Object-Oriented Programming
Introduction to
UML Class Diagrams

CSIE Department, NTUT

Woei-Kae Chen
UML: Unified Modeling Language

- Successor to OOA&D methods
  - late 1980s and early 1990s
- Unifies
  - Jacobson & OMT (Booch & Rumbaugh)
- Graphical notation used to express designs
  - Use cases
  - Class diagrams
  - Interaction diagrams
    - Sequence diagrams
    - Collaboration diagrams
  - Package diagrams
  - State diagrams
  - Activity diagrams
  - Deployment diagrams

GoF Book
UML class diagrams

• Three perspectives
  – Conceptual
    • represents of the domain under study
    • relate to the class that implement them, but often no direct mapping
  – Specification
    • looking at types rather than classes
    • a type represents an interface that may have different implementations
  – Implementation
    • looking at classes

for our OOP class
UML: a class

Class Name

+ public
# protected
- private

Abstract
Concrete

• data type
• parameter

| variable1 |
| variable2 |

| function1() |
| function2() |
Example: OBSort1.cpp

<table>
<thead>
<tr>
<th>IntArray</th>
</tr>
</thead>
<tbody>
<tr>
<td>-a</td>
</tr>
<tr>
<td>-size</td>
</tr>
<tr>
<td>+getInput()</td>
</tr>
<tr>
<td>+printOutput()</td>
</tr>
<tr>
<td>+Sort()</td>
</tr>
<tr>
<td>+cleanUp()</td>
</tr>
<tr>
<td>+getSize()</td>
</tr>
</tbody>
</table>
Example: OBSort1.cpp

Let’s assume `main()` is a class

```
IntArray
- a
- size
+ getInput()
+ printOutput()
+ Sort()
+ cleanUp()
+ getSize()
```

Context (main)

relationship
UML: class relationship

- **Association**
  \[ \text{X} \rightarrow \text{Y} \] (knows a)

- **Dependency**
  \[ \text{X} \rightarrow \text{Y} \] (uses a)

- **Composition**
  \[ \text{X} \rightarrow \text{Y} \] (has a)

- **Aggregation**
  \[ \text{X} \rightarrow \text{Y} \] (has a)

- **Inheritance**
  \[ \text{Y} \rightarrow \text{X} \] (is a)

- **Class template**
  \[ \text{X} \rightarrow \text{Y} \] (parameterized class)
“Uses a” ⇔ “Knows a” relationship

● “Uses a”
  – Dependency
  – One object issues a function call to a member function of another object

● “Knows a”
  – Association
  – One object is aware of another; it contains a pointer or reference to another object
"Is a" $\Leftrightarrow$ "Has a" relationship

- **"Is a" relationships**
  - Inheritance
  - a class is derived from another class

- **"Has a" relationships**
  - Composition or Aggregation
  - a class contains other classes as members
Both are “Has a” or “part-of” relationship

Composition
- a stronger variety of aggregation
- the part object may belong to only one whole
- expected to live and die with the whole
  - delete whole → delete part

Aggregation
- cascading delete is often
- an aggregated instance can be shared
Example: “has a” relationship

- A Point may appear in only one Polygon or Circle.
- A Style may be shared by many Polygons and Circles.

Multiplicity:
- A Point may appear in 1 Polygon or Circle.
- A Style may be a part of 1 Circle.

{ordered}:
- A Point is part of a Polygon in an ordered manner.

Delete:
- Delete Polygon ⇒ delete Point
- Delete Polygon × delete Style
class X {
    X(Y &y) : y_ref(y) {}  
    void SetY(Y *y) {y_ptr = y;}  
    void f() {y_ptr->doIt();}
    ... 
    Y *y_ptr; // pointer  
    Y &y_ref; // reference
};
class X {
  ...
  void f1(Y y) { ... ; y.doIt(); }
  void f2(Y *y_ptr);
  void f3(Y &y_ref);
};
Example: OBSort3.cpp

Sorter
+sort()

IntArray
- a
- size
+ getInput()
+ printOutput()
+ cleanUp()
+ getSize()
+ operator[]()
UML Example (C++): Composition 1

class X {
    ...
    Y a;       // 1; Composition
    Y b[10];  // 0..10; Composition
    vector<Y> c; // ??
};

Java?

Composition of vector<Y>

NOT Composition of Y
UML Example (C++): Composition 2

```cpp
class X {
    X() { a = new Y[10]; }
    ~X(){ delete [] a; }
    ...
    Y *a;       // 0..10; Composition
};
```

NOT Association
UML Example: OBSort3.cpp

```
+getInput()
+printOutput()
+cleanUp()
+getSize()
+operator[](a)
```

IntArray

```
-a
-size
+getInput()
+printOutput()
+cleanUp()
+getSize()
+operator[](a)
```
class X {
    X() { a = new Y[10]; }
    ~X(){ delete [] a; }
... 
    Y *a;       // 0..n; Aggregation
    vector<Y> b; // Y’s are instantiated
                  // and destroyed by X
};

The same as composition?

May be considered as aggregation of Y
class X {
    ...
    vector<Y> b;
};
class Y {
... 
};

class X : public Y {
... 
};

“is a” relationship
Example: OOSort2.cpp

```
+getInput()
+printOutput()
+cleanUp()
+getSize()
+operator[]()
```

```
Sorter
+sort()

Context (main)

IntArray
-a
-size
+getInput()
+printOutput()
+cleanUp()
+getSize()
+operator[]()

CountingSorter

-a
-compareCount
-exchangeCount
+sort()
```
UML Example (C++): Template Class

template <class T>
class X {

...  

...  

...  

};

X<Y> a;

...
**Order**

- `dateReceived` : String
- `isPrepaid` : Boolean
- `number` : String
- `price` : Money

**Operations**

- `dispatch()`
- `close()`

**Association**

**Customer**

- `name`
- `address`
- `creditRating()` : String

**Generalization**

**Corporate Customer**

- `contactName`
- `creditRating`
- `creditLimit`

**Operations**

- `remind()`
- `billForMonth(Integer)`

**Personal Customer**

- `creditCard#`

**Order Line**

- `quantity` : Integer
- `price` : Money
- `isSatisfied` : Boolean

**Operations**

- `*`

**Employee**

- `sales rep` : 0..1

**Product**

- `1`

**Constraints**

- `if Order.customer.creditRating is "poor," then Order.isPrepaid must be true`
Order
- dateReceived
- isPrepaid
- number: String
- price: Money

Order Line
- quantity: Integer
- price: Money
- isSatisfied: Boolean

Customer
- name
- address
- creditRating(): String

Corporate Customer
- contactName
- creditRating
- creditLimit

Personal Customer
- creditCard#

{creditRating() == "poor"}

Employee

Product

Navigability

{if Order.customer.creditRating is "poor," then Order.isPrepaid must be true}
C++ static member