UNIT-I

UML class diagram

MADHESWARI.K
AP/CSE
SSNCE

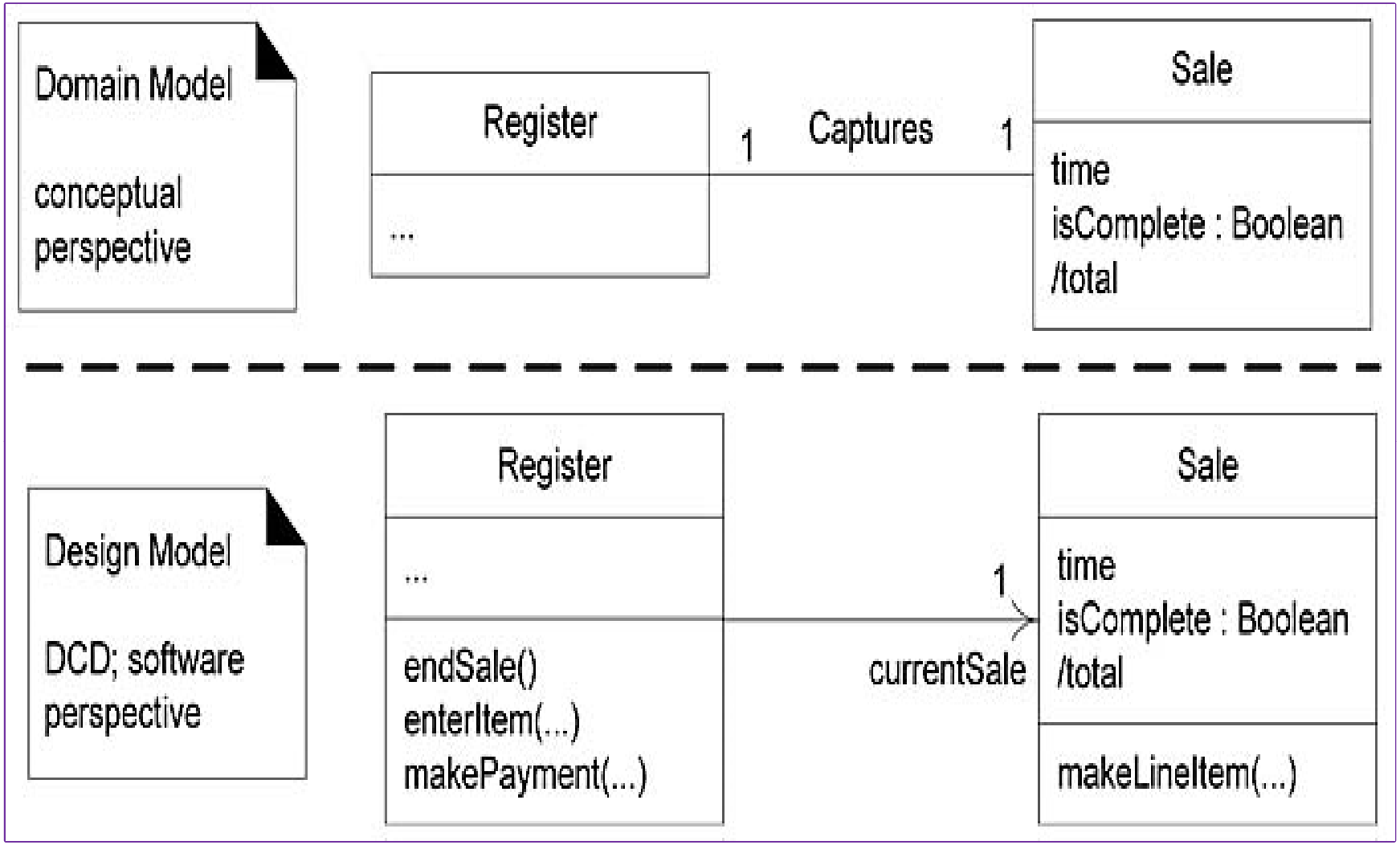
UML CLASS DIAGRAM

- Class diagrams illustrate classes, interfaces, and their associations.
- They are used for static object modeling.

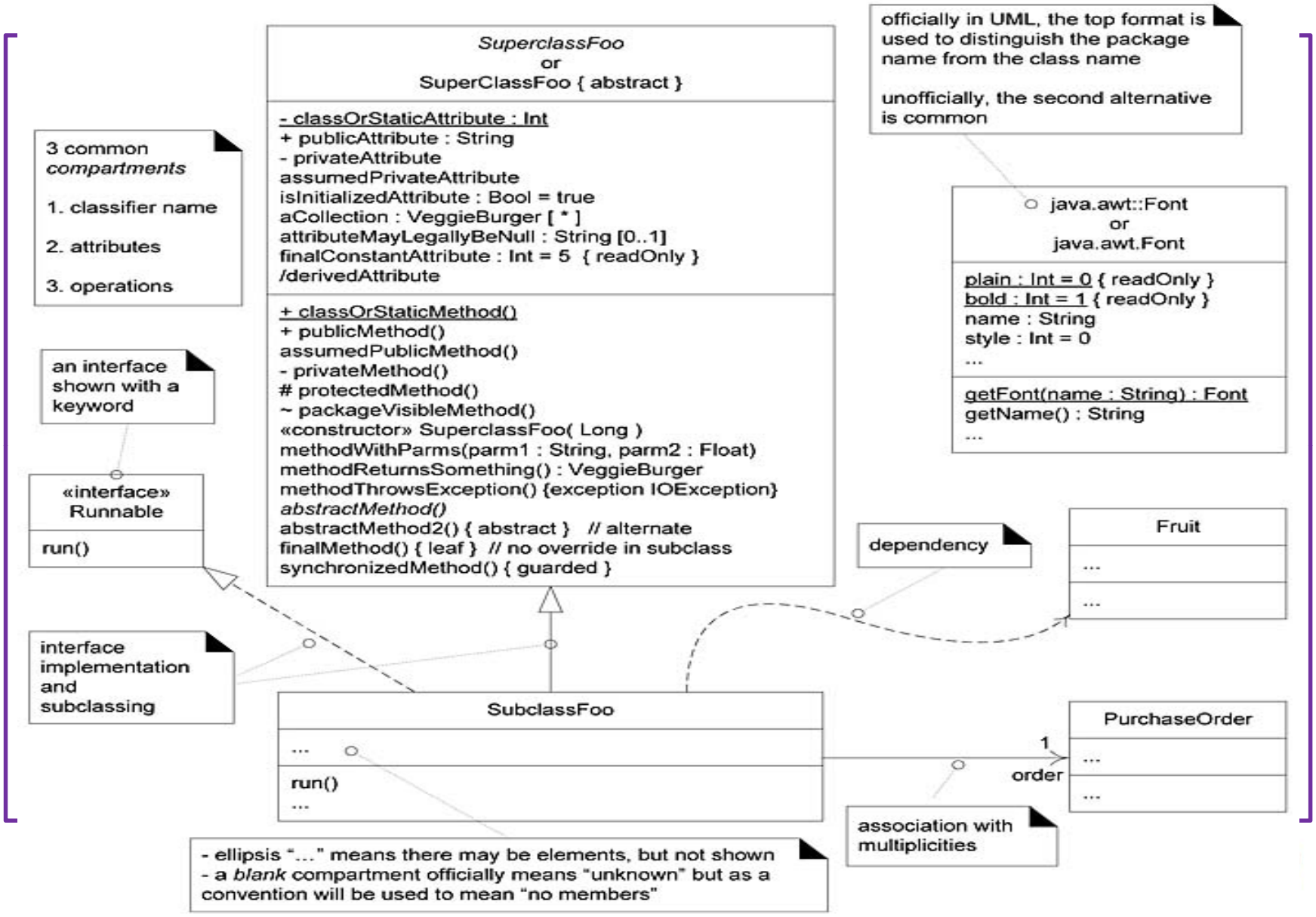
Design class diagram(DCD)

- Same UML diagram can be used in multiple perspectives
- In a conceptual perspective the class diagram can be used to visualize a domain model.
- A unique term to clarify when the class diagram is used in a software or design perspective is called **design class diagram** (**DCD**), and all DCDs form part of the Design Model.
- Other parts of the Design Model include UML interaction and package diagrams.

