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)

GoF Book

- 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

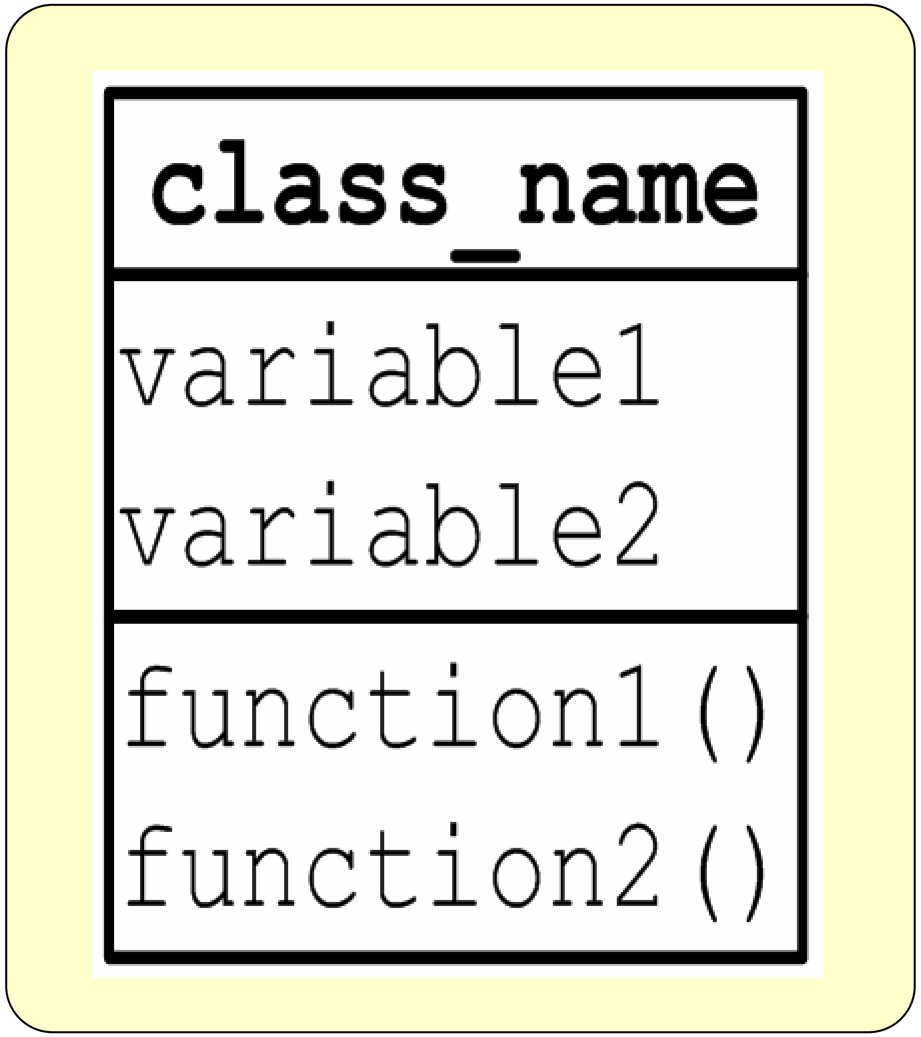
for our OOP class

UML: a class

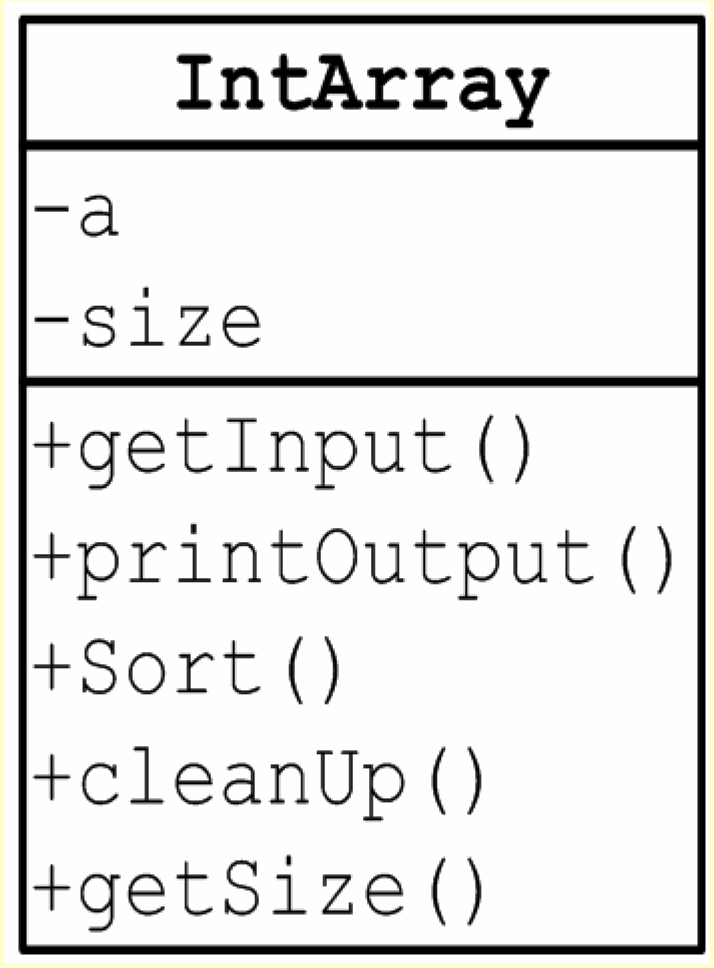
```
+    public
#    protected
-    private
```

