

Object-oriented Analysis and Design

**Applying UML and Patterns**

An Introduction to  
Object-oriented Analysis  
and Design  
and Iterative Development

Part III Elaboration Iteration I – Basic<sup>2</sup>

Software Engineering

Object-oriented Analysis and Design

**Chap 16**  
**UML Class Diagrams**

Software Engineering

Object-oriented Analysis and Design

**Introduction**

- Objective of this chapter
  - Provide a reference for frequently used UML class diagram notation
- The UML includes **class diagrams** to illustrate classes, interfaces, and their associations. They are used for **static object modeling**

Software Engineering

Object-oriented Analysis and Design

**Common UML class diagram notation 1**

3 common compartments  
1. classifier name  
2. attributes  
3. operations

an interface shown with a keyword

interface implementation and subclassing

dependency

officially in UML, the top format is used to distinguish the package name from the class name; unofficially, the second alternative is common

Software Engineering

Object-oriented Analysis and Design

**Common UML class diagram notation 2**

interface implementation and subclassing

association with multiplicities

- ellipsis "... " means there may be elements, but not shown  
- a blank compartment officially means "unknown" but as a convention will be used to mean "no members"

Software Engineering

Object-oriented Analysis and Design

**Design Class Diagram (DCD)**

- The same UML diagram can be used in multiple perspectives
  - In a conceptual perspective the class diagram can be used to visualize a domain model.
  - Class diagram is used in a software or design perspective, called *design class diagram (DCD)*

Domain Model conceptual perspective

Design Model DCD - software perspective

Software Engineering

Object-oriented Analysis and Design

## 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**.

Software Engineering

Object-oriented Analysis and Design

## Ways to Show UML Attributes

- Ways to Show UML Attributes:
  - Attribute Text and Association Lines
- Attributes of a classifier are shown several ways:
  - attribute text notation, such as `currentSale : Sale`.
  - association line notation
  - both together
- The full format of the attribute text notation is:
  - `visibility name : type multiplicity = default {property-string}`
- **Guideline:** *Attributes are usually assumed private if no visibility is given*

Software Engineering

Object-oriented Analysis and Design

## Ways to Show UML Attributes 1

using the attribute text notation to indicate Register has a reference to one Sale instance

OBSERVE: this style visually emphasizes the connection between these classes

using the association notation to indicate Register has a reference to one Sale instance

thorough and unambiguous, but some people dislike the possible redundancy

Software Engineering

Object-oriented Analysis and Design

## Ways to Show UML Attributes 2

the association name, common when drawing a domain model, is often excluded (though still legal) when using class diagrams for a software perspective in a DCD

UP Domain Model conceptual perspective

UP Design Model DCD software perspective

**Attribute text versus association line notation for a UML attribute**

Software Engineering

Object-oriented Analysis and Design

## Ways to Show UML Attributes 3

- **Guideline:** When showing attributes-as-associations, follow the style in DCDs, which is suggested by the UML specification. (Fig 16.4 upper)
- **Guideline:** when using class diagrams for a domain model do show association names but avoid navigation arrows, as a domain model is not a software perspective.

Software Engineering

Object-oriented Analysis and Design

## Ways to Show UML Attributes 4

- **Guideline: When to Use Attribute Text versus Association Lines for Attributes**
  - Use the attribute text notation for **data type objects** and the association line notation for others.
  - Both are semantically equal, but showing an association line to another class box in the diagram (as in Figure 16.3) gives **visual emphasis** - it catches the eye, emphasizing the connection between the class of objects on the diagram.

Software Engineering

Object-oriented Analysis and Design

### Ways to Show UML Attributes 5

applying the guideline to show attributes as attribute text versus as association lines

Register has THREE attributes:  
1. id  
2. currentSale  
3. location

```

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

Software Engineering 17

Object-oriented Analysis and Design

### Ways to Show UML Attributes 6

How to Show Collection Attributes with Attribute Text and Association Lines ?

```

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

Software Engineering 14

Object-oriented Analysis and Design

### Ways to Show UML Attributes 7

Two ways to show a collection attribute

notice that an association end can optionally also have a property string such as {ordered, List}

Software Engineering 15

Object-oriented Analysis and Design

### Operations and Methods 1

- Operations  
*visibility name (parameter-list) : return-type [property-string]*
- Guideline: Assume the version that includes a return type.
- Guideline: **Operations are usually assumed public if no visibility is shown.**
- Example
  - + 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 implementation

Software Engineering 16

Object-oriented Analysis and Design

### Operations and Methods 2

How to Show Methods in Class Diagrams?

