CS 619 Introduction to OO Design and Development

UML Interaction Diagram and Class Diagram

Fall 2012

On to Object Design

- Design collaborating objects to fulfill project requirement.
  - CRC cards
- Other ways to design objects?

1) Code
   - Design-while-coding (Java, C#, …). From mental model to code.

2) Draw, then code
   - Drawing some UML on a whiteboard or UML CASE tool, then switching to #1 with a text-strong IDE (e.g., Eclipse or Visual Studio).

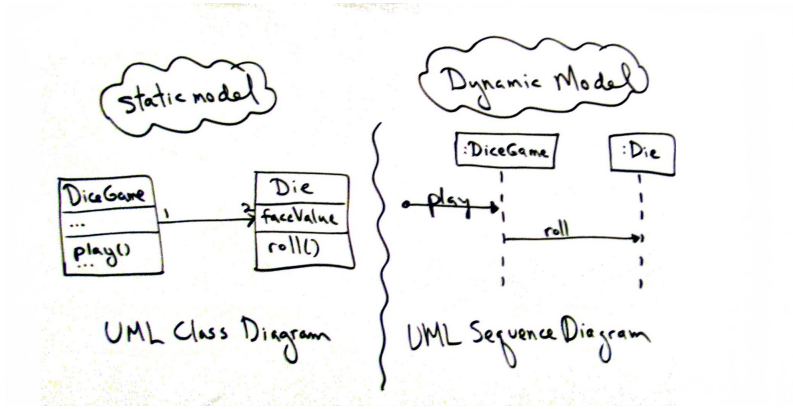
3) The text introduces lightweight drawing before coding,

Designing Objects: Static and Dynamic Modeling

- There are two kinds of object models: **dynamic** and **static**.
- **Dynamic models**,
  - Such as UML interaction diagrams (sequence diagrams or communication diagrams), help design the logic, the behavior of the code or the method bodies.
    - They tend to be the more interesting, difficult, important diagrams to create.

Designing Objects: Static and Dynamic Modeling

- **Static models**, such as UML class diagrams, help design the definition of packages, class names, attributes, and method signatures (but not method bodies).



Designing Objects: Static and Dynamic Modeling

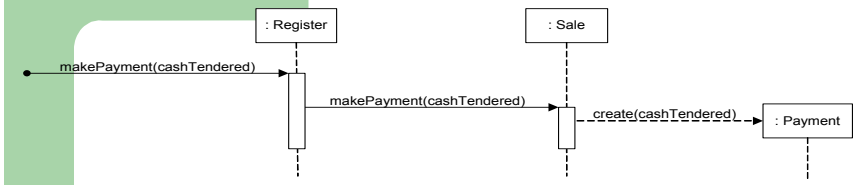
- It's during *dynamic object modeling that helps us to think* what objects need to exist and how they collaborate via messages and methods.
- Guideline:
  - Spend significant time doing interaction diagrams (e.g. sequence diagrams), not just class diagrams.
- Ignoring this guideline is a very common worst-practice with UML.

- While drawing a UML object diagram, we need to answer key questions:
  - **What are the responsibilities of the object?**
  - **Who does it collaborate with?**
  - **What design patterns should be applied?**
- Far more important than knowing the difference between UML 1.4 and 2.0 notation!
- Fundamental object design requires knowledge of:
  - principles of responsibility assignment
  - design patterns

UML Interaction Diagrams

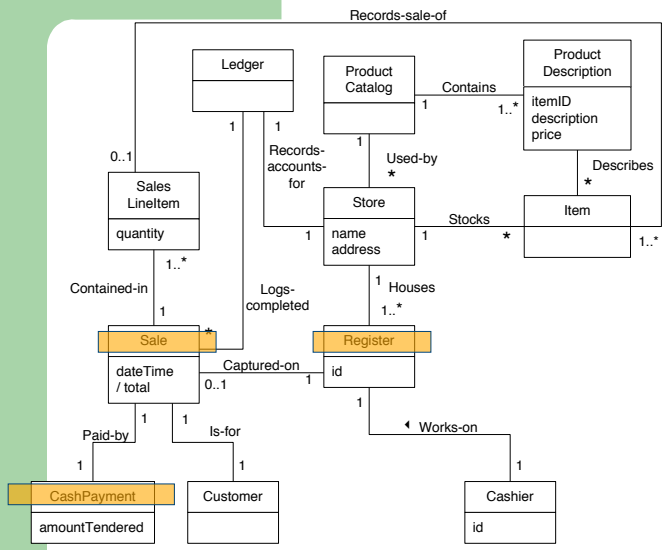
- The UML includes ***interaction diagrams*** to illustrate how objects interact via messages.
- Used for dynamic object modeling.
- There are two common types:
  - **sequence diagrams**.
  - **communication diagrams**.