Abstract
Concrete

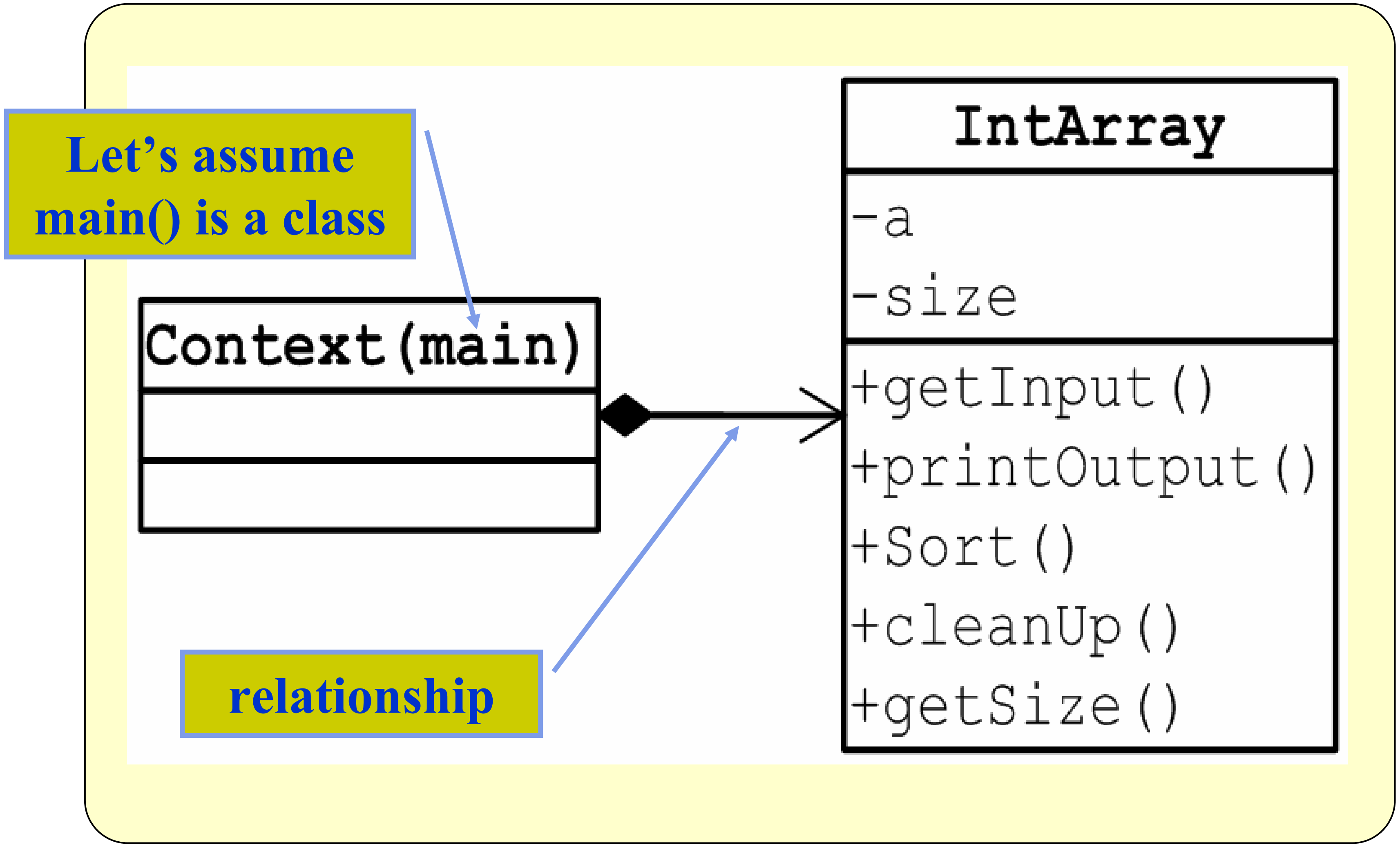
- data type
- parameter



Example: OBSort1.cpp

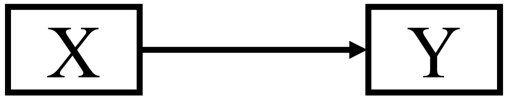


Example: OBSort1.cpp



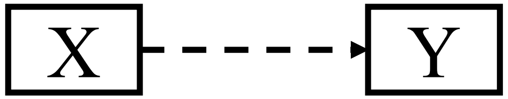
UML: class relationship

- Association



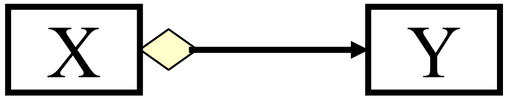
(knows a)

- Dependency



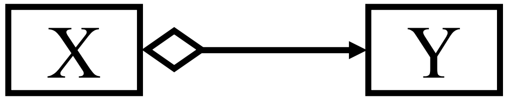
(uses a)

- Composition



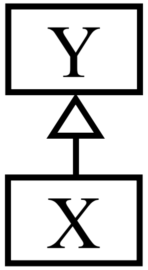
(has a)

- Aggregation



