(Customer, Order,3)

Explicit binding

OrderMap

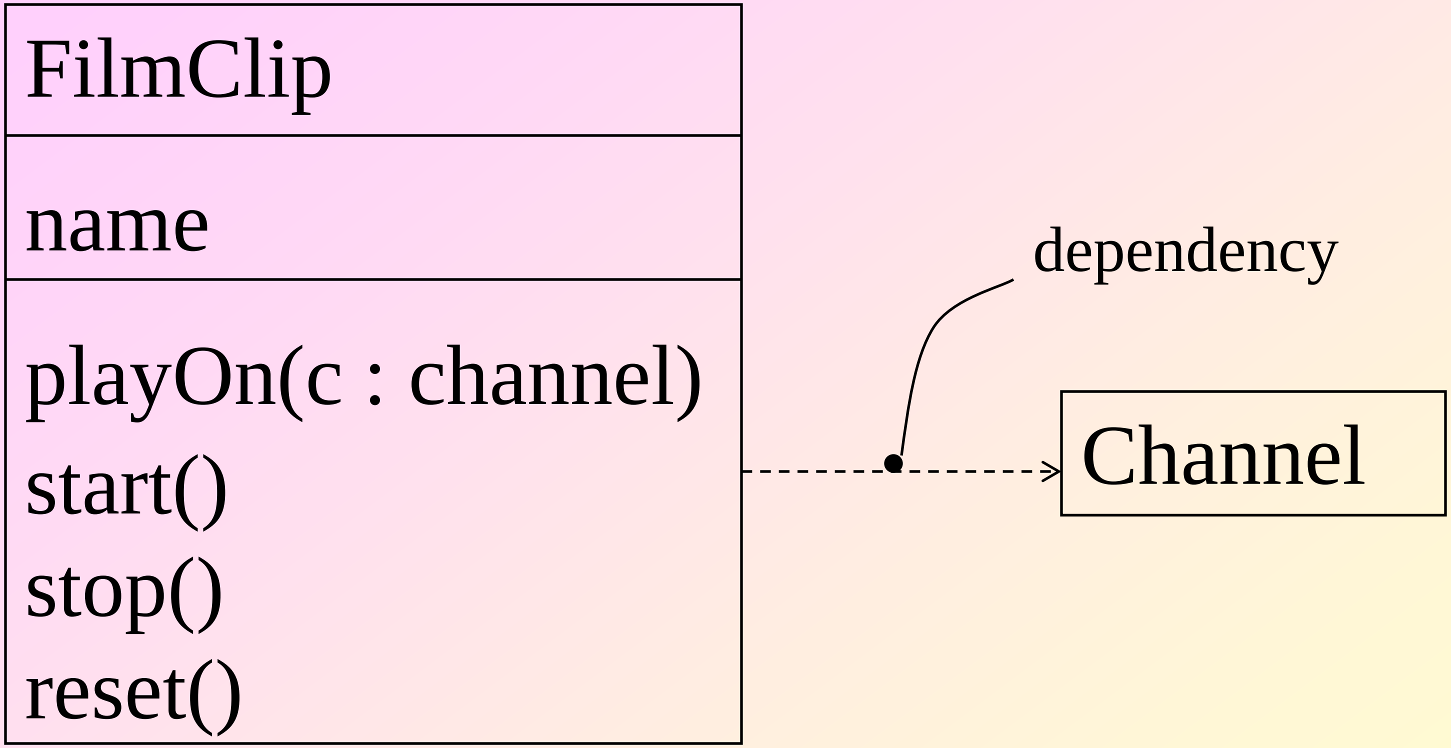
Stereotypes for Class

- The UML defines four standard sterotypes that apply to classes
  - Metaclass : specifies a classifier whose objects are all classes
  - Powertype : specifies a classifier whose objects are the children of a given parent
  - Stereotype : classifier is a stereotype that may be applied to other elements
  - Utility : specifies a class whose attributes and operations are all class scoped

Relationships

- A relationship is a connection among things
- A relationship is rendered as a path, with different kinds lines
- Dependency
  - It is a using relationship that states that a change in specification of one thing may affect another thing that uses it; but not necessarily the reverse
  - It is rendered as a dashed directed line

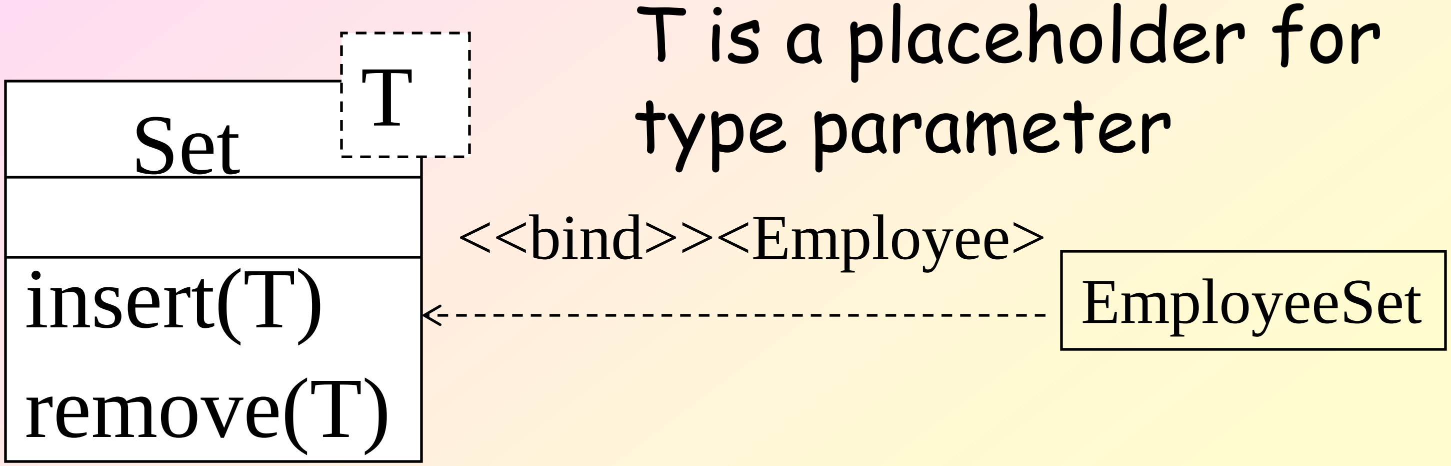
Dependency



- Dependencies will be used in the context of classes to show that one class uses another class as an argument in the signature of an operation