Example Sequence Diagrams



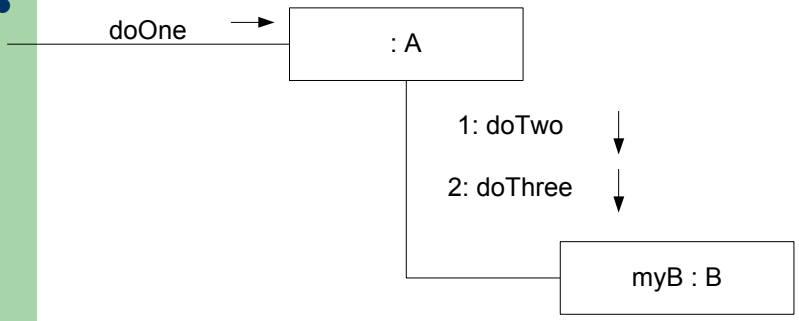

```
public class Sale {
        private Payment payment;

        public void makePayment( Money cashTendered ) {
                payment = new Payment( cashTendered );
                //…
        }
        // …
} `
```

Example

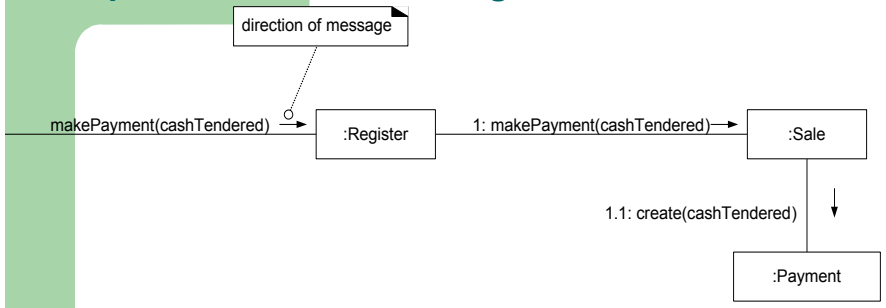


UML Communication diagrams

- ***Communication diagrams*** illustrate object interactions in a graph or network format, in which objects can be placed anywhere on the diagram (the essence of their wall sketching advantage).
-



Example Communication Diagrams



•Although the static-view class diagrams are indeed useful, the dynamic-view interaction diagrams, or the acts of dynamic interaction modeling, are incredibly valuable.

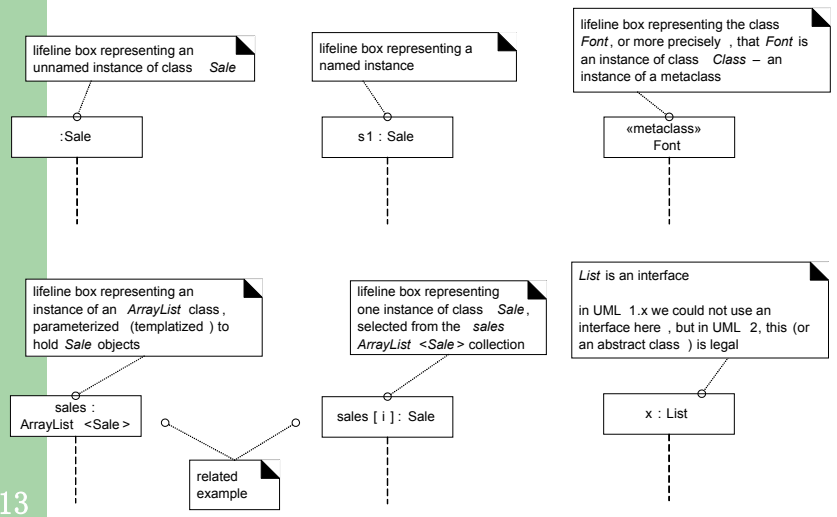
•Spend time doing dynamic object modeling with interaction diagrams, not just static object modeling with class diagrams

