

# Chapter 14 – Object Design

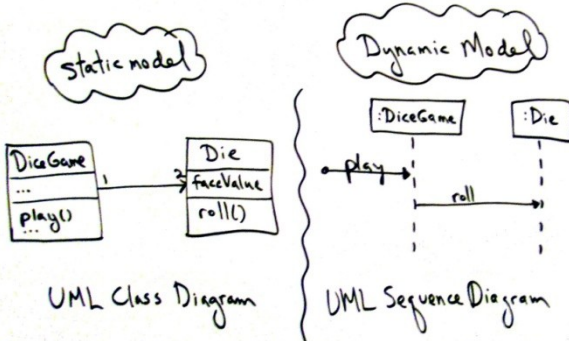
## 3 approaches to object design →

- Code directly from mental model
- Draw model, then code from visual model
  - Use IDE for coding
- Just draw (assume CASE tool will generate code for you)
  - Likely to disappoint

## Agile Design → Agile modeling

- **discourages**
  - Drawing lots of visual models
  - Drawing formal or ‘dressed up’ models
    - Argument is that time is better spent coding
- **encourages**
  - Modeling with others
  - Creating models in parallel
- Very helpful to have tool that reverse engineers diagrams from code

**Static vs. Dynamic modeling →** Dynamic modeling is the key skill since it involves responsibility assignment



There are two kinds of models: dynamic & static.

## Dynamic models → UML interaction diagrams (SD or CD)

help design the logic, the behavior of the code or the method bodies. They tend to be the most interesting, difficult, important diagrams to create.

## Static models → UML DCD

help design the definition of packages, class names, attributes and method signatures (but not method bodies)

Note: during dynamic modeling, we apply responsibility – driven design and GRASP patterns.

Other dynamic tools in UML include State Machine Diagrams

**SMD** and Activity Diagrams **AD**.

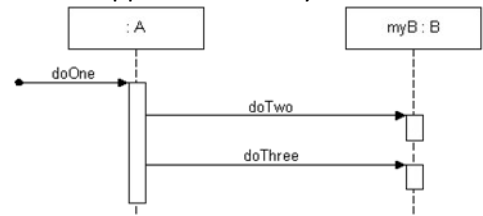
**Interaction Diagrams →** illustrate how SW objects interact via messages. It has two common types: Sequence Diagrams **SD** and Communication or Collaboration Diagrams **CD**

A related diagram is **interaction overview diagram**; it provides a big picture overview of how a set of interaction diagrams are related in terms of logic and process flow. (new in UML.2)

## SD vs. CD

**Sequence Diagrams SD →** illustrates interactions in a kind of fence format, in which each new obj. is added to the right.

- Easier to see sequence of method calls over time
- More expressive UML notation
- Better support from many tools



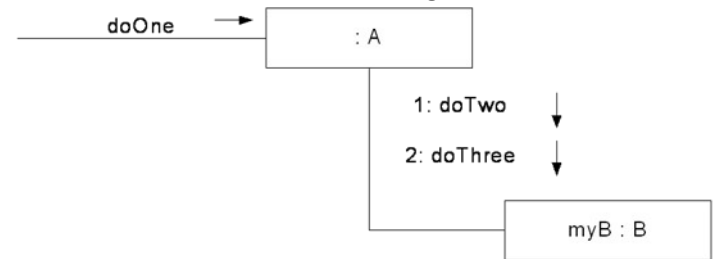
```
public class A
{
    private b myB = new B();

    public void doOne()
    {
        myB.doTwo();
        myB.doThree();
    }
}
```

- Sequence diagram's 'fence' format
- Time is increasing in the downward direction

**Communication Collaboration Diagrams CD →** illustrates objects interactions in a graph or network format, in which obj. can be place anywhere on the diagram.