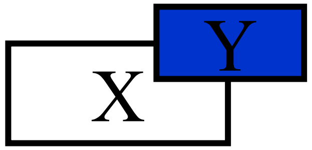
(has a)

- Inheritance



(is a)

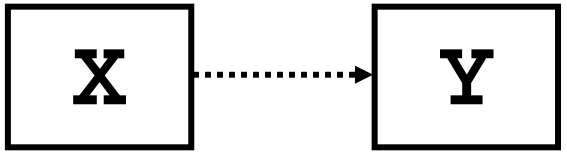
- Class template



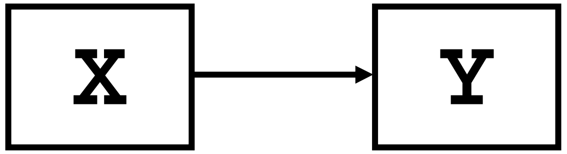
(parameterized class)

“Uses a” ⇔ “Knows a” relationship

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

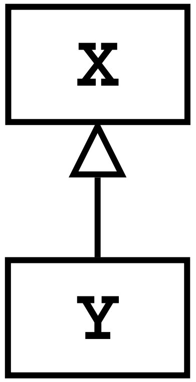


- “Knows a”
  - Association
  - One object is aware of another; it contains a pointer or reference to another object