Sequence v. Communication Diagrams

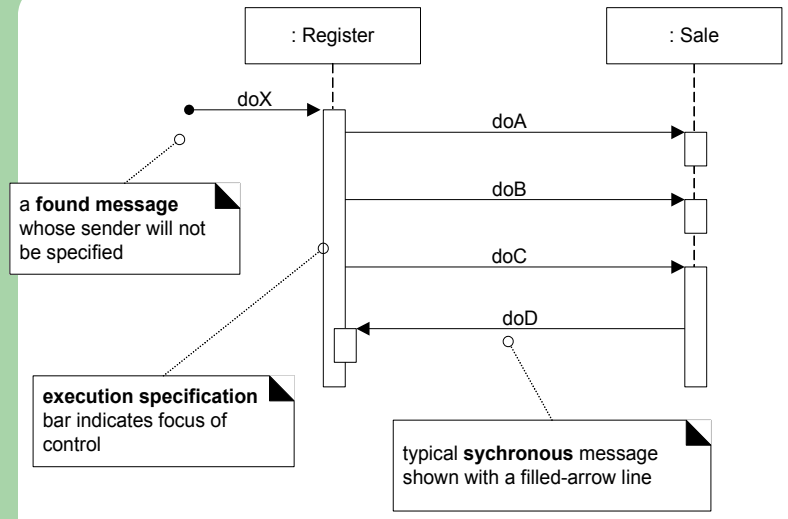
Type	Strengths	Weaknesses
sequence	clearly shows sequence or time ordering of messages large set of detailed notation options	forced to extend to the right when adding new objects; consumes horizontal space
communication	space economicalflexibility to add new objects in two dimensions	more difficult to see sequence of messages fewer notation options

Common UML Interaction Diagram Notation

Illustrating Participants with Lifeline Boxes

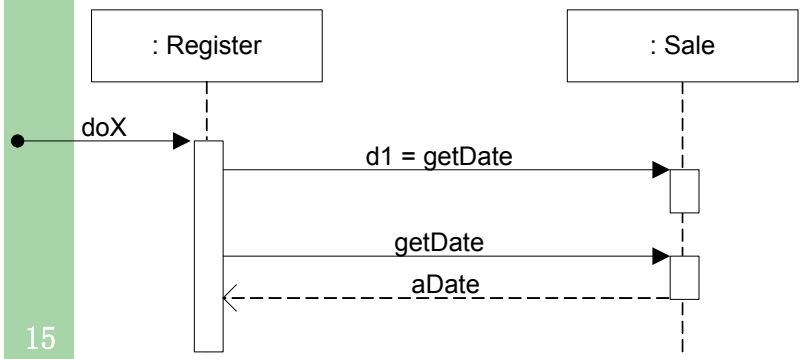


Messages, Focus Control and Execution Spec. Bar



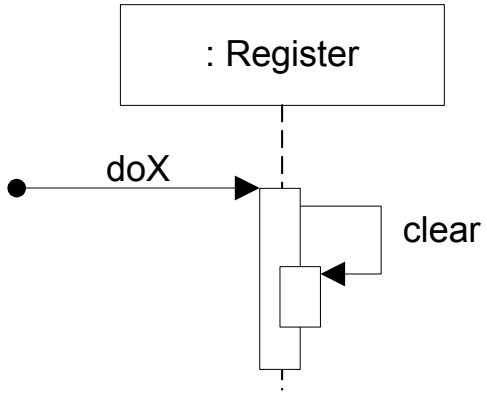
Illustrating Reply or Returns

- There are two ways to show the return result from a message:
  - Using the message syntax returnVar = message(parameter).
  - Using a reply (or return) message line at the end of an execution spec. bar.



Message to “self” or “this”

- You can show a message being sent from an object to itself by using a nested Exec. Spec. bar



Creation of Instance

- The typical interpretation (in languages such as Java or C#) of a *create* message on a ***dashed line*** with a filled arrow is "invoke the new operator and call the constructor".

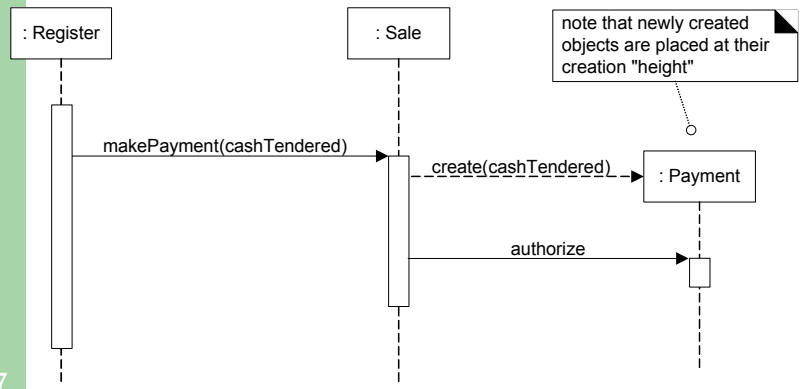


Diagram Frames in UML Sequence Diagram

- To support conditional and looping constructs (among many other things), the UML uses frames.
- Frames are regions or fragments of the diagrams; they have an operator or label (such as loop) and a guard(conditional clause).