UML class Diagrams in two perspectives



UML CLASS DIAGRAM NOTATION



UML classifier

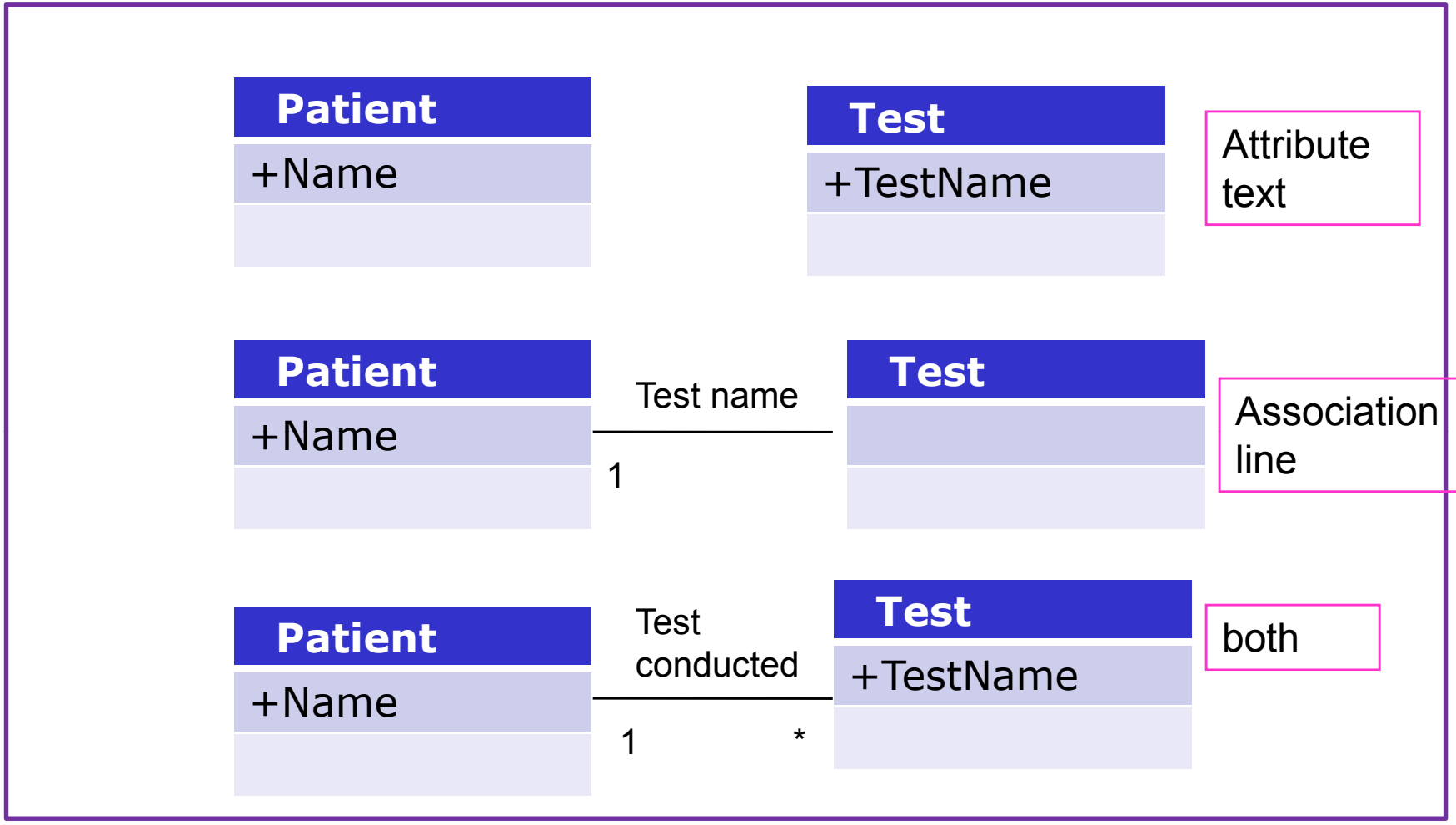
- A UML **classifier** is "a model element that describes behavioral and structure features" Classifiers can also be specialized.
- They are a generalization of many of the elements of the UML, including classes, interfaces, use cases, and actors.
- In class diagrams, the two most common classifiers are
  - regular classes and
  - interfaces.

Attributes and Association lines

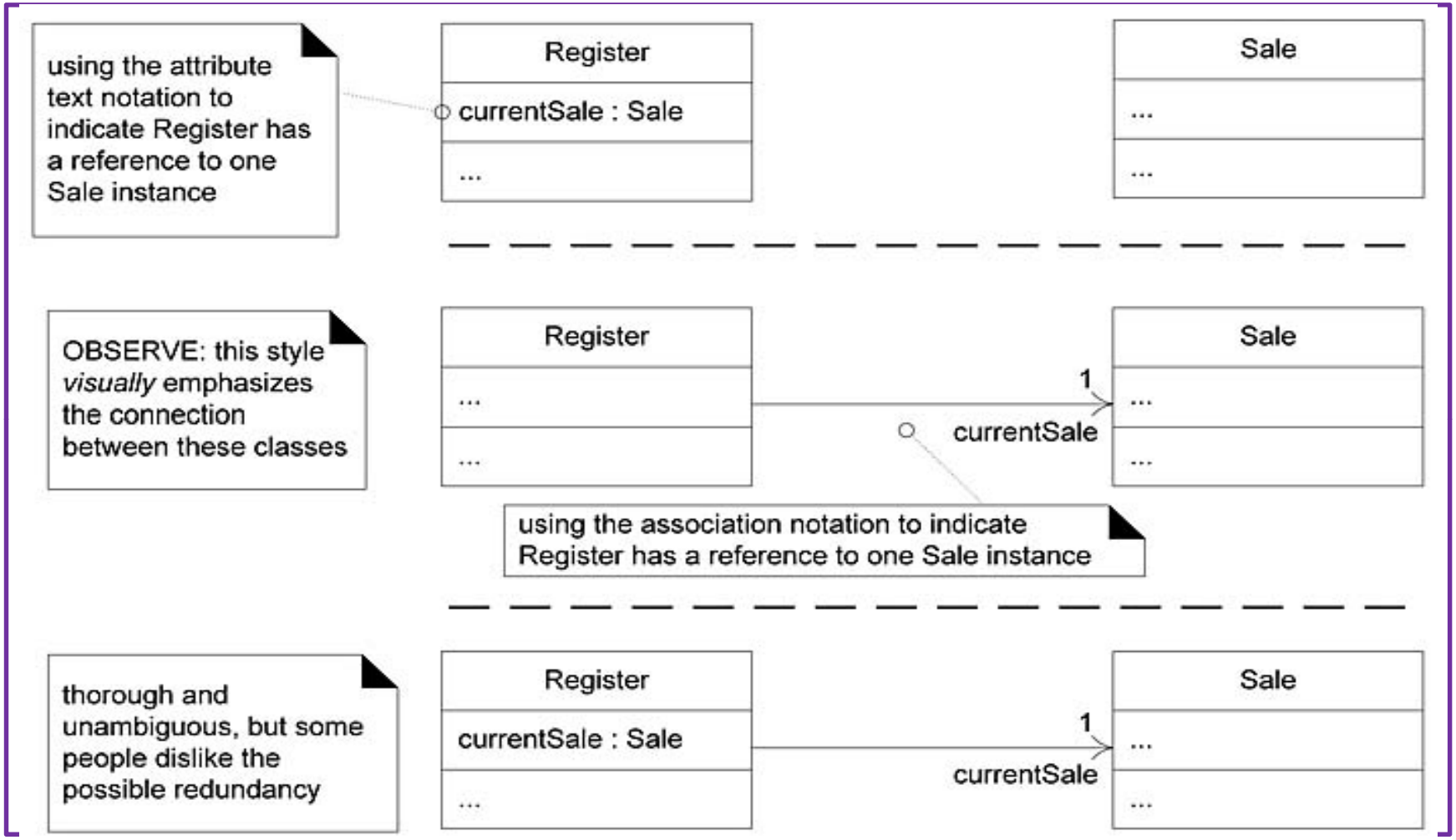
ways to show UML attributes

- Attributes of a classifier (also called **structural properties** in the UML[1]) are shown several ways:
- **attribute text** notation, such as *currentSale : Sale.*
- **association line** notation
- **both** together

ways to show UML attributes



ways to show UML attributes



UML attributes- format

- Format of the attribute text notation

Visibility name: type multiplicity=default{property-string}

visibility mark includes

+(public)

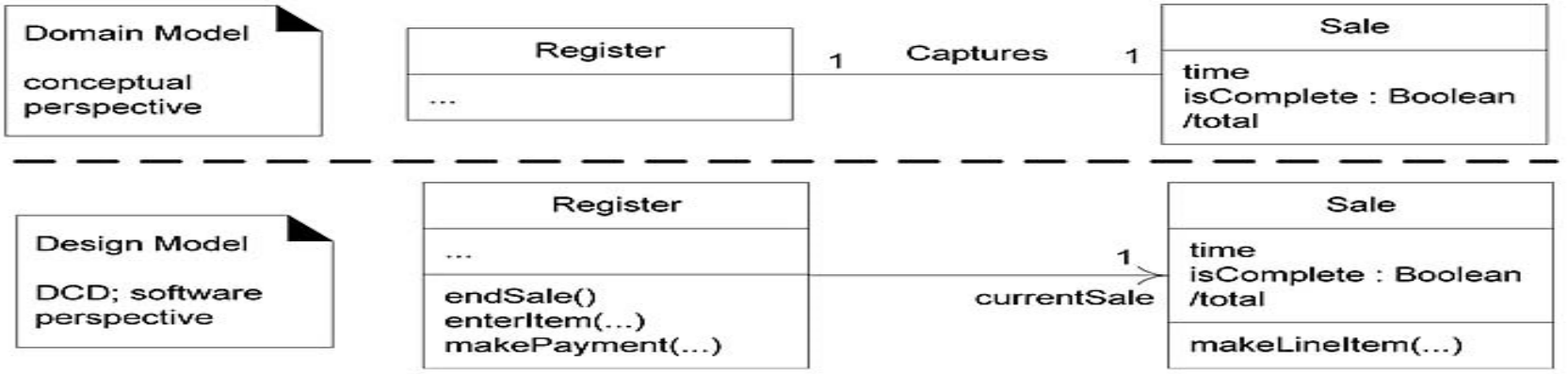
-(private)

(#) protected

Note: attributes are usually assumed private if no visibility is given

Attribute as association line has the following style

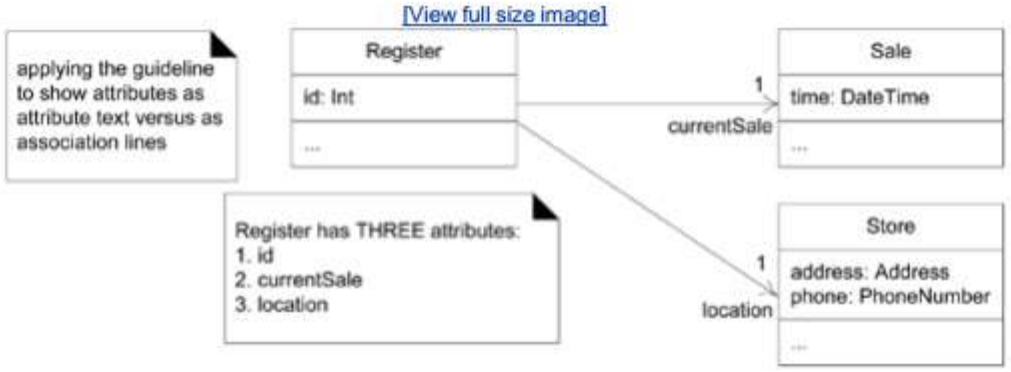
- A Navigability arrow pointing from the source(register) to target (Sale) object, indicating a Register object has an attribute of one sale
- A rolename(currentsale) only at the target end to show the attribute name
- Multiplicity can be 1 …1, 1..* and so on. It is denoted at the end of the association line
- When using class diagrams for domain model do show association names but avoid navigation arrows, as a domain model is not a



when to use attribute text and association lines

- When an object can be described by certain properties which are associated with primitive data types then such properties are denoted by attributes of that object
- When there is a visual relationship between two objects that has visual emphasis then those objects are connected by the association line

Figure 16.5. Applying the guidelines to show attributes in two notations.