Dependency

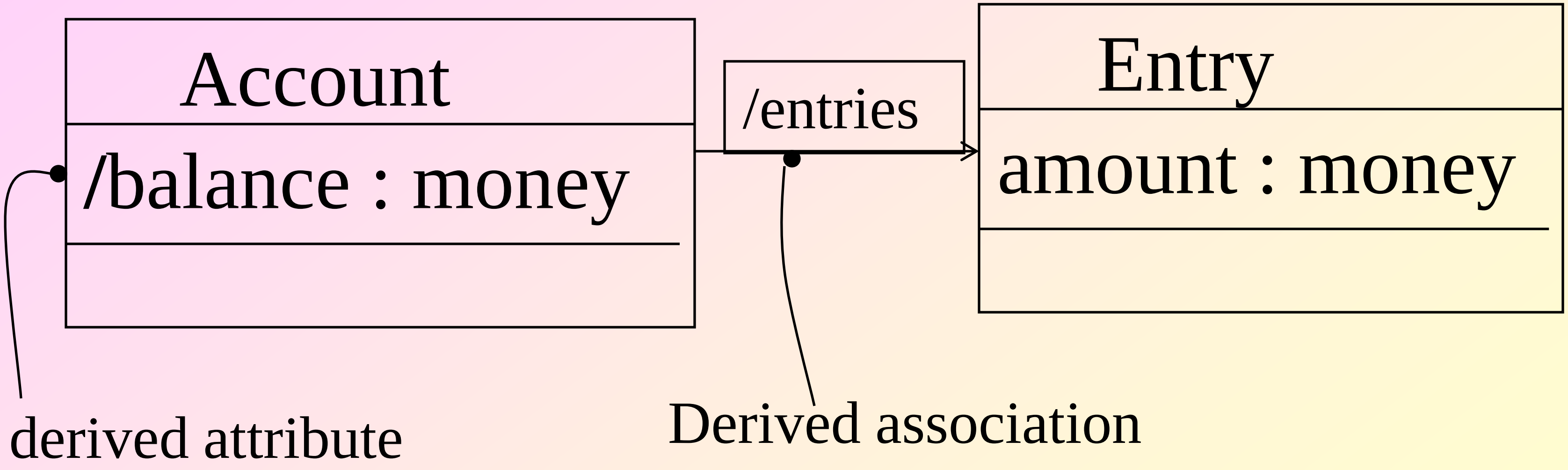
- There are 8 stereotypes that apply to dependency relationship
  - 1. Bind : specifies that the source instantiates the target template using the given actual parameters



T is a placeholder for type parameter

Dependency

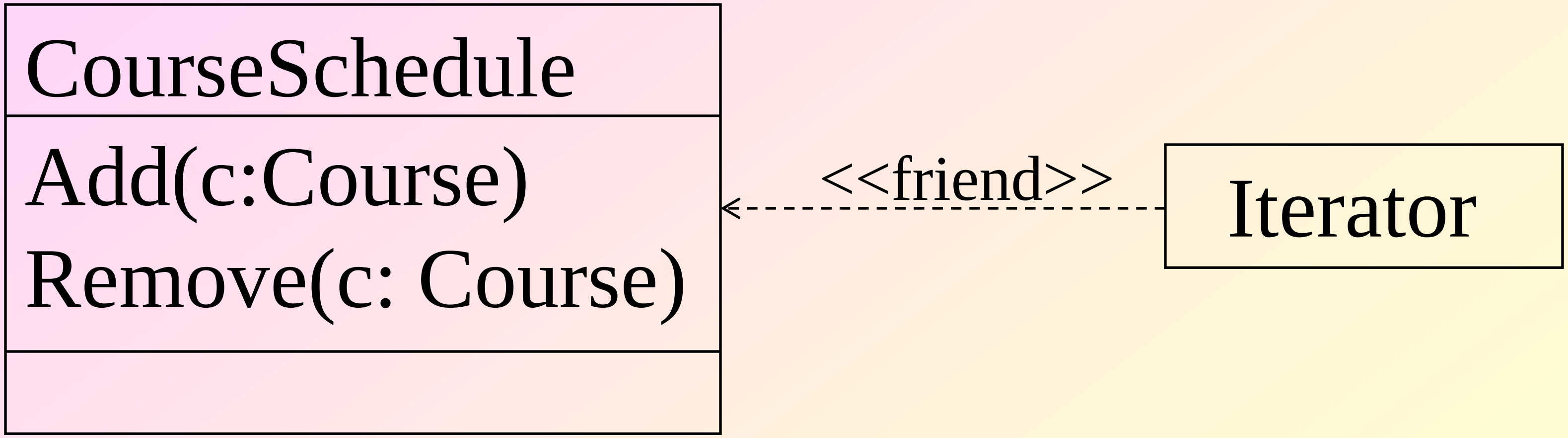
- 2. Derive : specifies that source may be computed from target