- in interaction diagrams, by the details and sequence of messages
- in class diagrams, with a UML note symbol stereotyped with «method»

```

«method»
// pseudo-code or a specific language is OK
public void enterItem( id, qty )
{
    ProductDescription desc= catalog.getProductDescriptor(id);
    sale.makeLineItem(desc, qty);
}
    
```

Software Engineering 17

Object-oriented Analysis and Design

### Keywords 1

- 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 on p. 91 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>'.

Software Engineering 18

Object-oriented Analysis and Design

## Keywords 2

- Most keywords are shown in guillemet (« ») but some are shown in curly braces, such as {abstract}, which is a constraint containing the abstract keyword.
- In general, when a UML element says it can have a "property string" - such as a UML operation and UML association end have - some of the property string terms will be keywords used in the curly brace format.

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

Software Engineering

Object-oriented Analysis and Design

## Stereotypes, Profiles, and Tags

- Stereotypes**
  - are shown with guillemets symbols
  - represents a refinement of an existing modeling concept and is defined within a UML profile
  - 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
- Profiles**
  - a collection of related stereotypes, tags, and constraints to specialize the use of the UML for a specific domain or platform
  - For example, UML profile for project management or for data modeling.

Software Engineering

Object-oriented Analysis and Design

## Stereotypes

declaring the stereotype

UML extension relationship to a basic UML metamodel term - Element

using the stereotype

a tool will probably allow a popup to fill in the tag values once an element has been stereotyped with «authorship»

Stereotype declaration and use

Software Engineering

Object-oriented Analysis and Design

## Property and Property String

- 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.
- Textual presentation approach
  - UML property string {name1=value1, name2=value2}
  - such as {abstract, visibility=public}.
  - Some properties are shown without a value, such as {abstract};

Software Engineering

Object-oriented Analysis and Design

## Abstract Classes and Abstract Operations

- Abstract classes and operations can be shown either with an {abstract} tag (useful when sketching UML) or by italicizing the name (easy to support in a UML tool).
- The opposite case, final classes and operations that can't be overridden in subclasses, are shown with the {leaf} tag.

Software Engineering

Object-oriented Analysis and Design

## Constraint

- Constraints**
  - 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;
    - for example: { size >= 0 }.
  - The text may be natural language or anything else, such as Object Constraint Language (OCL)

Software Engineering

Object-oriented Analysis and Design

### Singleton Classes

UML notation : in a class box - an underlined attribute or method indicates a static (class level) member, rather than an instance member

UML notation : this "1" can optionally be used to indicate that only one instance will be created (a singleton.)

Software Engineering 25

Object-oriented Analysis and Design

### Generalization

- Generalization in the UML is shown with a solid line and fat triangular arrow from the subclass to superclass
- 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.
- Generalization → inheritance?
  - It depends. In a domain model conceptual-perspective class diagram, the answer is no.
  - In a DCD software-perspective class diagram, it implies OOP inheritance from the superclass to subclass.

Software Engineering 26

Object-oriented Analysis and Design

### Dependency 1

- 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**
- There are many kinds of dependency
  - having an attribute of the supplier type
  - sending a message to a supplier; the visibility to the supplier could be:
    - an attribute, a parameter variable, a local variable, a global variable, or class visibility (invoking static or class methods)
  - receiving a parameter of the supplier type
  - the supplier is a superclass or interface

Software Engineering 27

Object-oriented Analysis and Design

### Dependency 2

- All of these could be shown with a dependency line in the UML, but some of these types already have special lines that suggest the dependency
- When to show a dependency?
  - Guideline:** In class diagrams use the dependency line to depict global, parameter variable, local variable, and static-method (when a call is made to a static method of another class) dependency between objects.

```

public class Sale
{
    public void updatePriceFor( ProductDescription description )
    {
        Money basePrice = description.getPrice();
        //...
    }
}
    
```

Software Engineering 28

Object-oriented Analysis and Design

### Dependency 2

the Sale has parameter visibility to a ProductDescription, and thus some kind of dependency