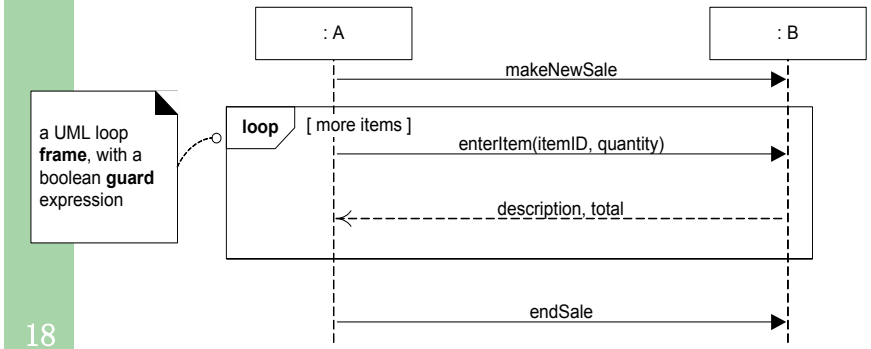
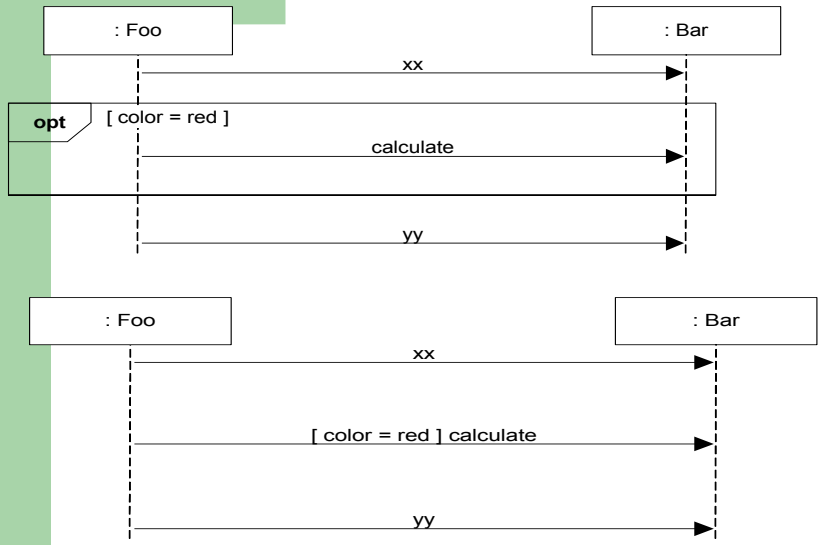
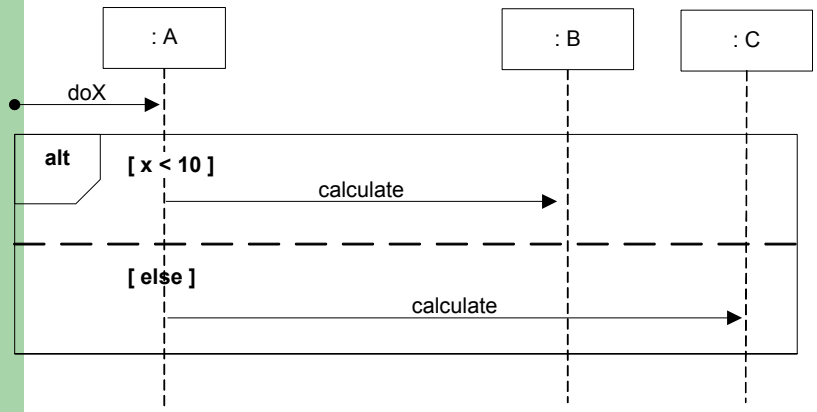


Diagram Frames in UML Sequence Diagram



opt	Optional fragment that executes if guard is true.