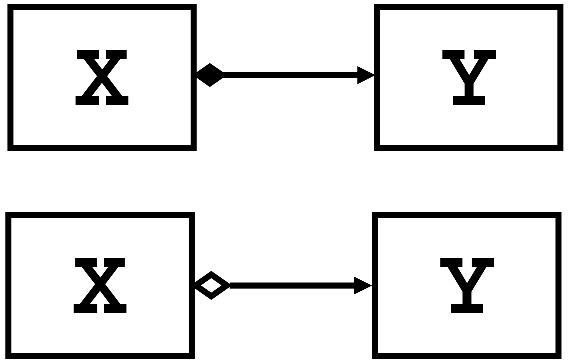


"Is a" ⇔ "Has a" relationship

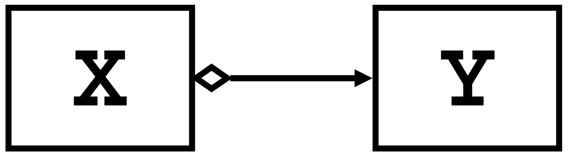
- "Is a" relationships
  - Inheritance
  - a class is derived from another class

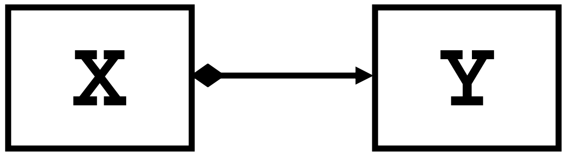


- "Has a" relationships
  - Composition or Aggregation
  - a class contains other classes as members