Showing dependency

Software Engineering 29

Object-oriented Analysis and Design

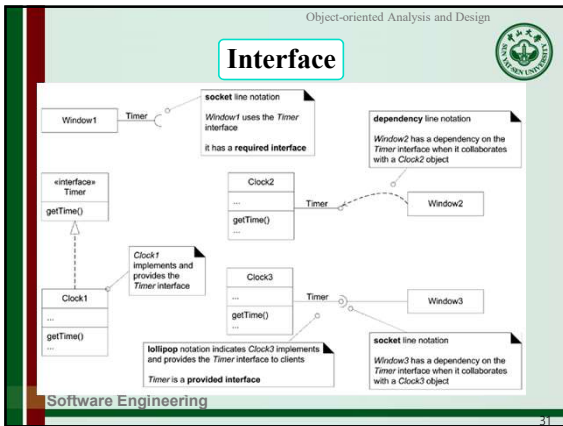
### Dependency 3

the doX method invokes the runFinalization static method, and thus has a dependency on the System class

Showing dependency

Optional dependency labels in the UML

Software Engineering 30



### Composition Over Aggregation 1

★★

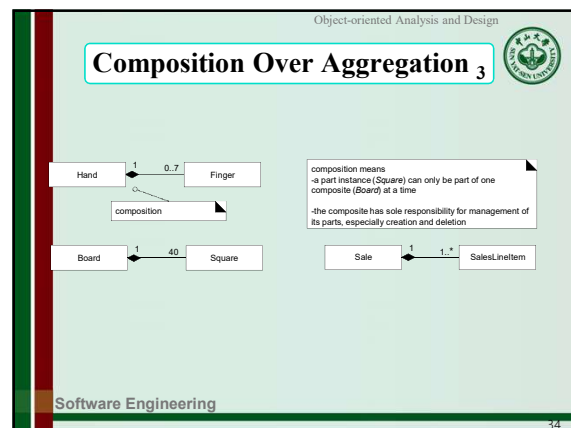
- Aggregation is a vague kind of association in the UML that **loosely suggests** whole-part relationships
  - It has no meaningful distinct semantics in the UML versus a plain association,
  - but the term is defined in the UML. Why?
  - In spite of the few semantics attached to aggregation, everybody thinks it is necessary (for different reasons). Think of it as a modeling placebo. [RJB04]
- Guideline: Therefore, following the advice of UML creators, don't bother to use aggregation in the UML; rather, use composition when appropriate

Software Engineering

### Composition Over Aggregation 2

- 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.
- Guideline: The association name in composition is always implicitly some variation of "Has-part," therefore don't bother to explicitly name the association

Software Engineering

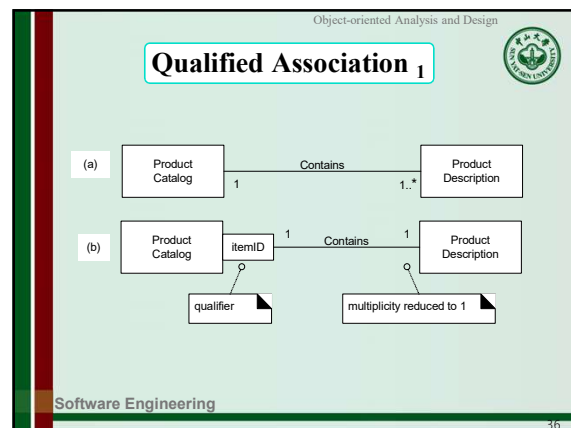


### Association Class

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

a person may have employment with several companies

Software Engineering



Object-oriented Analysis and Design

## Qualified Association 2

- 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.
- Informally, in a software perspective, it suggests looking things up by a key, such as objects in a *HashMap*. For example, if a *ProductCatalog* contains many *ProductDescriptions*, and each one can be selected by an *itemID*

Software Engineering

Object-oriented Analysis and Design

## Relationship Between Interaction and Class Diagrams

Software Engineering

Object-oriented Analysis and Design

## Template Classes and Interfaces 1

- Many languages (Java, C++, ...) support *templated* types, also known (with shades of variant meanings) as templates, parameterized types, and generics.
- They are most commonly used for the element type of collection classes, such as the elements of *lists* and *maps*. For example, in Java, suppose that a *Board* software object holds a *List* (an interface for a kind of collection) of many *Squares*. And, the concrete class that implements the *List* interface is an *ArrayList*:

```

public class Board {
    private List<Square> squares = new ArrayList<Square>();
    // ...
}
    
```

Software Engineering

Object-oriented Analysis and Design

## Template Classes and Interfaces 2

Software Engineering

Object-oriented Analysis and Design

## User-Defined Compartments

DataAccessObject
id : Int ...
doX() ...
<b>exceptions thrown</b>
DatabaseException IOException
<b>responsibilities</b>
serialize and write objects read and deserialize objects ...

Software Engineering

Object-oriented Analysis and Design

## Active Class

- An **active object** runs on and controls its own thread of execution.
- The class of an active object is an active class

Software Engineering