{balance = sum of amount of entries}

Dependency

- 3. friend : specifies that source is given special visibility into the target

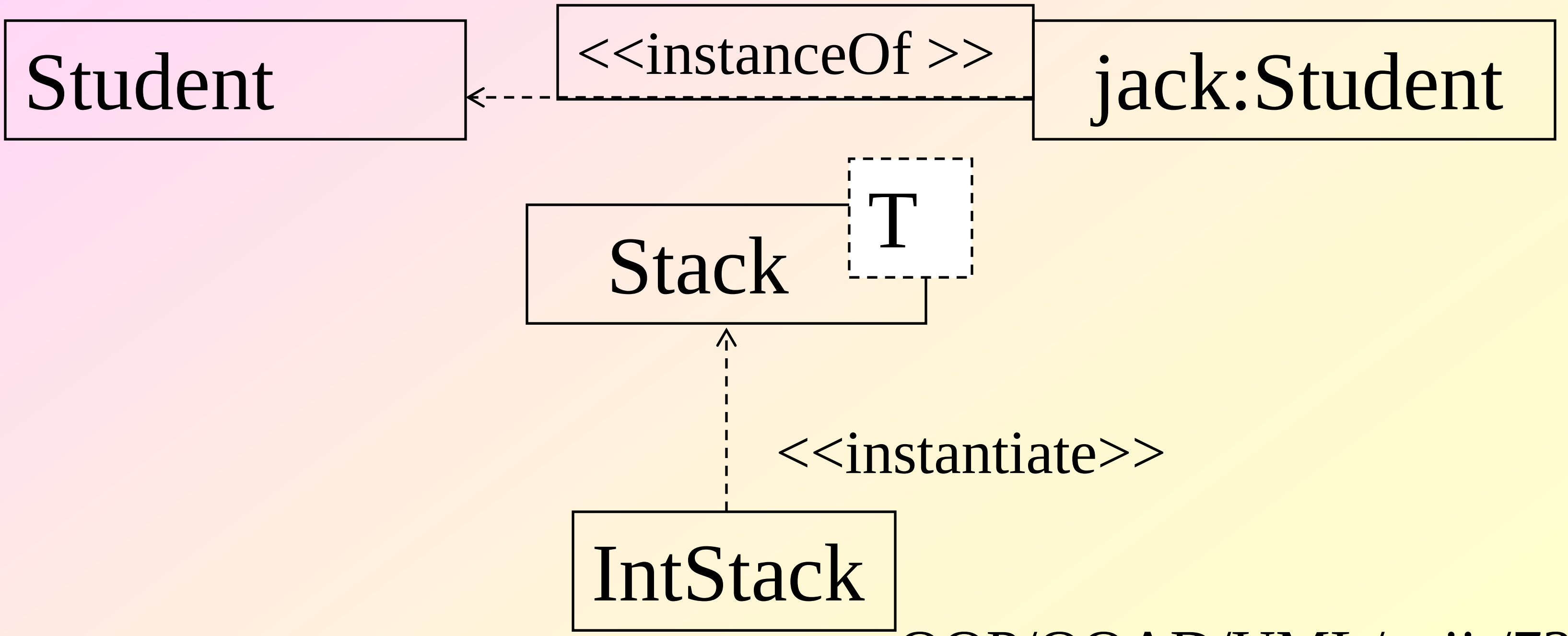


Iterator can visualize CourseSchedule

But the reverse may not be true

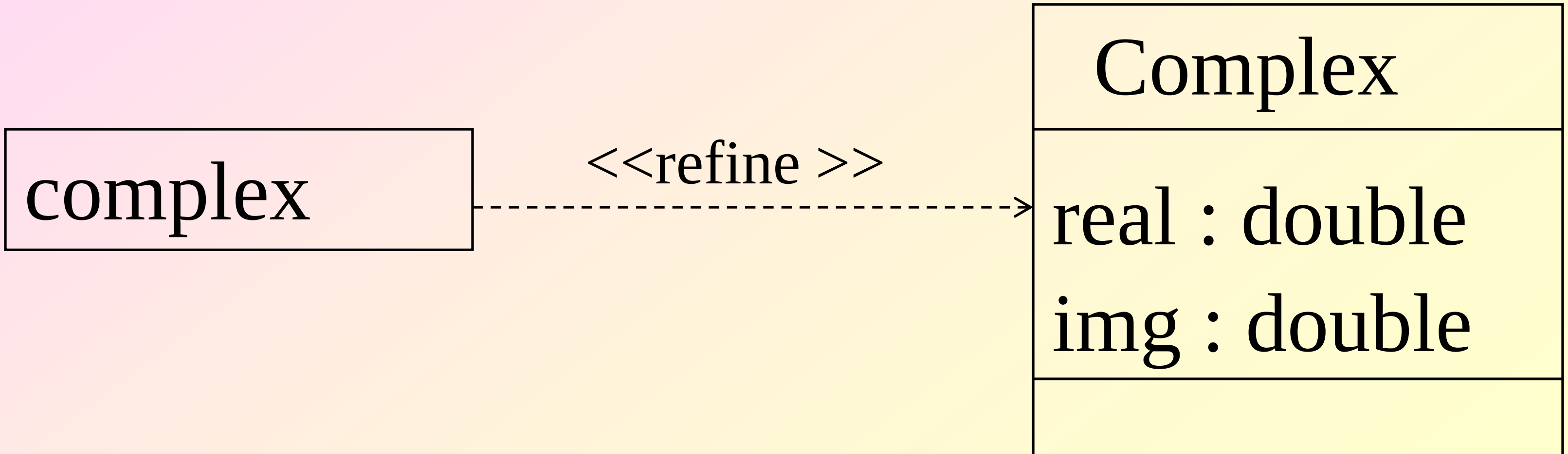
Dependency

- 4. instanceOf : specifies that source object is an instance of target classifier
- 5. Instantiate : specifies that the source creates instances of the target



Dependency

- 6. powertype : specifies that target is a powertype of the source;
  - A powertype is a classifier whose objects are all the children of a given parent
- 7. Refine : specifies that source at a finer degree of abstraction than the target
- 8. Use : specifies that the semantics of the source element depends on the public part of the target



complex

<<refine >>

Complex

real : double

img : double

Dependency

- There are two stereotypes that apply to dependency relationship among packages
  - 1. Access : specifies that the source package is granted the right to reference the elements of the target package
  - 2. Import : it is a kind of access that specifies that the public contents of the target package enter the flat namespace of the source, as if they had been declared in the source

