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
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

+ public
# protected
- private

Abstract
Concrete

• data type
• parameter

class_name

<table>
<thead>
<tr>
<th>variable1</th>
</tr>
</thead>
<tbody>
<tr>
<td>variable2</td>
</tr>
<tr>
<td>function1()</td>
</tr>
<tr>
<td>function2()</td>
</tr>
</tbody>
</table>
Example: OBSort1.cpp

IntArray

- a
- size
+ getInput()
+ printOutput()
+ Sort()
+ cleanUp()
+ getSize()
Example: OBSort1.cpp

Let’s assume main() is a class

**IntArray**
- $a$
- $size$

+ getInput()
+ printOutput()
+ sort()
+ cleanUp()
+ getSize()
UML: class relationship

- Association (knows a)
- Dependency (uses a)
- Composition (has a)
- Aggregation (has a)
- Inheritance (is a)
- Class template (parameterized class)
“Uses a” $\Leftrightarrow$ “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” ⇔ “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
Aggregation $\iff$ Composition

- 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 $\Rightarrow$ 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 appears in only one Polygon or Circle.
- A Style can be associated with many Polygons and Circles.

Relations:
- 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

-\( a \)
-\( \text{size} \)

+\( \text{getInput()} \)
+\( \text{printOutput()} \)
+\( \text{cleanUp()} \)
+\( \text{getSize()} \)
+\( \text{operator[]()} \)

uses
-\( \text{getSize()} \)
-\( \text{operator[]()} \)
class X {
    ...
    Y a;        // 1; Composition
    Y b[10];    // 0..10; Composition
    vector<Y> c; // ??
};

Composition of vector<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

Context (main) -> Sorter

Sorter:
+ sort()

IntArray

IntArray:
- a
- size
+ getInput()
+ printOutput()
+ cleanUp()
+ getSize()
+ operator[](0)
UML Example (C++): Aggregation 1

```cpp
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
UML Example (C++): Aggregation 2

```cpp
class X {
    ...
    vector<Y> b;
};
```

Implementation detail

Hiding implementation detail
UML Example (C++): Inheritance

```cpp
class Y {
    ...
};

class X : public Y {
    ...
};
```

“is a” relationship
Example: OOSort2.cpp

```
Context (main)

Sorter
+sort()

CountingSorter
-compareCount
-exchangeCount
+sort()

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

![Class Diagram](image-url)
UML Example (C++): Template Class

template <class T>
class X {
  ...
  ...
  ...
};
**Order**

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

- dispatch()
- close()

**Customer**

- name
- address
- creditRating() : String

**Navigability**

1

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

**Corporate Customer**

- contactName
- creditRating
- creditLimit

- remind()
- billForMonth(Integer)

**Personal Customer**

- creditCard#

{creditRating() == "poor"}

**Order Line**

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

**Employee**

**Product**

1

0..1
C++ static member