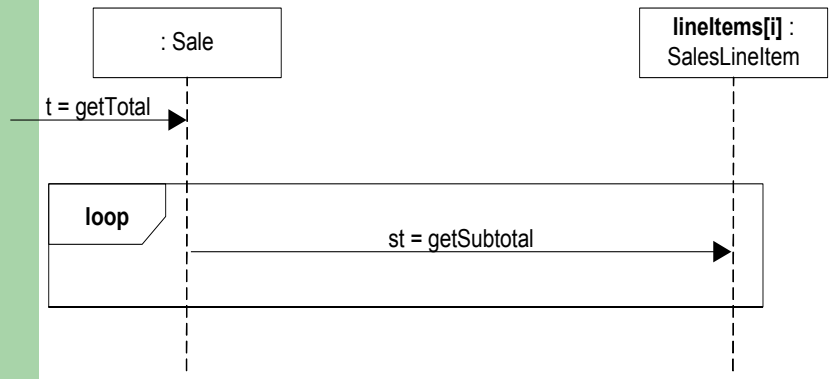
Diagram Frames in UML Sequence Diagram



alt	Alternative fragment for mutual exclusion conditional logic expressed in the guards.

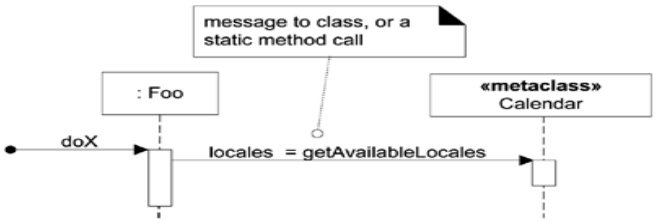
Diagram Frames in UML Sequence Diagram

Basic Sequence Diagram Notation



loop	Loop fragment while guard is true. Can also write *loop(n)* to indicate looping n times. There is discussion that the specification will be enhanced to define a *FOR* loop, such as *loop(i, 1, 10)*

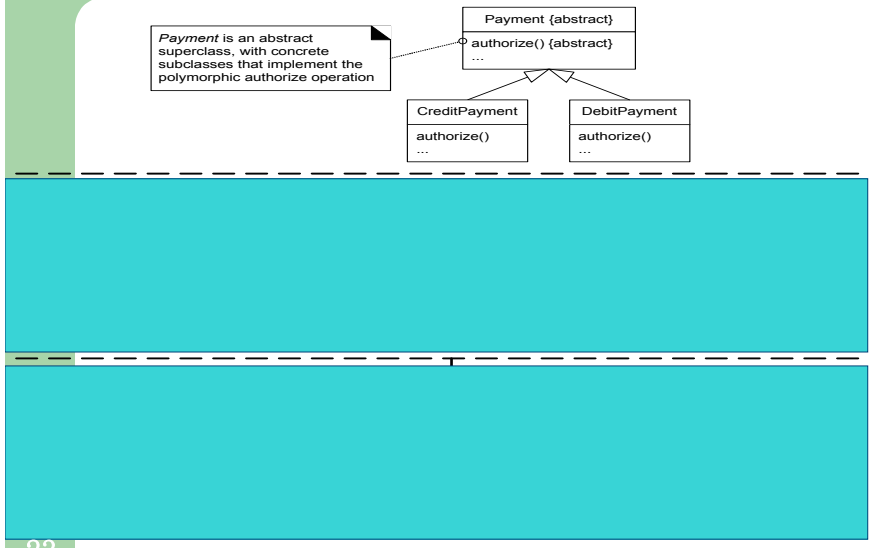
Message to Classes to Invoke Static Methods



In code, a likely implementation is:

```
public class Foo
{
public void doX()
{
   // static method call on class Calendar
   Locale[] locales = Calendar.getAvailableLocales();
   // …
}
// …
}
```

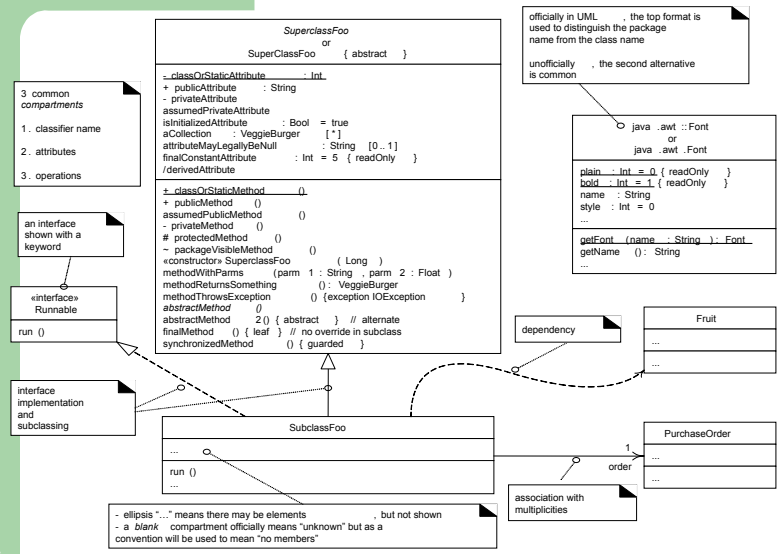

Polymorphic Messages and Cases



Design Class Diagram

- Note: same UML diagram can be used in multiple perspectives.
  - In a conceptual perspective the class diagram can be used to visualize a domain model.
- Now we use the class diagram is used in a software or design perspective.
- A common modeling term for this purpose is **Design Class Diagram (DCD)**.
- In the UP, the set of all DCDs form part of the **Design Model**. Design Model also includes UML interaction diagrams.