Dependency

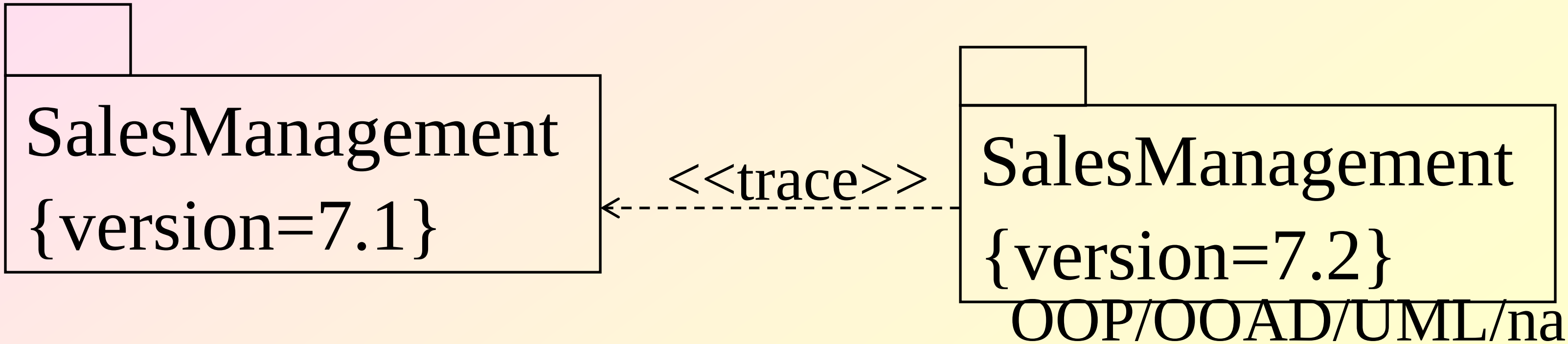
- There are two stereotypes that apply to dependency relationship among use cases
  - 1. external : specifies that the target use case extends the behaviour of the source
  - 2. include : specifies that the source use case explicitly incorporates the behaviour of another use case of a location specified by the source

Dependency

- There are three stereotypes that apply to objects
  - 1. become : specifies that the target is the same object as the source but at a later point in time and with possibly different values, sate or roles
  - 2. call : specifies that the source operation invokes the target operation
  - 3. Copy : specifies that the target object is an exact, but independent copy of the source

Dependency

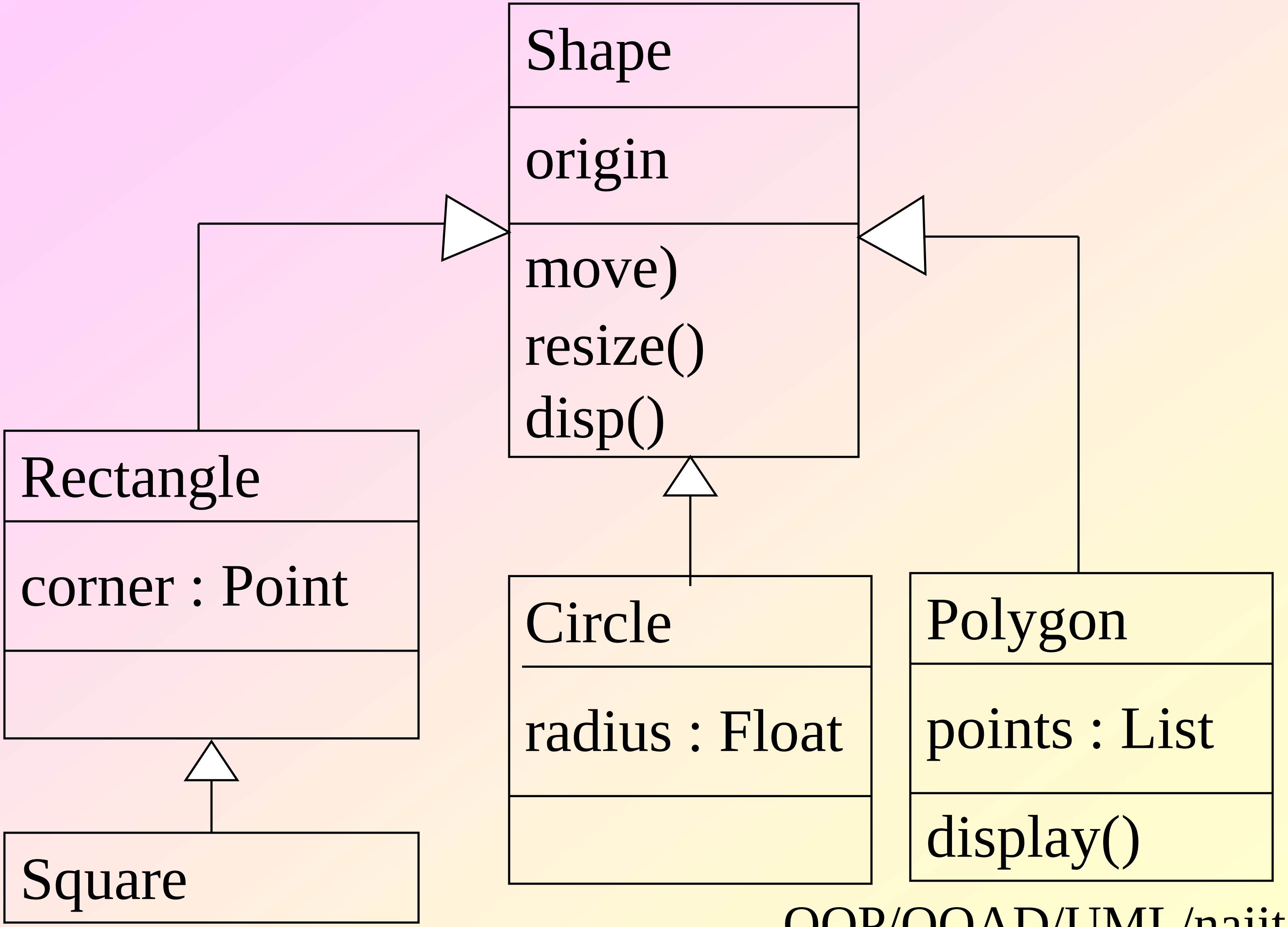
- There is one stereotype that apply to state machines
  - Send : specifies that the source operation sends the target event
- There is one stereotype that apply to subsystems
  - trace : specifies that the target is an historical ancestor of the source



Generalization

- It is a relationship between a general thing (super type) and a more specific kind of that thing (sub type)
- Objects of the child is substitutable for the parent, but not the reverse
- An operation of a child that has the same signature as an operation in a parent overrides the operation of the parent
- It is rendered as a solid directed line with a large open arrow head