Aggregation ⇔ Composition



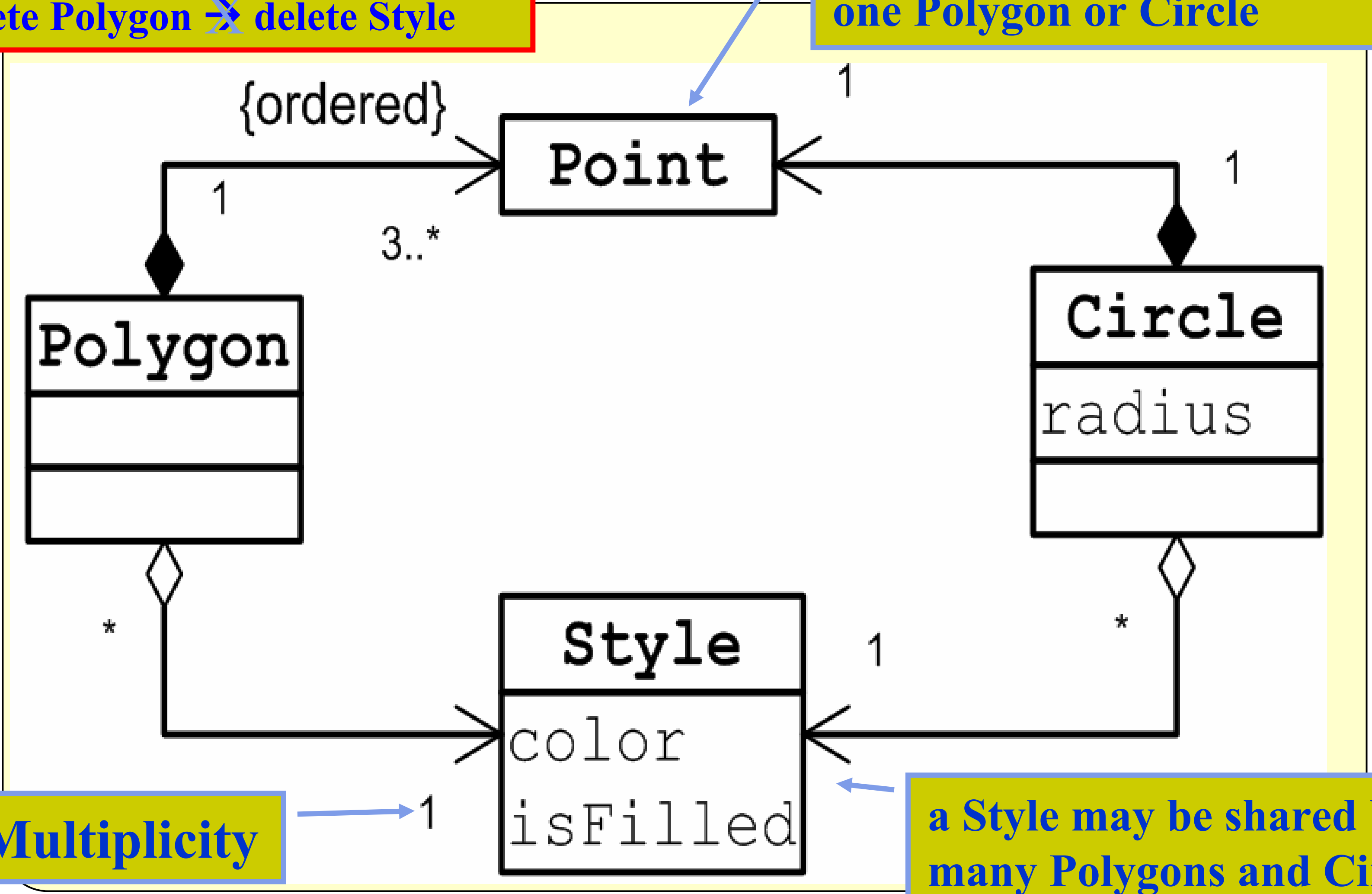


- Both are “Has a” or “part-of” relationship
- Composition
  - a stronger variety of aggregation
  - the part object may belong to only one whole
  - expected to live and die with the whole
    - delete whole → delete part
- Aggregation
  - cascading delete is often
  - an aggregated instance can be shared

Example: "has a" relationship

Delete Polygon → delete Point
Delete Polygon ↛ delete Style

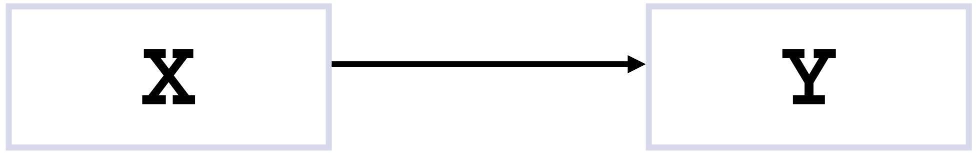
a Point may appear in only one Polygon or Circle



Multiplicity

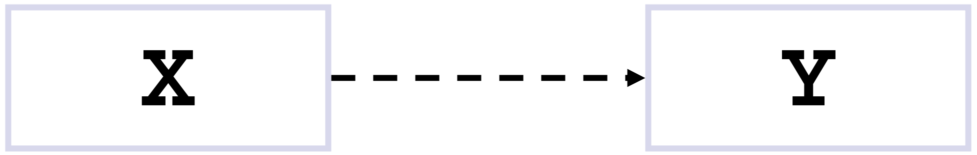
a Style may be shared by many Polygons and Circles