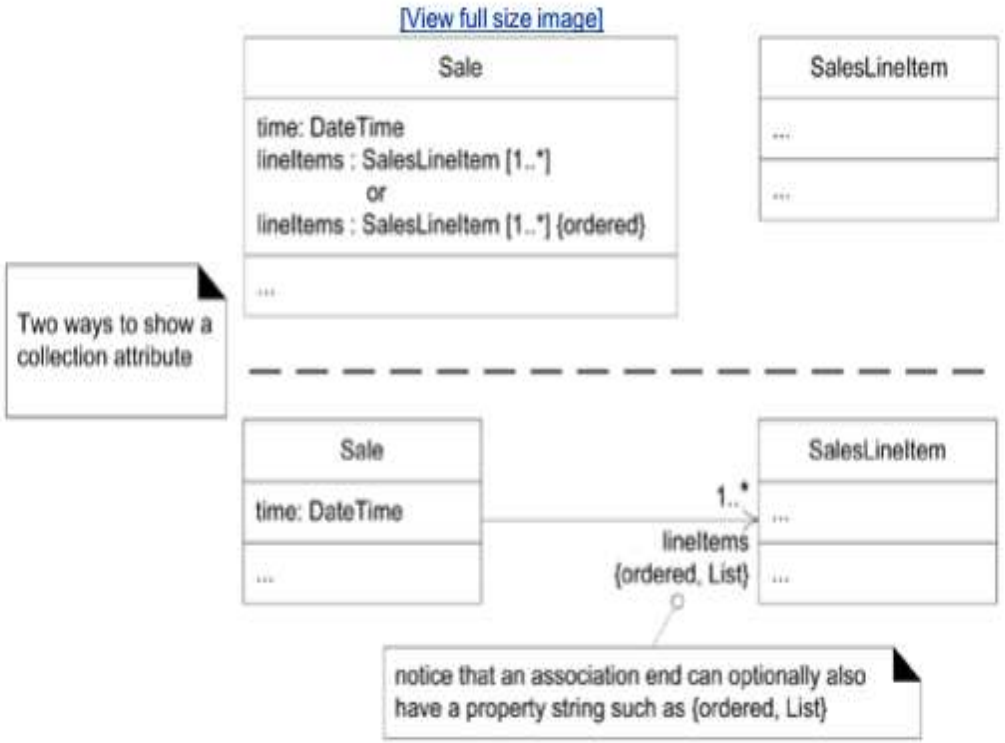

```
public class Register
{
private int id;
private Sale currentSale;
private Store location;
// …
}
```

Property string

- **property string** such as *{ordered}* or *{ordered, List}* is possible.
- *{ordered}* is a UML-defined **keyword** that implies the elements of the collection are (the suspense builds…) ordered.
- Another related keyword is *{unique}*, implying a *set* of unique elements.
- The keyword *{List}* illustrates that the UML also supports user-defined keywords.

Property string

Figure 16.6. Two ways to show a collection attribute in the UML.




```
public class Sale
{
private List<SalesLineItem> lineItems =
                            new ArrayList<SalesLineItem>();
```

Note Symbols: Notes, Comments, Constraints, and Method Bodies

Notes

- The use of note symbol is very common in UML, but this symbol can be specially used in class diagram to represent the note or comment, constraint and method.
- Note or comment the **extra information** about the element used in the UML diagram can be provided by the note.
- A UML **note symbol** is displayed as a dog-eared rectangle with a dashed line to the annotated element;

- a UML **note** or **comment**, which by definition have no semantic impact
- a UML **constraint**, in which case it must be encased in braces '{...}'
- a **method** body—the implementation of a UML operation

Operations and Methods

Operations

- One of the compartments of the UML class box shows the signatures of operations
- format of the operation syntax is:

  visibility name (parameter-list) {property-string}

  visibility name (parameter-list) : return-type {property-string}
- ***Guideline***: Operations are usually assumed public if no visibility is shown.
- The property string contains arbitrary additional information, such as exceptions that may be raised, if the operation is abstract, and so forth.
- UML allows the operation signature to be written in any programming language, such as Java

```
+ getPlayer( name : String ) : Player {exception IOException} public
  Player getPlayer( String name ) throws IOException
```

- An operation is *not* a method. A UML **operation** is a *declaration*, with a name, parameters, return type, exceptions list, and possibly a set of *constraints* of pre- and post-conditions. But, it isn't an implementation—rather, methods are implementations

How to Show Methods in Class Diagrams

- A UML **method** is the implementation of an operation; if constraints are defined, the method must satisfy them. A method may be illustrated several ways, including:
  - in interaction diagrams, by the details and sequence of messages
  - in class diagrams, with a UML note symbol stereotyped with «method»

[View full size image]

Keywords

Keywords

- A UML **keyword** is a textual adornment to categorize a model element.
- For example, the keyword to categorize that a classifier box is an interface is «interface».
- The «actor» keyword was used to replace the human stick-figure actor icon with a class box to model computer-system or robotic actors.
- ***Guideline***: When sketching UML—when we want speed, ease, and creative flow—modelers often simplify keywords to something like '<interface>' or '<I>'.

Examples of using Keywords

Keyword	Meaning	Example Usage
«actor»	classifier is an actor	in class diagram, above classifier name
«interface»	classifier is an interface	in class diagram, above classifier name
{abstract}	abstract element; can't be instantiated	in class diagrams, after classifier name or operation name
{ordered}	a set of objects have some imposed order	in class diagrams, at an association end

Stereotypes, Profiles, and Tags

stereotype

- As with keywords, stereotypes are shown with guillemets symbols, such as «authorship»
- A **stereotype** represents a refinement of an existing modeling concept and is defined within a UML **profile**
- **Stereotypes** are used to extend the UML notation elements
- The UML predefines many stereotypes, such as «destroy» (used on sequence diagrams), and also allows user-defined ones. Thus, stereotypes provide an *extension mechanism* in the UML.

Profile

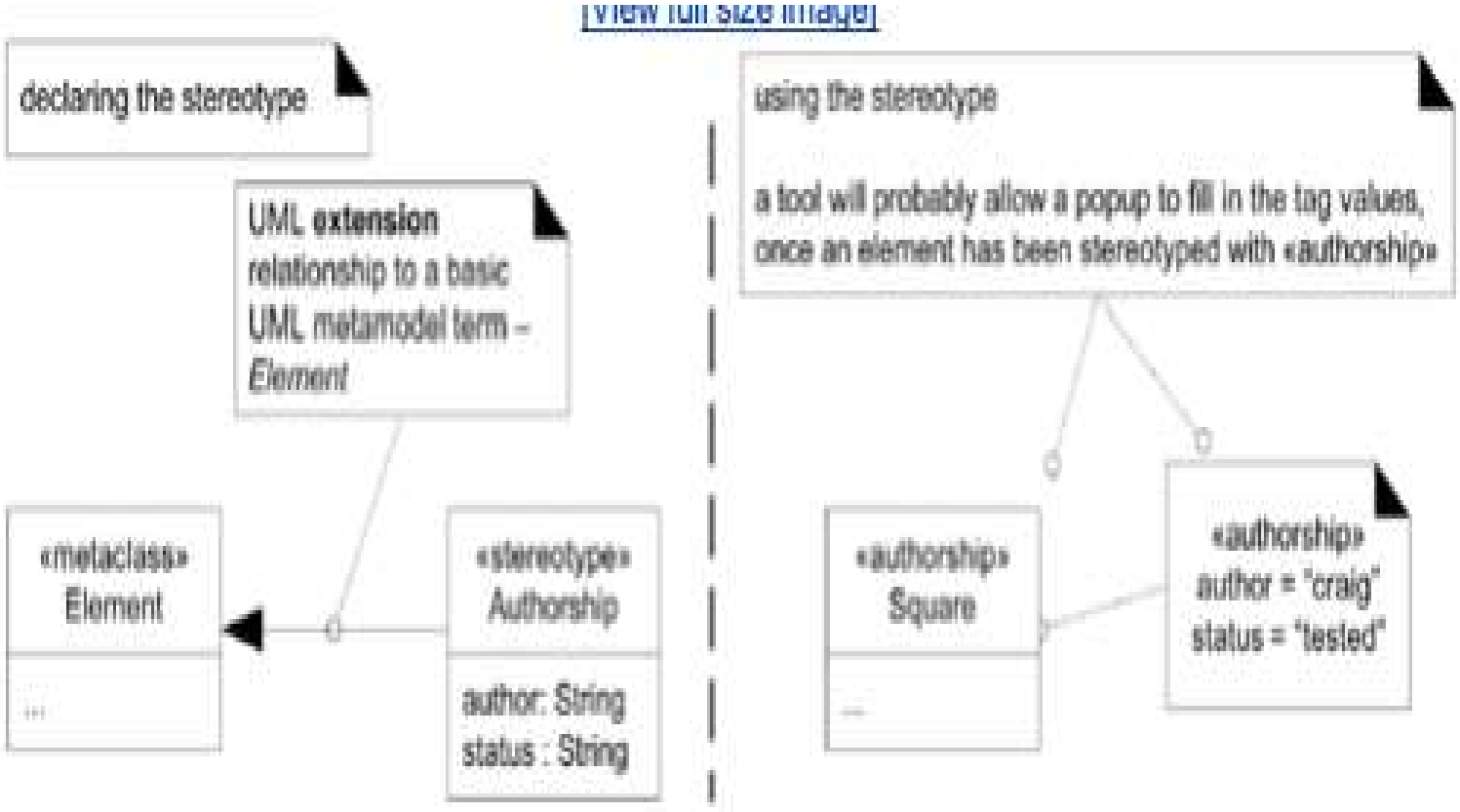
- Profile is a collection of related stereotypes, tags and constraints that are normally used to specialize the specific domain using UML notations.