Generalization

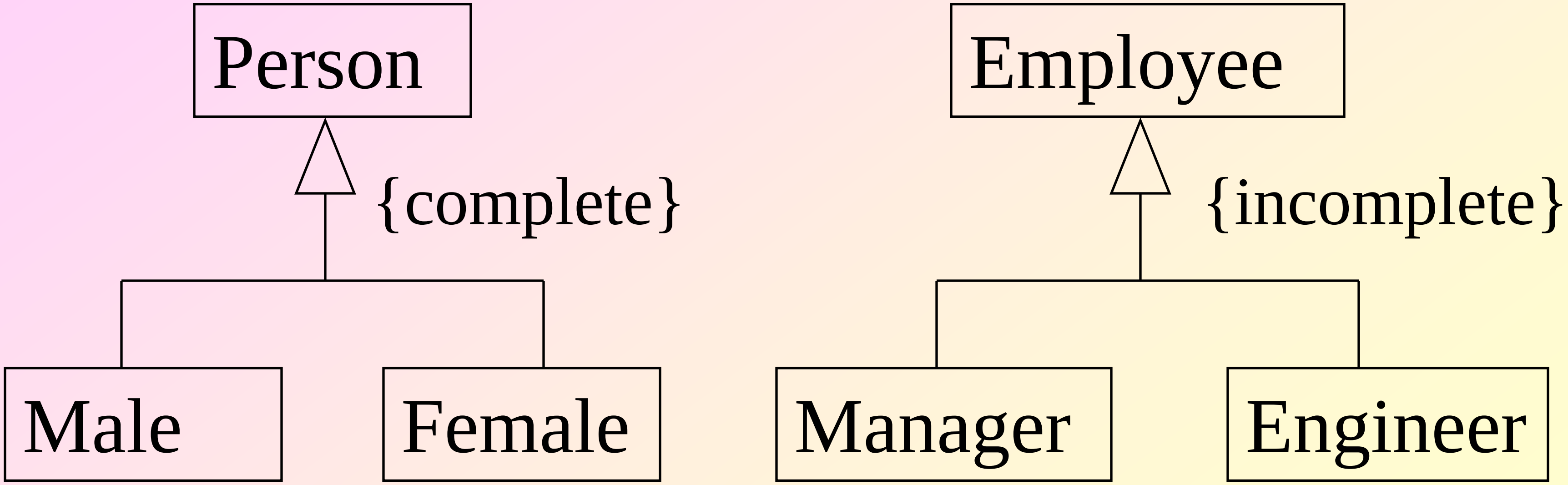


Generalization

- There is one stereotype that apply to generalization
  - implementation : specifies that the child inherits the implementation of the parent, but does not make public nor support its interface, thereby violating substitutability
  - It maps to private inheritance in C++
- Four standard constraints that apply to generalization
  - 1. complete : specifies that all children in the generalization have been specified and no additional children are permitted

Generalization

- incomplete : specifies that not all children in the generalization have been specified and additional children are permitted



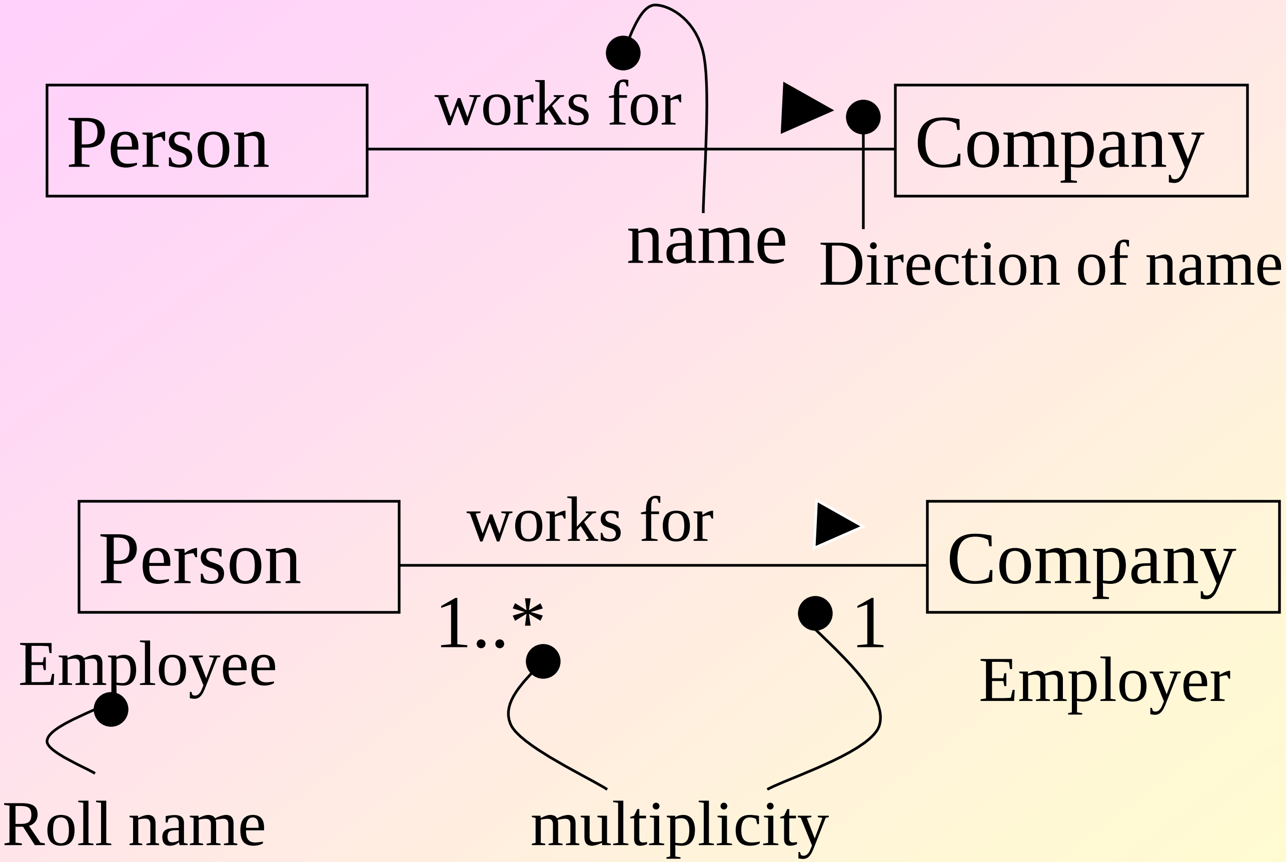
Generalization

- Rest two applies for multiple inheritance
  - 3. disjoint : specifies that objects of parent may have no more than one of the children as a type
  - 4. Overlapping : specifies that objects of parent may have more than one of the children as a type
  - More than one subtype can be a type for an instance