UML Class Diagram

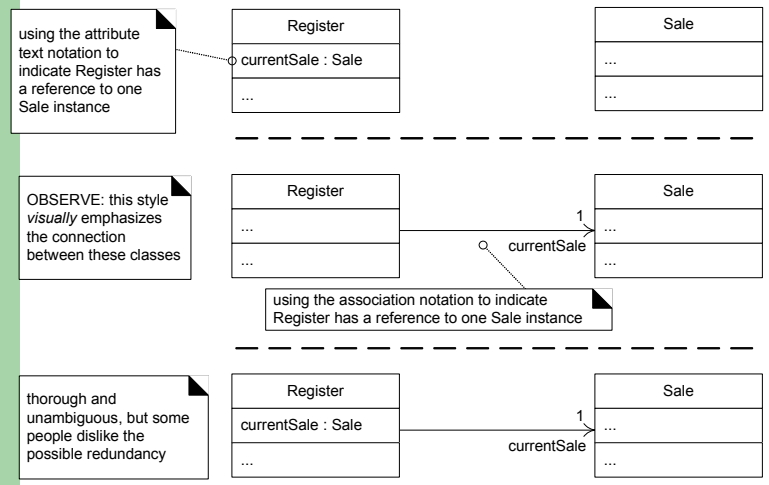


UML Classifiers and Attributes

- A UML classifier is "a model element that describes behavioral and structure features".
- In class diagrams, the two most common classifiers are *regular classes and interfaces*.

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

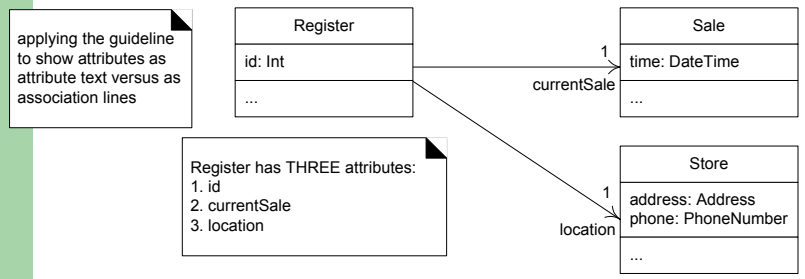
Show UML Attributes: Attribute Text and Association Lines



When to Use Attribute Text versus Association Lines for Attributes?

- ***Guideline:***

  Use the attribute text notation for data type objects and the association line notation for others.

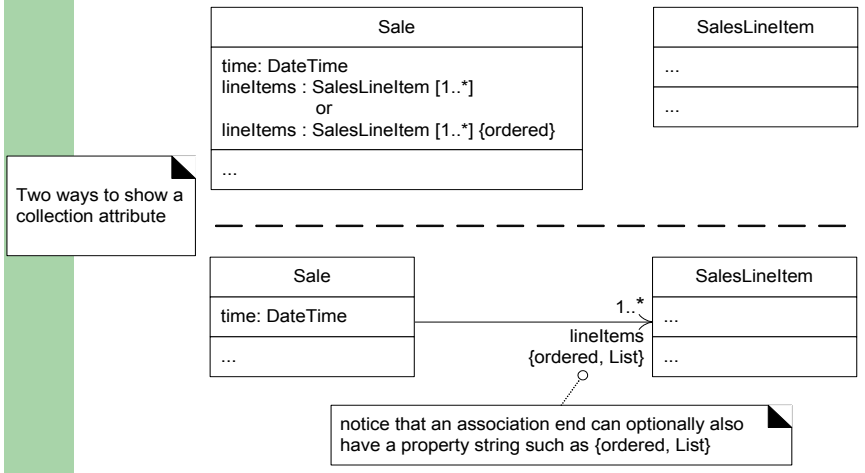


When to Use Attribute Text versus Association Lines for Attributes?