Tags

- The stereotypes declare the set of tags.
- When stereotypes is used in the model element then values of its properties are referred as tagged values.
- Tagged values are standard meta attribute values

stereotype

- For example, Figure shows a stereotype declaration, and its use. The stereotype declares a set of **tags**, using the attribute syntax. When an element (such as the *Square* class) is marked with a stereotype, all the tags apply to the element, and can be assigned values.



UML Properties and Property Strings

Properties and Property Strings

- In the UML, a **property** is "a named value denoting a characteristic of an element.
- A property has semantic impact."
- Some properties are predefined in the UML, such as *visibility*—a property of an operation. Others can be user-defined.
- property string can be denoted by the name value pair,

Example

Property-string{abstract, visibility=private}

Some properties can also be shown without values

Example

Property-string{abstract}

Generalization, Abstract Classes, Abstract Operations

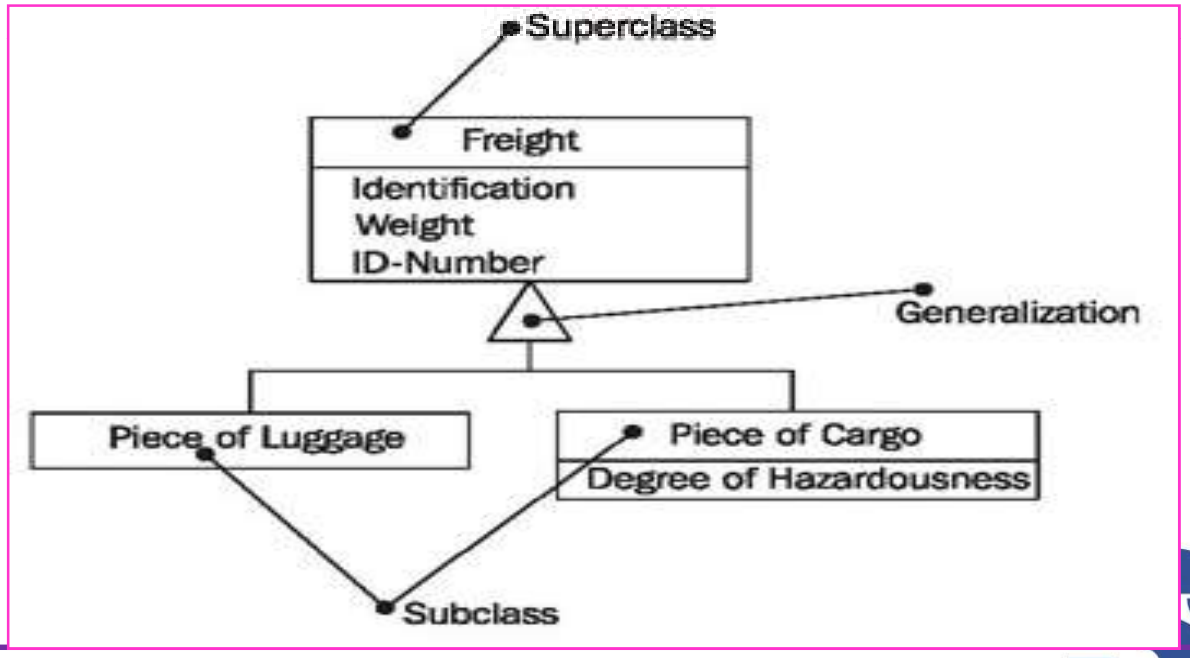
Generalization, Abstract Classes, Abstract Operations

Generalization

- **Generalization** in the UML is shown with a solid line and fat triangular arrow from the subclass to superclass

Generalization—A taxonomic relationship between a more general classifier and a more specific classifier. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier indirectly has features of the more general classifier.

Example



Generalization, Abstract Classes, Abstract Operations

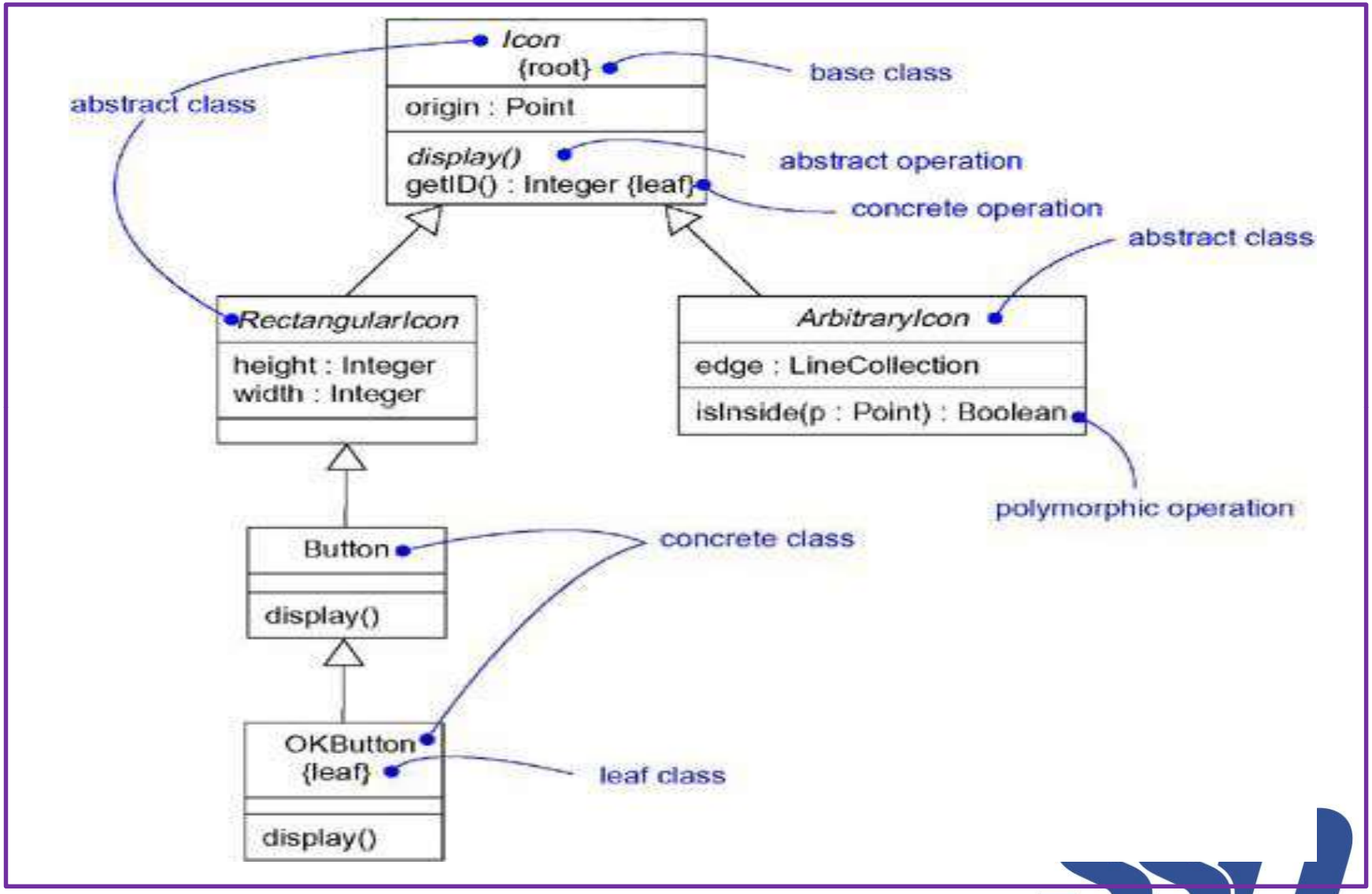
Abstract classes

abstract classes and operations can be shown either with an *{abstract}* tag (useful when sketching UML) or by italicizing the name

Final classes

The opposite case, **final classes** and operations that can't be overridden in subclasses, are shown with the *{leaf}* tag.