Association

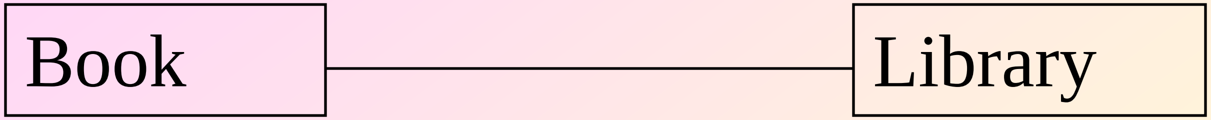
- It is a structural relationship that specifies that objects of one thing are connected to objects of another thing
- An association has name, role and multiplicity
  - Name : used to describe the relationship
  - Role : Each class plays a role in an association
  - Multiplicity : it tells how many objects may be connected across an instance of an association

Association



Association

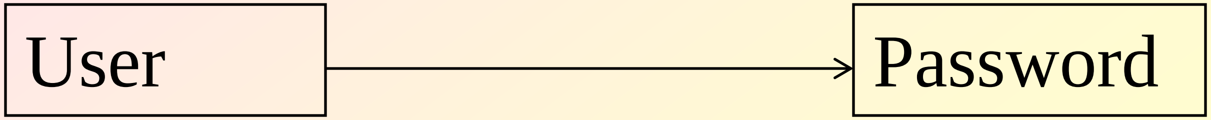
- Properties of Association
  - Navigation : Given an association between two classes, it is possible to navigate from object of one kind to object of other



Book

Library

  - i.e. Given a book you can find out the library and vice-versa
  - If not specified, then an association is bi-directional

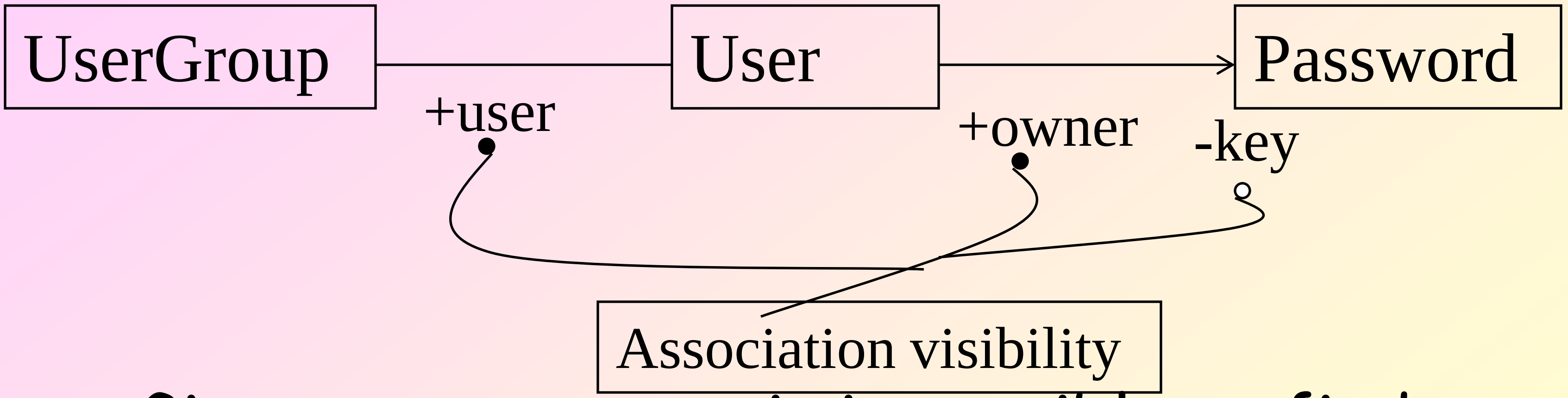


User

Password

Association

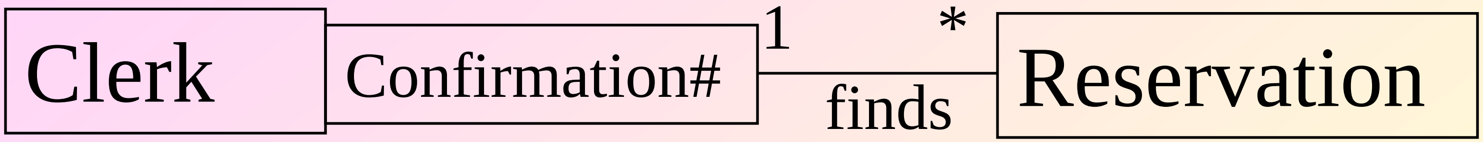
- Visibility : Given an association between two classes, objects of one class can see and navigate to objects of the other.



- Given a user group, it is possible to find out a user but not the password
- This visibility can be limited across association by giving a visibility symbol to a role name

Association

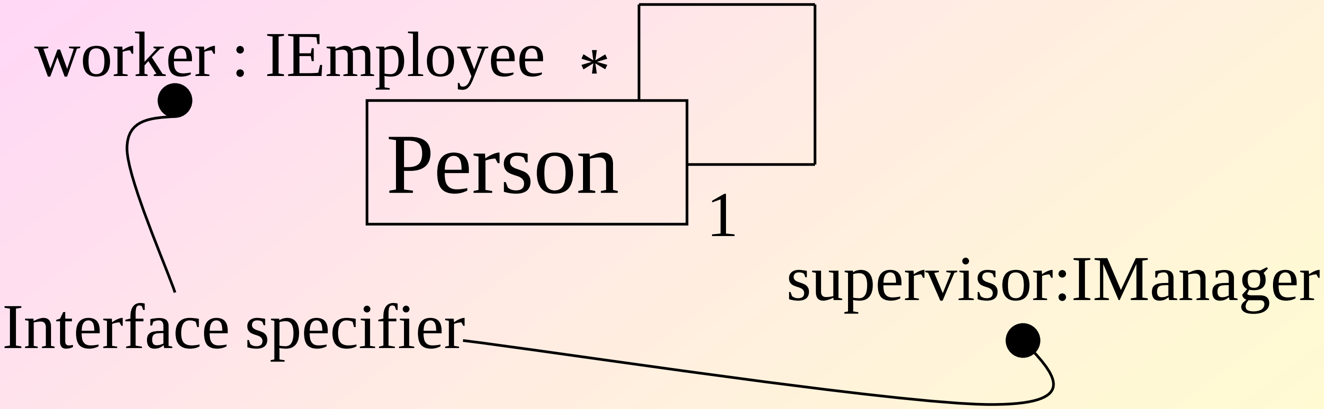
- Qualification : It is required to choose the object at the other end in case of a one-many relation



