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

- public
- protected
- private

Abstract
Concrete

- data type
- parameter

class_name

variable1
variable2

function1()
function2()
Example: OBSort1.cpp

**IntArray**

-a
-size

+getInput()
+printOutput()
+Sort()
+cleanUp()
+getSize()
Let's assume `main()` is a class.
UML: class relationship

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

Multiplicities:
- Polygon has 1 Point and 3..* Circles.
- Circle has 1 Style.
- Style has 1 isFilled and 1 color.

Deleted relationships:
- 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()
class X {
...  
    Y a;       // 1; Composition
    Y b[10];   // 0..10; Composition
    vector<Y> c; // ??
};
UML Example (C++): Composition 2

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

NOT Association
UML Example: OBSort3.cpp

Context(main)

Sorter
+sort()

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

```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
};
```

May be considered as aggregation of Y

The same as composition?
class X {
    ...
    vector<Y> b;
};
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[]( )
```
UML Example (C++): Template Class

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

X<Y> a;
...

...
Order

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

- dispatch()
- close()

Multiplicity: mandatory

Customer

- name
- address
- creditRating() : String

Association

Generalization

Constraint

{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

- line items

Employee

- sales rep : 0..1

Product

- 1
Order

dateReceived
isPrepaid
number : String
price : Money

dispatch()
close()

Customer

name
address
creditRating():String

Corporate Customer

contactName
creditRating
creditLimit

remind()
billForMonth(Integer)

Personal Customer

creditCard#

{creditRating() == "poor"}

Order Line

quantity : Integer
price : Money
isSatisfied : Boolean

line items

Employee

sales rep
0..1

Product

1

Navigability

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

Text Editor

Dependency

Window

{abstract}

toFront()
toBack()

Windows Window

toFront()
toBack()

X11 Window

toFront()
toBack()

Mac Window

toFront()
toBack()
C++ static member