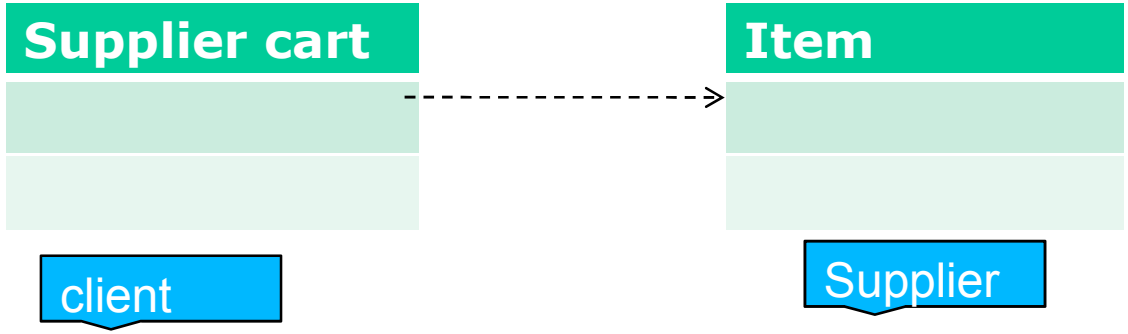
Generalization, Abstract Classes, Abstract Operations



Dependency

Dependency

- Dependency lines may be used on any diagram, but are especially common on class and package diagrams.
- The UML includes a general **dependency relationship** that indicates that a **client** element (of any kind, including classes, packages, use cases, and so on) has knowledge of another **supplier** element and that a change in the supplier could affect the client.
- Dependency is illustrated with a dashed arrow line from the client to supplier.



- Dependency can be viewed as another version of **coupling**, a traditional term in software development when an element is coupled to or depends on another.

.

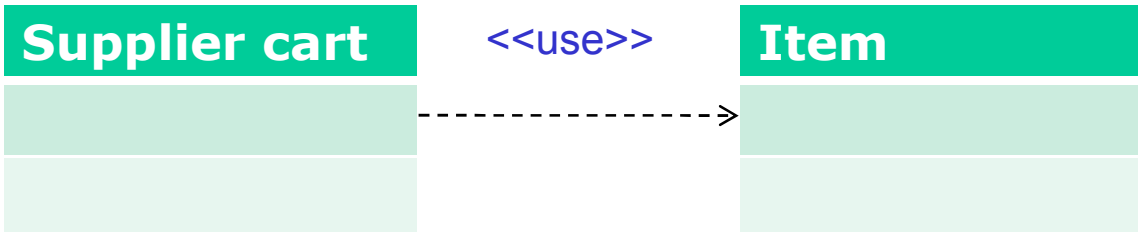
Dependency

The dependency relationship indicates that the client performs one of the following functions

- sends a message to supplier class
- Temporarily uses a supplier class that has global scope
- Temporarily uses a supplier class as a parameter for one of its operations

The dependency can be labeled using the keywords or stereotypes. Various keywords or stereotypes that can be used in dependency relationship are

<<bind>>, <<realize>>, <<substitute>>, <<trance>>, <<derive>>, <<refine>>, <<use>>, <<call>>, <<create>>, <<send>>,

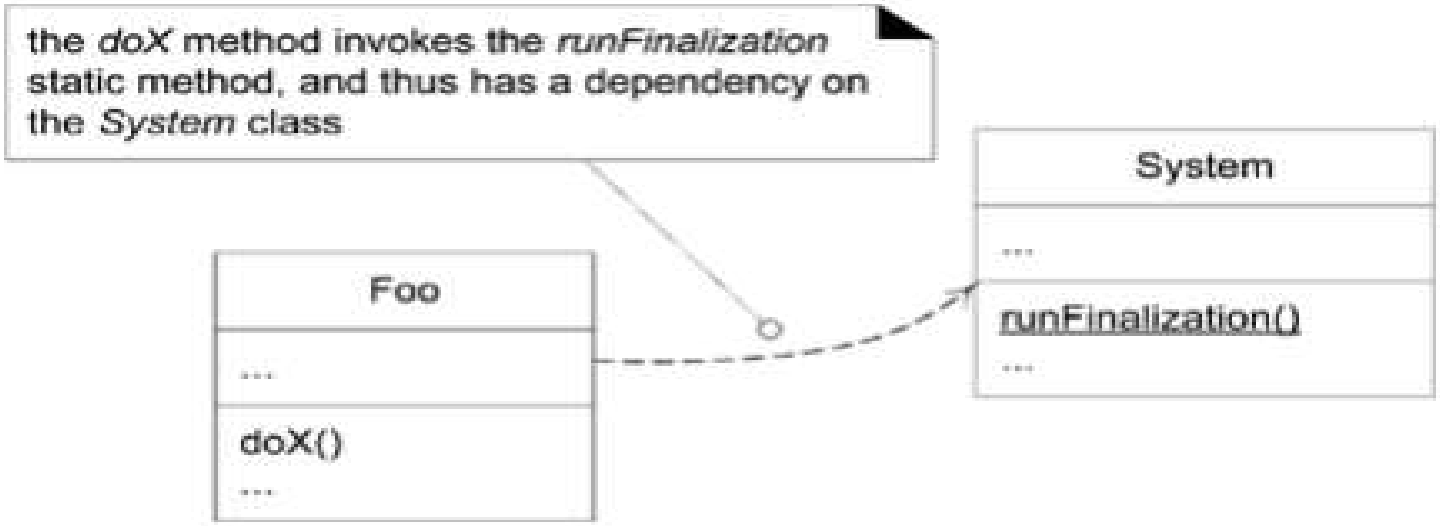


Dependency

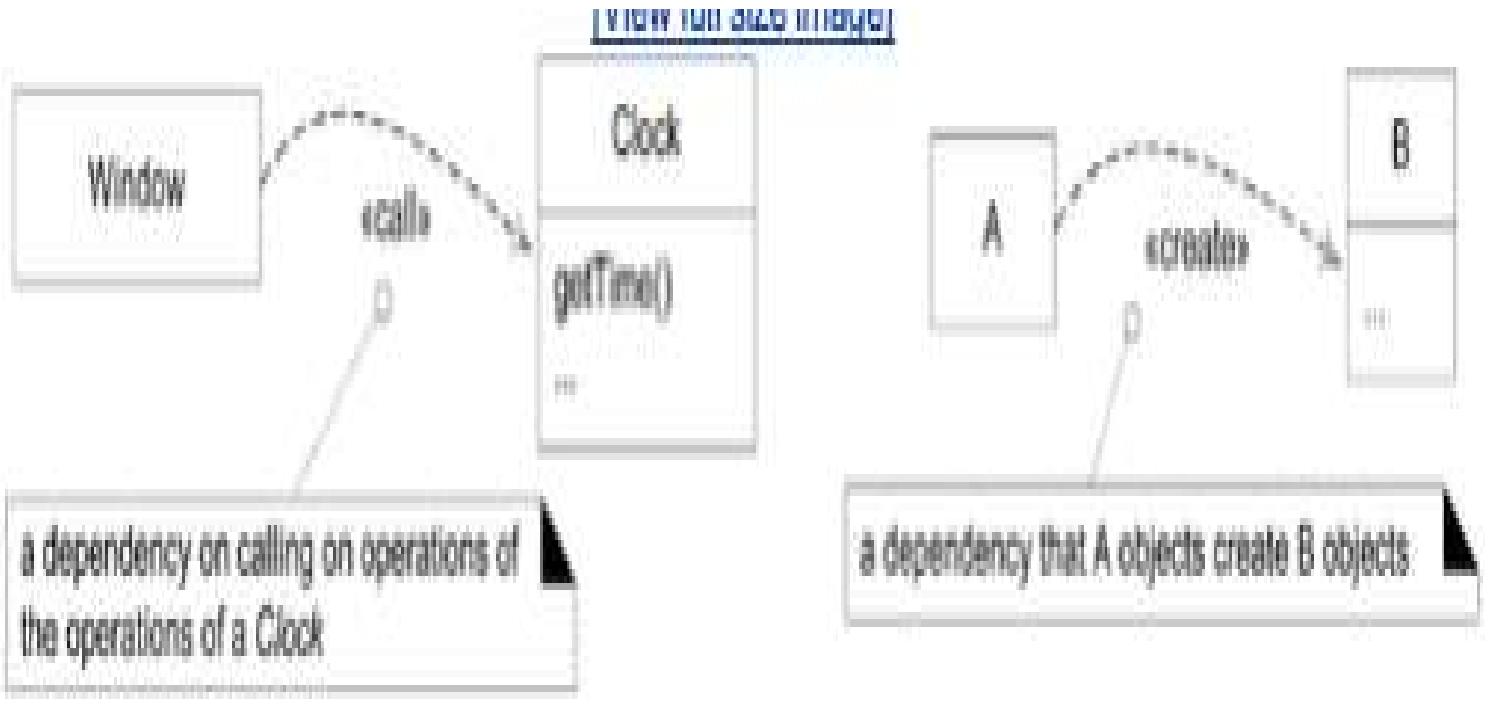
example: The following Java code shows a *doX* method in the *Foo* class:

```
public class Foo { public
void doX() {
System.runFinalization()
; //… } // … }
```

The *doX* method invokes a static method on the *System* class. Therefore, the *Foo* object has a static-method dependency on the *System* class. This dependency can be shown in a class diagram