- I.e. the first class has to rely on a specific attribute to find the target object
- The ID information is called the qualifier

Association

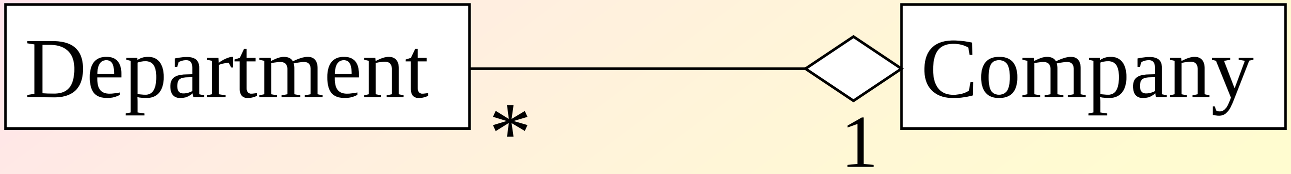
- Interface specifier : In the context of association, as source class may choose to present only one part of its face to the world



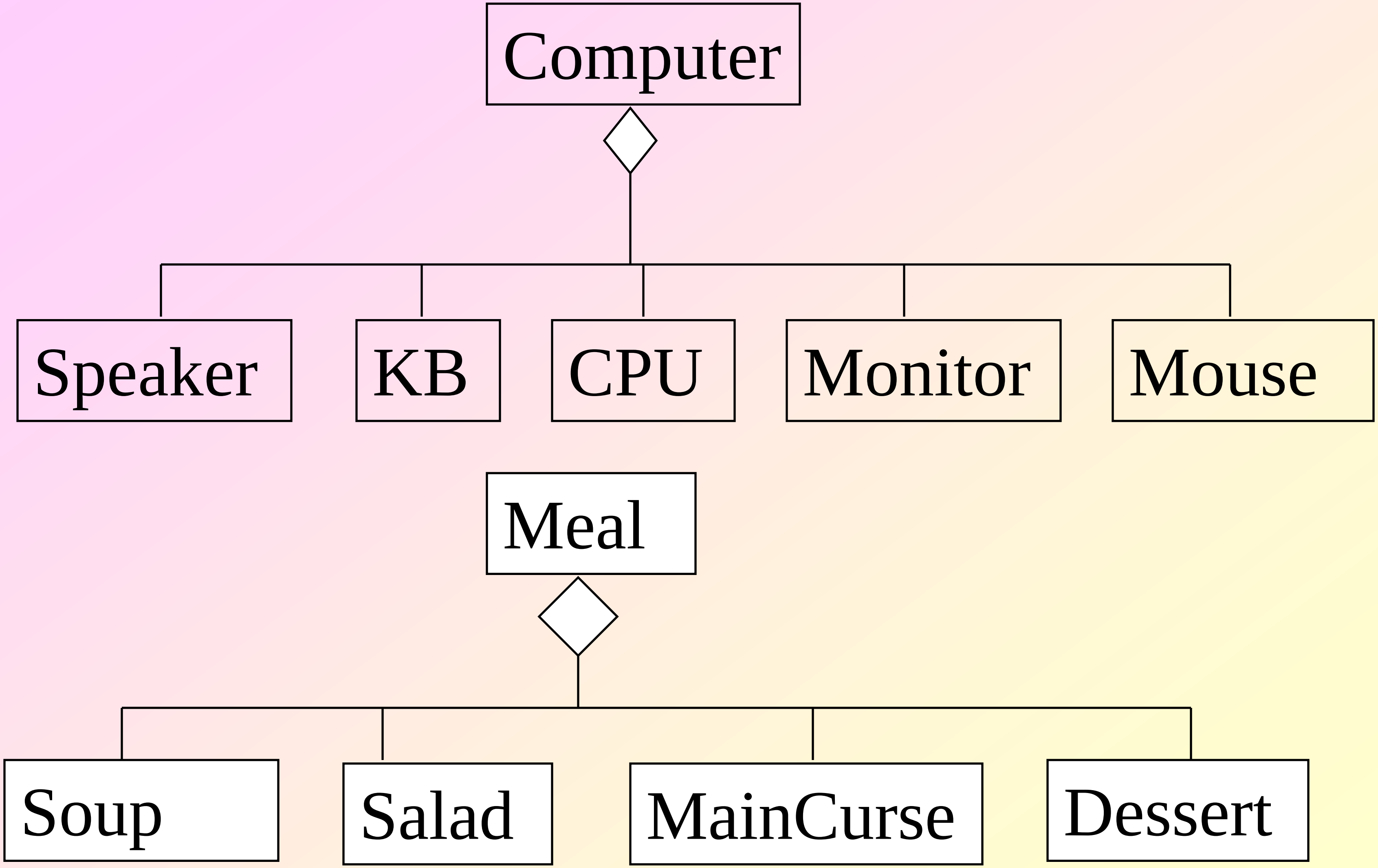
- A person may realize many interfaces like Imanager, Iemployee, and so on …
- Person in the role of supervisor presents only Imanager face to the worker.

Aggregation

- It is a whole/part relationship
- one class represents a larger thing and it consists of smaller things (has a)
- It is a special kind of association and is specified by plane association with an open diamond at the whole end

