Chapter 16
UML Class Diagrams

Larman, C. “Applying UML and Patterns”. 3rd Ed.
Fig. 16.1

- classifier name
- attributes
- operations

3 common compartments

- classOrStaticAttribute : Int
  + publicAttribute : String
  - privateAttribute
  assumedPrivateAttribute
  isInitializedAttribute : Bool = true
  aCollection : VeggieBurger [*]
  attributeMayLegallyBeNull : String [0..1]
  finalConstantAttribute : Int = 5 { readOnly }
  /derivedAttribute

+ classOrStaticMethod()
+ publicMethod()
  assumedPublicMethod()
  - privateMethod()
  # protectedMethod()
  ~ packageVisibleMethod()
  «constructor» SuperclassFoo( Long )
  methodWithParms(parm1 : String, parm2 : Float)
  methodReturnsSomething() : VeggieBurger
  methodThrowsException() { exception IOException }
  abstractMethod()
  abstractMethod2() ( abstract ) // alternate
  finalMethod() ( leaf ) // no override in subclass
  synchronizedMethod() { guarded }

- 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"

officially in UML, the top format is used to distinguish the package name from the class name

unofficially, the second alternative is common

- java.awt::Font
  or
  java.awt.Font

plain : Int = 0 { readOnly }
bold : Int = 1 { readOnly }
name : String { readOnly }
style : Int = 0...

gsFont(name : String) : Font
getName() : String...

interface implementation and subclassing

SubclassFoo

- run()

interface Runnable

Runnable

run()

interface Fruit

Fruit

dependency

PurchaseOrder

PurchaseOrder

association with multiplicities

1

order

...
Fig. 16.2

- **Domain Model**
  - **conceptual perspective**
  - **Register**
    - ... 
    - endSale()
    - enterItem(...)
    - makePayment(...)
  - Captures
    - 1
  - **Sale**
    - time
    - isComplete : Boolean
    - /total

- **Design Model**
  - **DCD; software perspective**
  - **Register**
    - ...
  - **Sale**
    - time
    - isComplete : Boolean
    - /total
    - makeLineItem(...)
    - currentSale

Captures 1 1
Fig. 16.3

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

Applying the guideline to show attributes as attribute text versus as association lines.
Fig. 16.6

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

Two ways to show a collection attribute
Fig. 16.7

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

<table>
<thead>
<tr>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
</tr>
<tr>
<td>endSale()</td>
</tr>
<tr>
<td>enterItem(id, qty)</td>
</tr>
<tr>
<td>makeNewSale()</td>
</tr>
<tr>
<td>makePayment(cashTendered)</td>
</tr>
</tbody>
</table>
using the stereotype

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

UML extension
relationship to a basic UML metamodel term – 
Element

«metaclass»
Element

using the stereotype

«stereotype»
Authorship

author: String
status : String

declare the stereotype

«authorship»

Square

«authorship»

author = “craig”
status = “tested”
the Sale has parameter visibility to a ProductDescription, and thus some kind of dependency
the `doX` method invokes the `runFinalization` static method, and thus has a dependency on the `System` class
Window

«call»

Clock

gTime()

... 

a dependency on calling on operations of the operations of a Clock

A

«create»

B

... 

a dependency that A objects create B objects
Window1 uses the Timer interface. It has a **required interface**.

Clock1 implements and provides the Timer interface.

Clock2

`getTime()`

... `getTime()` ...

Window2 has a dependency on the Timer interface when it collaborates with a Clock2 object.

Window3 has a dependency on the Timer interface when it collaborates with a Clock3 object.

Window1 uses the Timer interface. It has a **provided interface**.

**lollipop** notation indicates Clock3 implements and provides the Timer interface to clients.
Fig. 16.13

- 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.

composition means
Fig. 16.14

Stack

size : Integer \{ size >= 0 \}

push( element ) \quad \text{post condition: new size = old size + 1}

pop() : Object

\{ post condition: new size = old size – 1 \}

three ways to show UML constraints
Fig. 16.15

(a) Product Catalog
     Contains 1
     1..* Product Description

(b) Product Catalog
     Contains 1
     itemID
     qualifier
     multiplicity reduced to 1
     Product Description
a person may have employment with several companies

<table>
<thead>
<tr>
<th>Employment</th>
</tr>
</thead>
<tbody>
<tr>
<td>salary</td>
</tr>
<tr>
<td>startDate</td>
</tr>
</tbody>
</table>
ServicesFactory

instance : ServicesFactory

accountingAdapter : IAccountingAdapter
inventoryAdapter : IInventoryAdapter
taxCalculatorAdapter : ITaxCalculatorAdapter

getInstance() : ServicesFactory

getAccountingAdapter() : IAccountingAdapter
getInventoryAdapter() : IInventoryAdapter
getTaxCalculatorAdapter() : ITaxCalculatorAdapter

...
K is a **template parameter**

parameterized or template interfaces and classes

anonymous class with **template binding** complete

ArrayList\(<T\> -> Square\>)

  clear()
  ...

for example, the *elements* attribute is an array of type T, parameterized and bound before actual use.

the attribute type may be expressed in official UML, with the template binding syntax requiring an arrow

or

in another language, such as Java

there is a chance the UML 2 "arrow" symbol will eventually be replaced with something else e.g., '='

the attribute type may be expressed in official UML, with the template binding syntax requiring an arrow

or

in another language, such as Java
### Fig. 16.19

<table>
<thead>
<tr>
<th><strong>DataAccessObject</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>id : Int</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>doX()</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

**exceptions thrown**
- DatabaseException
- IOException

**responsibilities**
- serialize and write objects
- read and deserialize objects
...
Fig. 16.21

messages in interaction diagrams indicate operations in the class diagrams

classes identified in the interaction diagrams are declared in the class diagrams

Register

...  
makePayment(…)
...  

Sale

...  
makePayment(…)
...  

1

currentSale

makePayment(cashTendered)