Dependency Labels

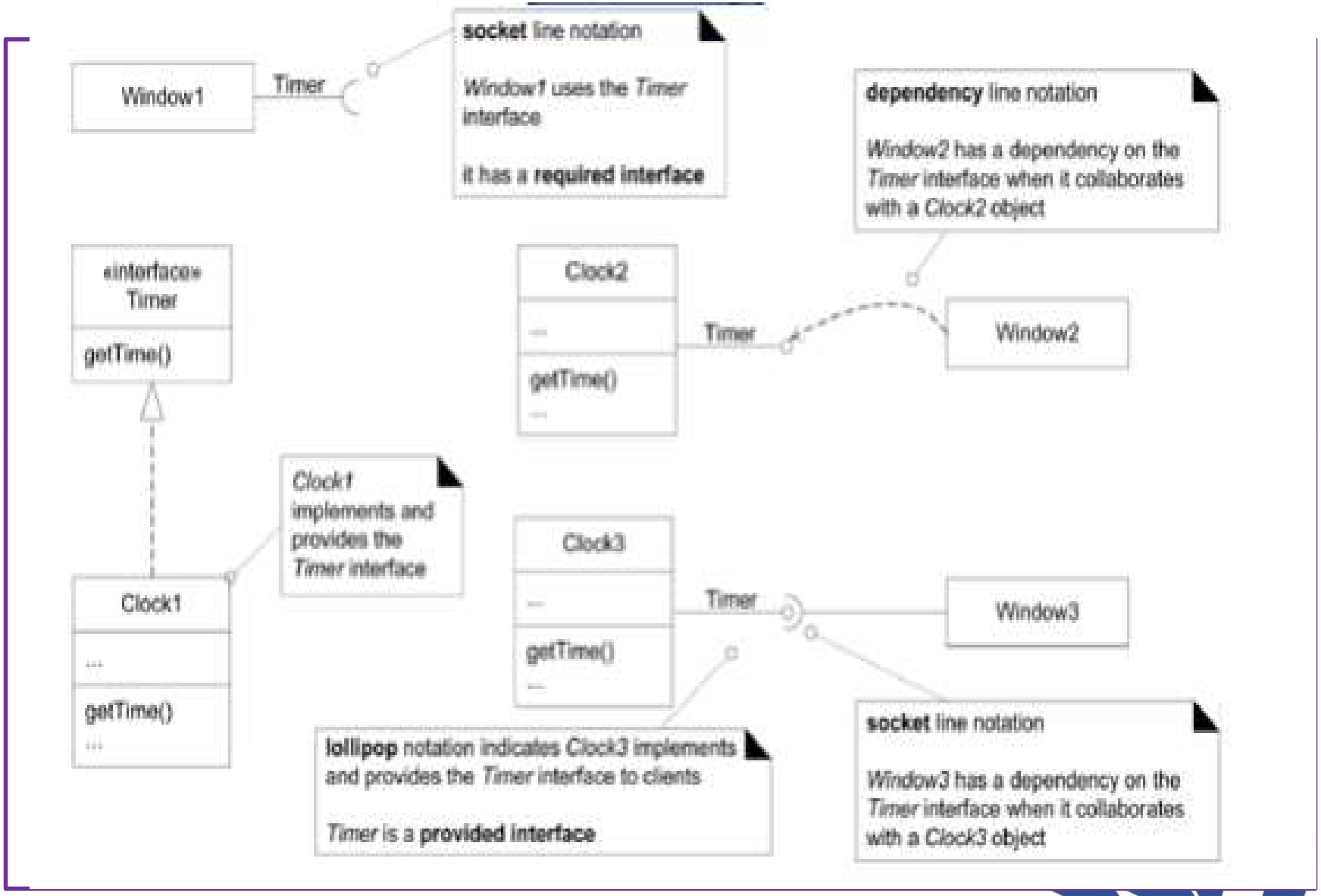


Interfaces

Interfaces

- The UML provides several ways to show **interface** implementation, providing an interface to clients, and interface dependency (a **required interface**).
- In the UML, interface implementation is formally called **interface realization**
- The **socket notation** is new to UML 2. It's useful to indicate "Class X requires (uses) interface Y" without drawing a line pointing to interface Y.

Interfaces-example



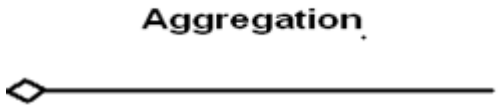
Composition and Aggregation

Aggregation

Definition

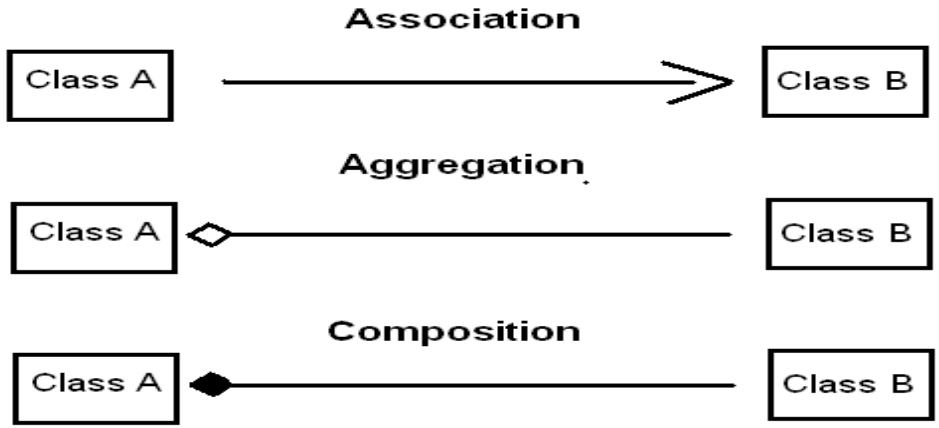
- **Aggregation** is a vague kind of association in the UML that loosely suggests whole-part relationships
- It has no meaningful distinct semantics in the UML versus a plain association, but the term is defined in the UML.
- It normally Posses the "has- a" relationship
- ***Guideline***: Therefore, following the advice of UML creators, don't bother to use aggregation in the UML; rather, use *composition* when appropriate.

Symbol:



Example

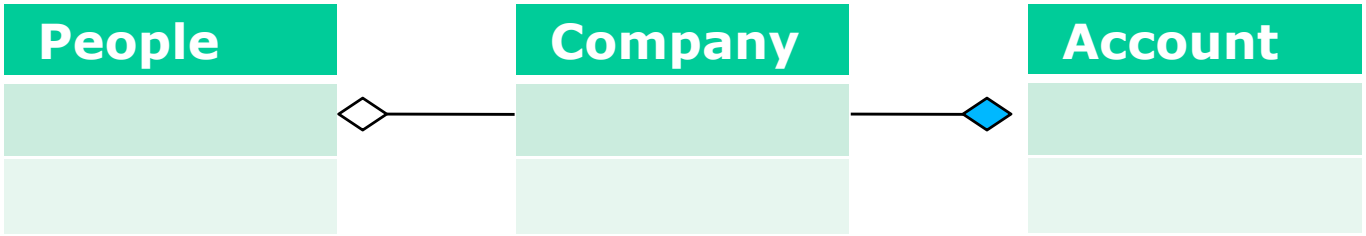
B "uses" A = Aggregation : A exists independently (conceptually) from B



Aggregation

Example

A Company is an aggregation of People. A Company is a composition of Accounts. When a Company ceases to do business its Accounts cease to exist but its People continue to exist.



Composition

Definition

- **Composition**, also known as **composite aggregation**, is a strong kind of whole-part aggregation and *is* useful to show in some models.
- A composition relationship implies that

  1) an instance of the part (such as a *Square*) belongs to only *one* composite instance (such as one *Board*) at a time,

  2) the part must *always belong* to a composite (no free-floating *Fingers),* and

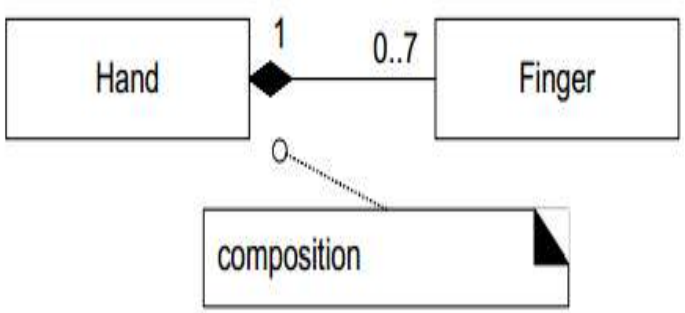
  3) the composite is responsible for the creation and deletion of its parts—either by itself creating/deleting the parts, or by collaborating with other objects.

  Related to this constraint is that if the composite is destroyed, its parts must either be destroyed, or attached to another composite—no free-floating *Fingers* allowed!