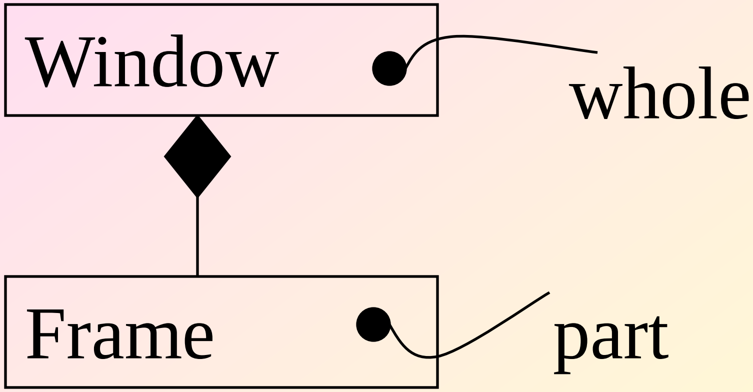


Aggregation



Composition

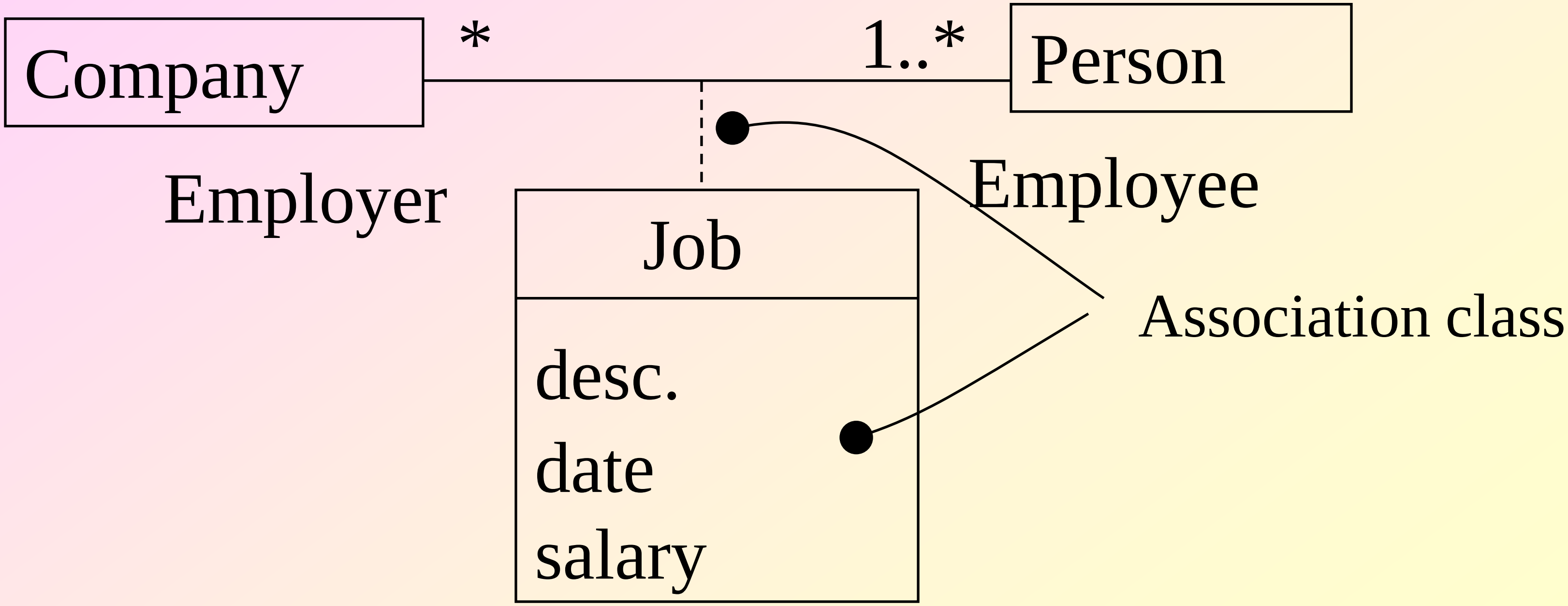
- Composition is a special form of aggregation
- Composite must manage the creation and destruction of its parts



- When an window object is destroyed, the frame objects is also destroyed

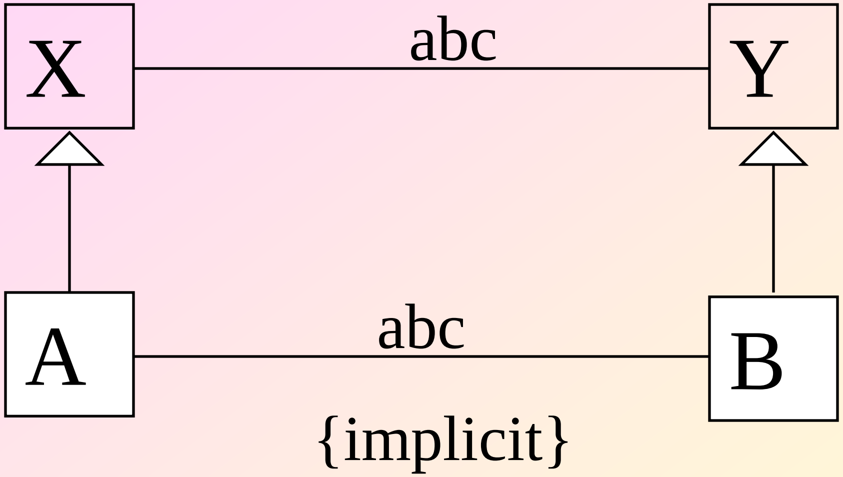
Association classes

- In an association between two classes, the association itself might have some properties



Association

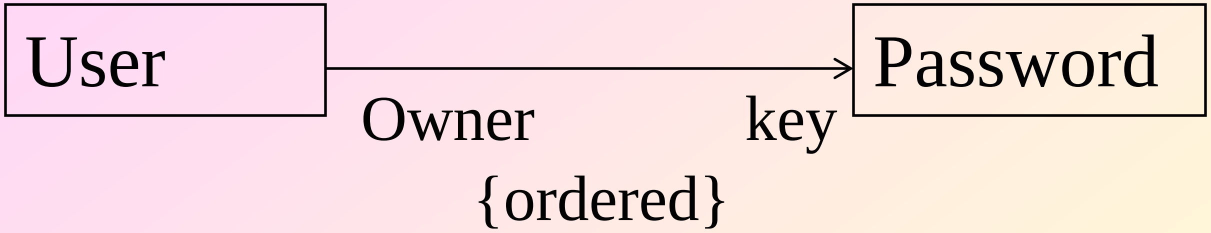
- There are five constraints that apply to association
  - Implicit : specifies that the relationship is not manifest but, rather, is only conceptual



- Association between two base classes specify that same association between children of these classes

Association constraints

- Ordered : specifies that the set of objects at one end of an association are in an explicit order



- Passwords associated with the user might be kept in a least-recently user order
- Changeable : links between objects may be added, removed and changed freely

Association constraints

- Addonly : new links may be added from an object
- Frozen : A link, once added from an object, may not be modified or deleted

Thank You