- these different styles exist only in the UML surface notation; in code, they boil down to the same thing - the Register class will have three attributes.
- For example, in Java:

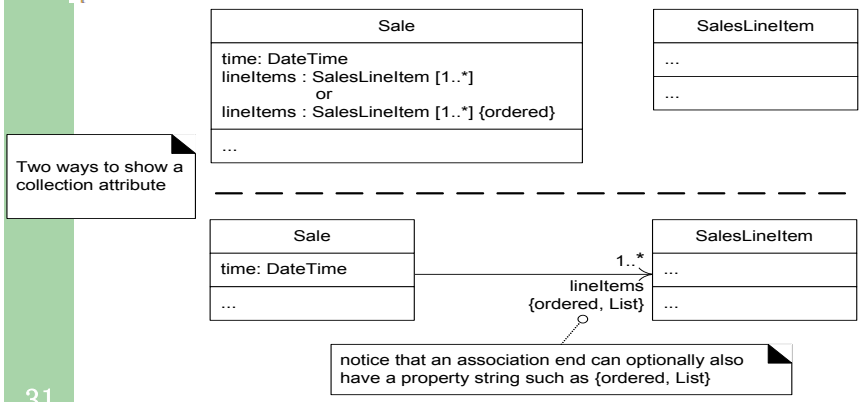
```
public class Register {
   private int id;
   private Sale currentSale;
   private Store location;
// …
}
```


The UML Notation for an Association End



Collection Attributes

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



Operations and Methods

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

  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.

Operations and Methods

- In addition to the official UML operation syntax, the UML allows the operation signature to be written in any programming language
- For example, both expressions are possible:

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

public Player getPlayer( String name ) throws IOException
```


Operations and Methods

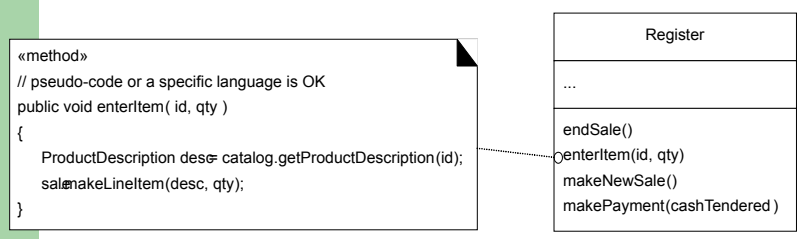
- An operation is not a method. A UML **operation** is a declaration, with a name, parameters, return type, exceptions list, and possibly a set of constraints of pre-and post-conditions.
- But, it isn't an implementation - rather, **methods** are implementations.
- When we explored operation contracts , in UML terms we were exploring the definition of constraints for UML operations.

Show Methods

- 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»

Show Methods

- When we use a ***UML note*** to show a method, we are mixing static and dynamic views in the same diagram.
- The method body (which defines dynamic behavior) adds a dynamic element to the static class diagram.



Operation Issues- The Create Operation

- The create message in an interaction diagram is normally interpreted as the invocation of the new operator and a constructor call in languages such as Java and C#.
- In a DCD this create message will usually be mapped to a constructor definition, using the rules of the language - such as the constructor name equal to the class name (Java, C#, C++, …).

Operation Issues- Operations to Access Attributes

- Accessing operations retrieve or set attributes, such as *getPrice* and *setPrice*.
- These operations are often *excluded (or filtered)* from the class diagram because of the high noise-to-value ratio they generate;
  - for n attributes, there may be 2n uninteresting getter and setter operations. Most UML tools support filtering their display, and it's especially common to ignore them while wall sketching.

Keywords

A UML keyword is a textual adornment to categorize a model element.

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

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

- Dependency is illustrated with a dashed arrow line from the client to supplier.
- some common types in terms of objects and class diagrams :
  - 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

Dependency

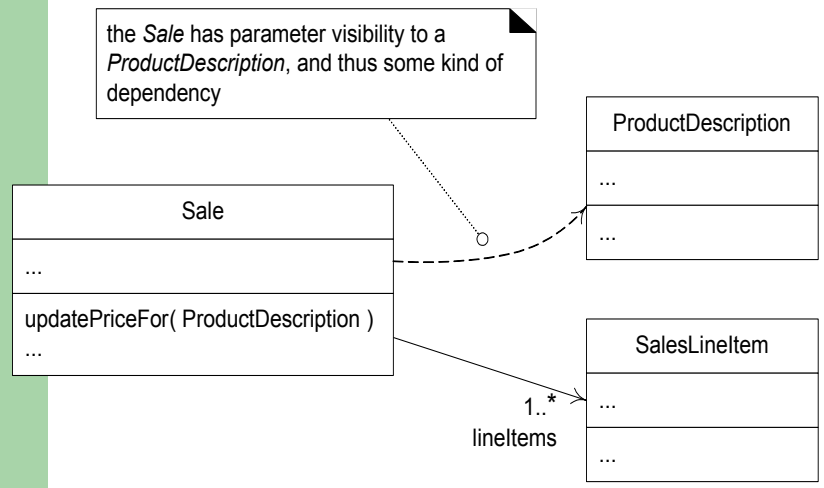
- when to show a dependency?
- ***Guideline:*** In class diagrams use the dependency line to depict global, parameter variable, local variable, and static-method (when a call is made to a static method of another class) dependency between objects.
- For example, the following Java code shows an *updatePriceFor* method in the *Sale* class:

Dependency

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

The ***updatePriceFor*** method receives a ***ProductDescription*** parameter object and then sends it a ***getPrice*** message. Therefore, the Sale object has ***parameter visibility*** to the ProductDescription, and ***message-sending*** coupling, and thus a dependency on the ***ProductDescription***. If the latter class changed, the Sale class could be affected. This dependency can be shown in a class diagram

Dependency



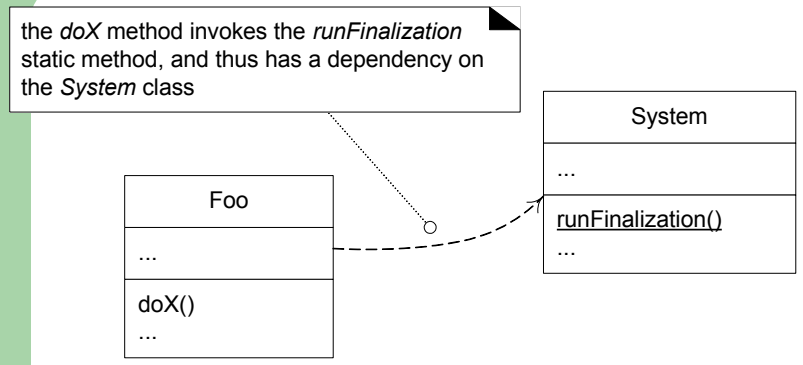
Dependency

- Another 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

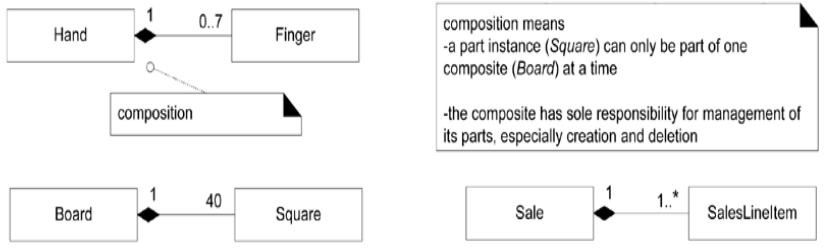
Dependency



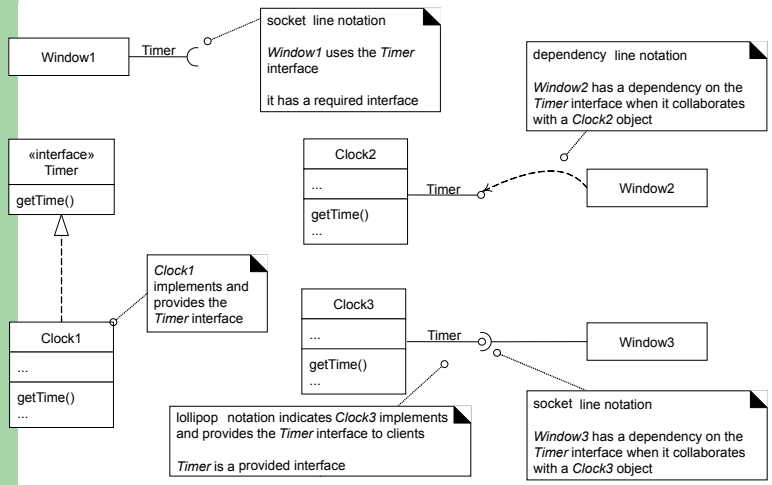
Composition

- 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
  - an instance of the part (such as a Square) belongs to only one composite instance (such as one Board) at a time,
  - the part must always belong to a composite (no free-floating Fingers), and
  - 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



Interface Implementation in UML



Summary: Relationship Between Interaction and Class Diagrams

- From interaction diagrams the definitions of class diagrams can be generated.
- This suggests a linear ordering of drawing interaction diagrams before class diagrams,
- In practice, especially when following the agile modeling practice of models in parallel, these complementary dynamic and static views are drawn concurrently.