Composition

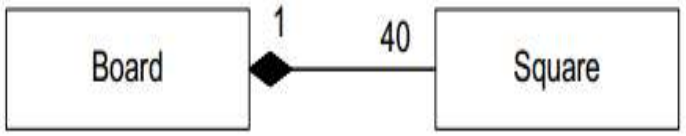
Rule

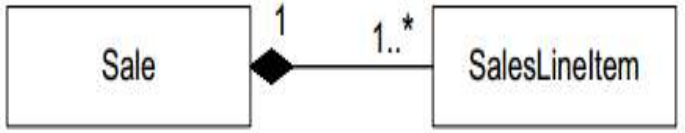
A "owns" B = Composition : B has no meaning or purpose in the system without A



composition means
-a part instance (*Square*) can only be part of one composite (*Board*) at a time

-the composite has sole responsibility for management of its parts, especially creation and deletion





Constraints

Constraints

Definition

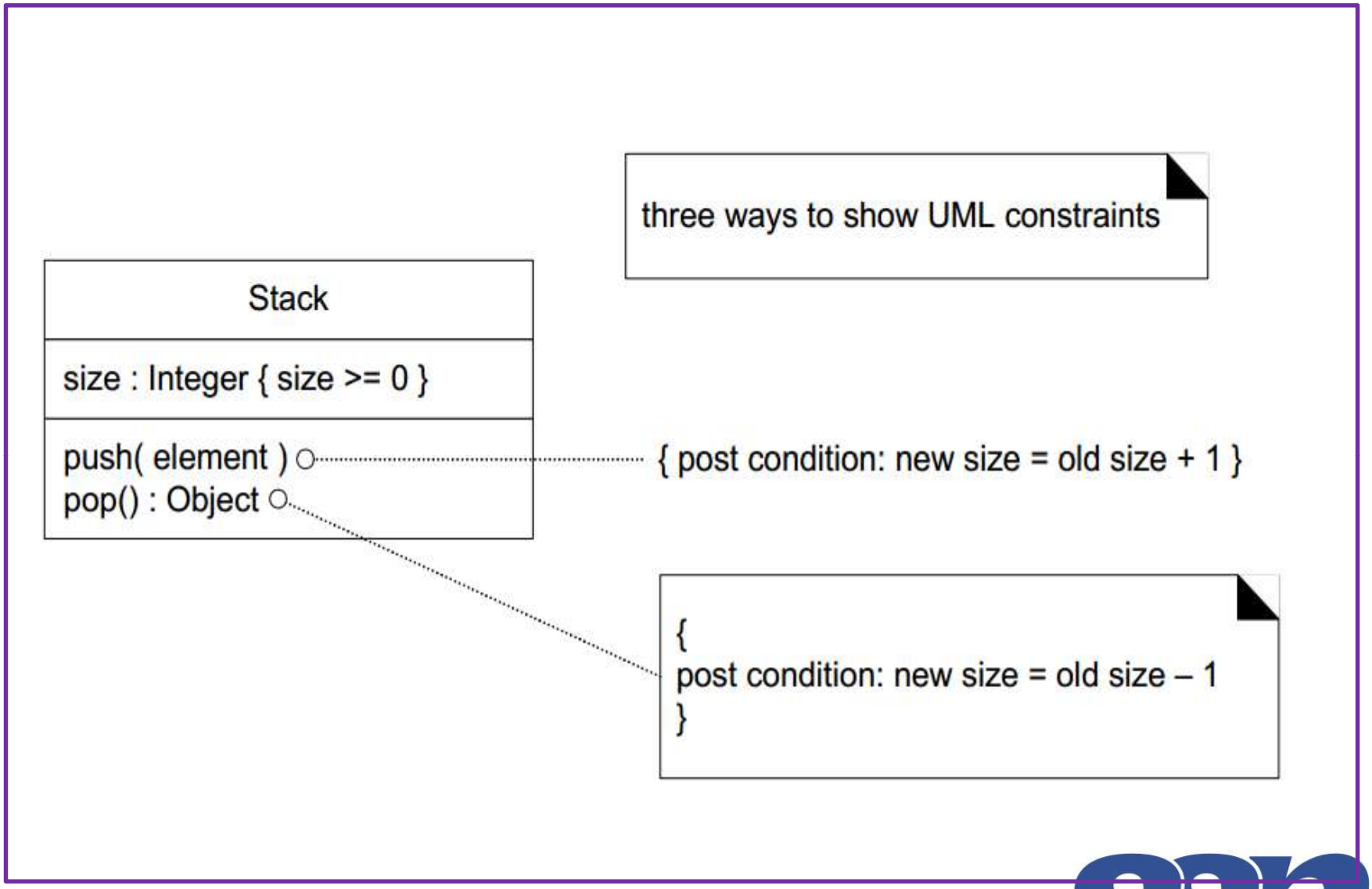
- Constraints may be used on most UML diagrams, but are especially common on class diagrams.
- A UML **constraint** is a restriction or condition on a UML element.
- It is visualized in text between braces;

Example

example: *{ size >= 0 }*.

The text may be natural language or anything else, such as UML's formal specification language, the **Object Constraint Language** (OCL)

Three ways to show UML Constraints



Qualified Association

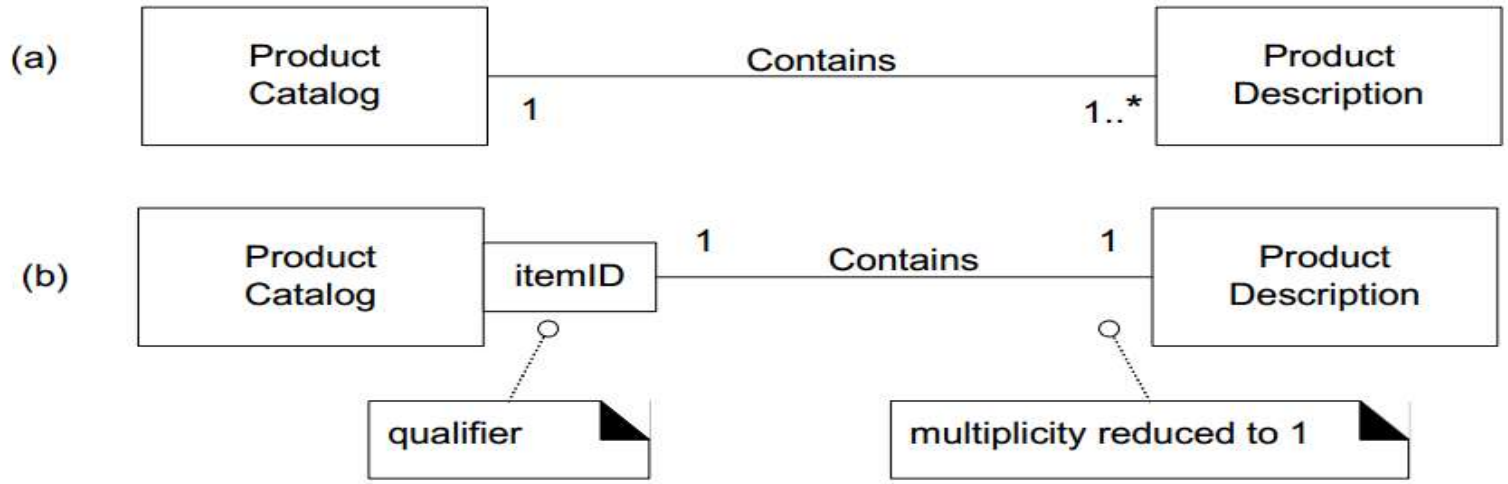
Qualified Association

Definition

- A **qualified association** has a **qualifier** that is used to select an object (or objects) from a larger set of related objects, based upon the qualifier key.
- qualification reduces the multiplicity at the target end of the association, usually down from many to one, because it implies the selection of usually one instance from a larger set.

Example

- For example, if a *ProductCatalog* contains many *ProductDescriptions*, and each one can be selected by an *itemID*



Association Class

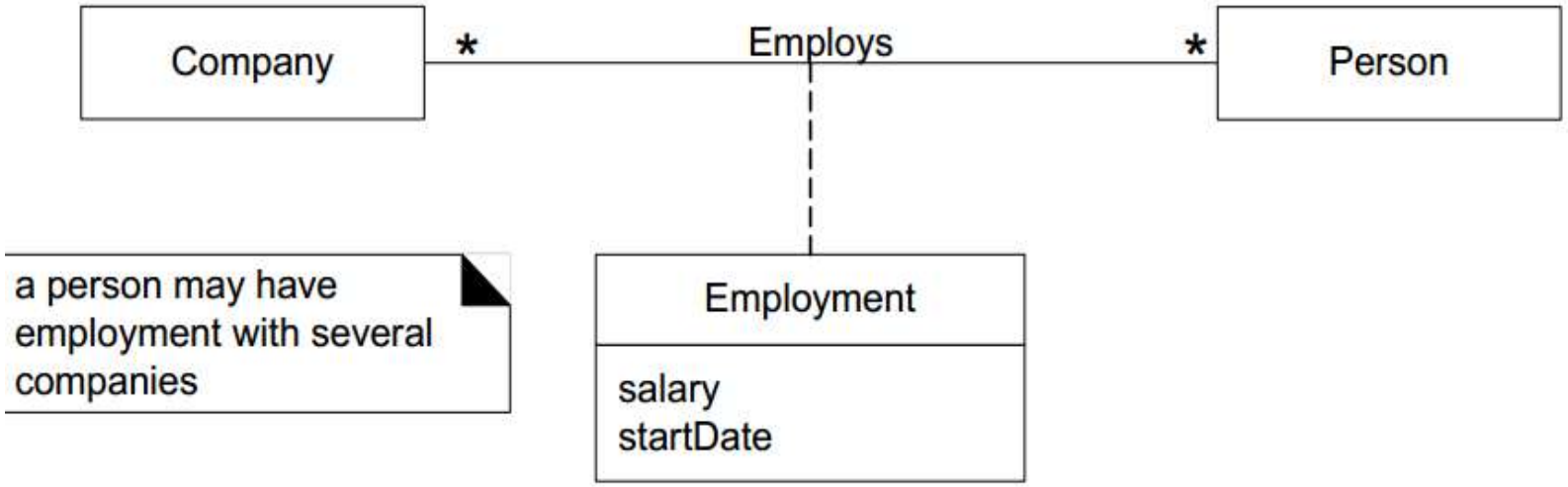
Association class

Definition

- An **association class** allows you treat an association itself as a class, and model it with attributes, operations, and other features.

Example

- For example, if a *Company* employs many *Persons,* modeled with an *Employs* association, you can model the association itself as the *Employment* class, with attributes such as *startDate*.