- Better fit on limited space (page or whiteboard)
- Easier to edit/amend
- Look more like class diagram



Same collaboration using communication diagram  
Uses network (or graph) format

# Chapter 15 – UML Interaction Diagrams

UML Interaction Diagrams		
-	SD	CD

<b>Strengths</b>	<ol style="list-style-type: none"> <li>Clearly shows sequence or time ordering of messages.</li> <li>Large set of UML detailed notation options.</li> <li>Better support from many tools</li> </ol>	<ol style="list-style-type: none"> <li>Space economical -&gt; better fit on limited space</li> <li>Flexibility to add new obj. in two dimensions.</li> <li>Look more like class diagram</li> </ol>
<b>Weaknesses</b>	<ol style="list-style-type: none"> <li>Forced to extend to the right when adding new obj.</li> <li>Consumes horizontal space.</li> </ol>	<ol style="list-style-type: none"> <li>More difficult to see sequence of messages.</li> <li>Fewer notation options.</li> </ol>

**Essential UML models for OOAD** →

- Use cases
  - Functional requirements
- Class diagram
  - Objects with knowledge (attributes) and behavior (operations)
  - Static relationships between objects
- Interaction diagrams
  - Dynamic collaboration between objects

# Chapter 16 – UML Class Diagrams

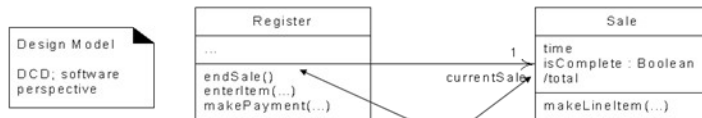
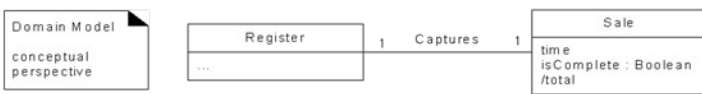
**Class Diagrams** → to illustrate classes, interfaces and their associations. They are used for **static object modeling**.

**Common Class Diagram Notation** → 3 Common compartments:

- Classifier Name**
- Attributes** → with +/- visibility
- Operations** → with +/- visibility

**UML Class Diagrams in two perspectives**

Domain model is the analysis class diagram  
Don't show methods



Avoid showing no-argument constructors & getters/setters

Design model shows methods and visibility (arrowhead on association)  
Register has reference to Sale; Sale does not have reference to Register

**Classifier** → a model element that describes behavioral and structure features. Classifiers can also be specialized. They are a generalization of many of UML elements including classes, interfaces, UCs, and actors. In DCD, the two most common classifiers are regular classes and interfaces.

**Attribute of a classifier** → (also called structural properties in UML.1) has 3 notation:

- Attribute text notation e.g. CurrentSale : Sale
- Association line notation
- Both

Attributes have Visibility:  
**+** (public)  
**-** (private)

Full format of the attribute:

visibility attributeName : type multiplicity = default {property-string}

**Operations** → one of the compartments of the UML class shows the signature of operations.

visibility operationName(parameter list) : return-type {property-string}

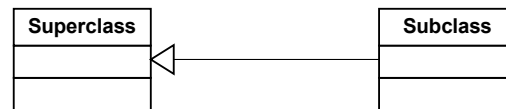
**Guidelines**

- Show return types
- Assume operations are public unless otherwise depicted
- Assume getters & setters, i.e., don't show them on model

**Methods** →

**Generalization**

- Same as inheritance when in design mode
- Solid line with open arrow

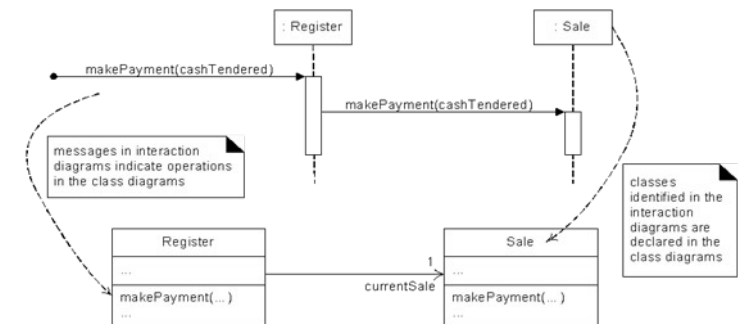


- To indicate *abstract*, use italics
- For final classes or operations, use {leaf}

**Dependency**

- When a client has knowledge of a server
- Changes in server affects client
- Client and server are 'coupled'
- NOT a good thing! Leads to brittle design
- Occurs anytime client has direct visibility to server
  - Can also be more subtle

**The influence of interaction diagrams on class diagrams**



Relationship between class & interaction diagrams  
Models must be consistent with each other!!