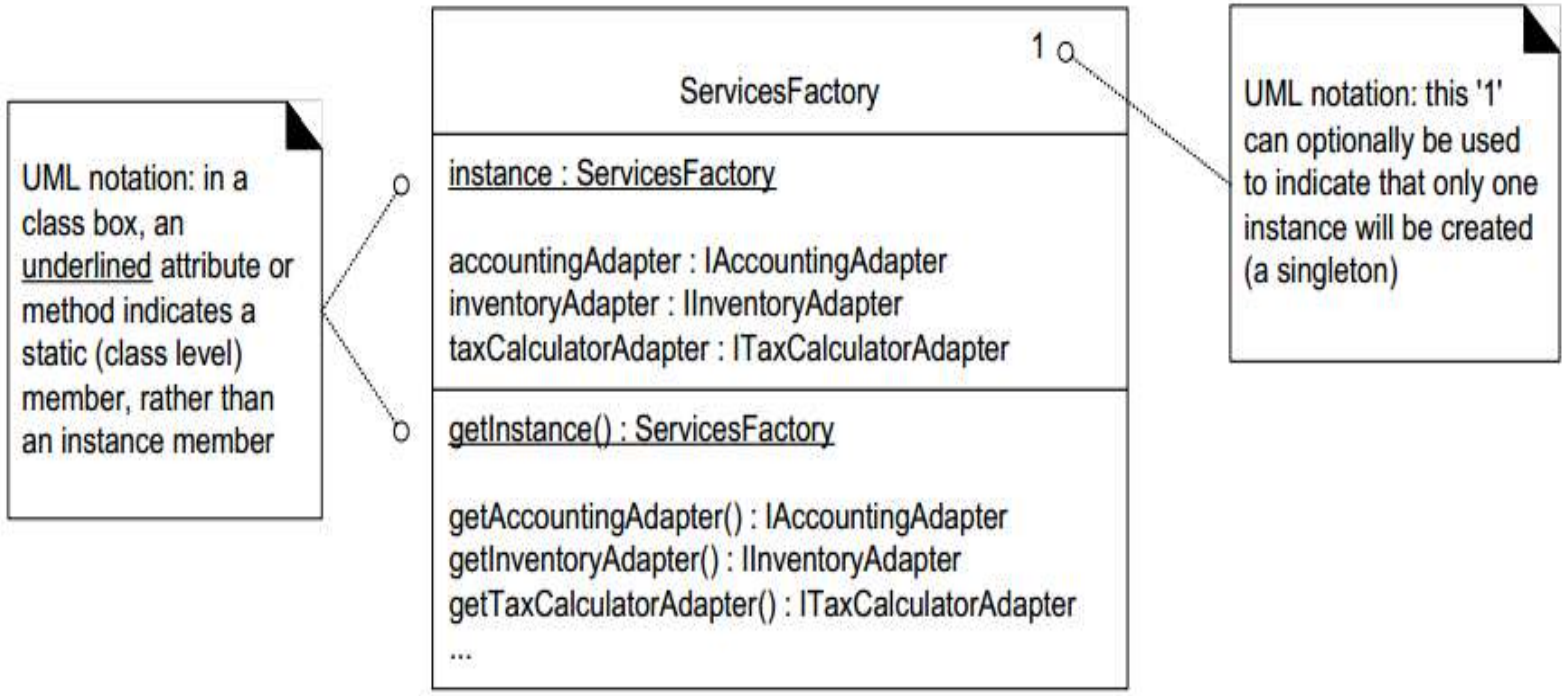
Singleton Class

Singleton class

Definition

- There is only *one* instance of a class instantiated—never two.
- In other words, it is a "singleton" instance.
- In a UML diagram, such a class can be marked with a '1' in the upper right corner of the name compartment.

Example



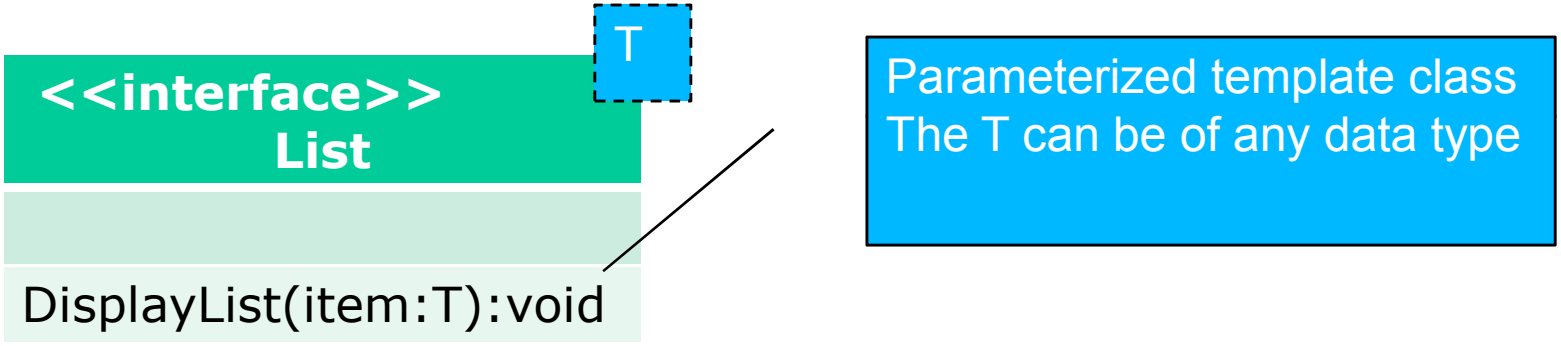
Template Classes and Interfaces

Template classes and interfaces

Definition

- In several programming languages , developer can design the class without specifying the exact data type by which a class can operate.
- UML allows the use of template for this purpose
- While representing the template class a dotted box containing the template parameter is shown at the upper right corner of the class.

Example



Template classes and interfaces

Binding

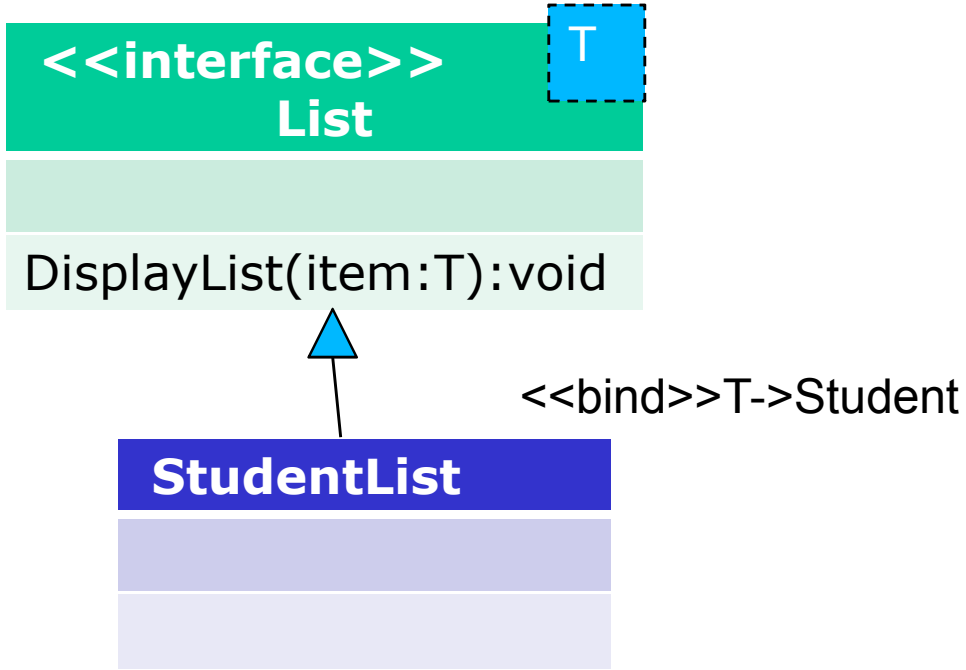
- Associating the data type with a template class is called binding
- 2 types of binding 1. implicit binding 2. Explicit binding

Implicit Binding

- The class can be named with template arguments

List<T->student>

Explicit Binding : In explicit binding the stereotype relationship is shown



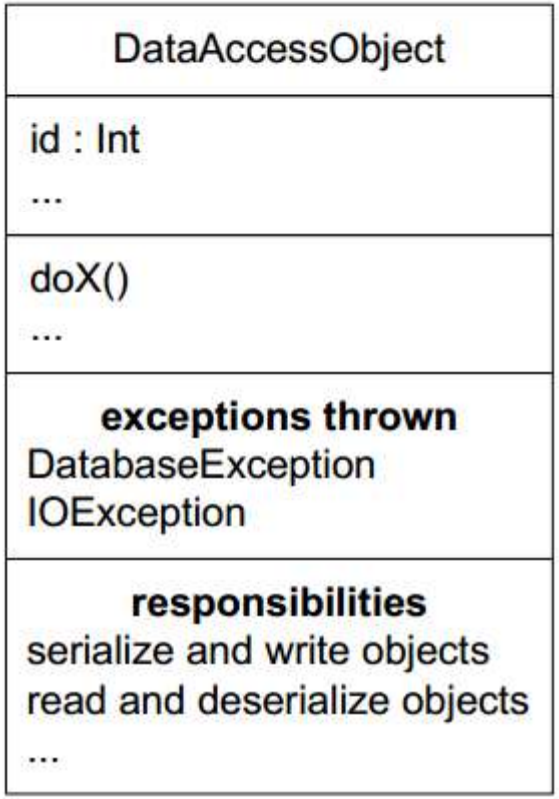
User Defined Compartments

User Defined Compartments

Definition

- In addition to common predefined **compartments** class compartments such as name, attributes, and operations, user-defined compartments can be added to a class box.

Example



Active Class

Active Class

Definition

- An **active object** runs on and controls its own thread of execution.
- An active class indicates that, when instantiated, the class controls its own execution. Rather than being invoked or activated by other objects, it can operate standalone and define its own thread of behavior.
- In the UML, it may be shown with double vertical lines on the left and right sides of the class box

Example