UML Example (C++): Association



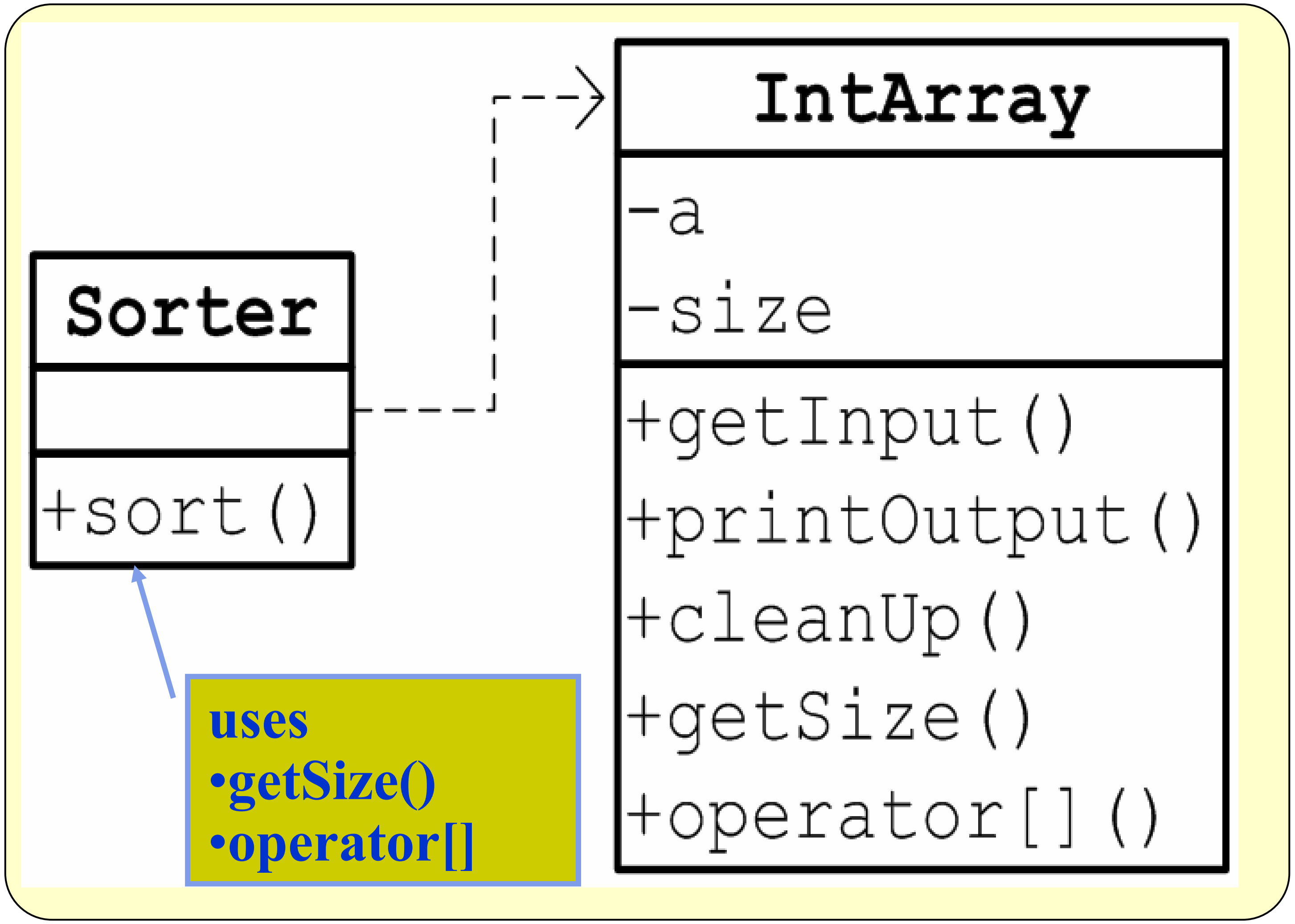
```
class X {
   X(Y &y) : y_ref(y) {}
   void SetY(Y *y) {y_ptr = y;}
   void f() {y_ptr->doIt();}
...
   Y *y_ptr; // pointer
   Y &y_ref; // reference
};
```

UML Example (C++): Dependency

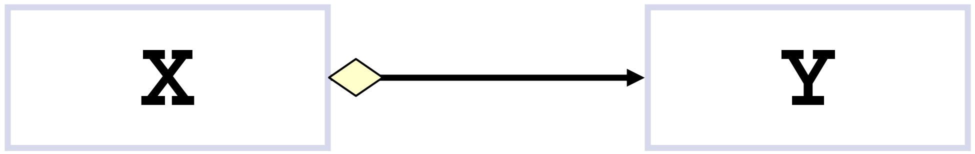


```
class X {
...
    void f1(Y y) {...; y.doIt();}
    void f2(Y *y_ptr);
    void f3(Y &y_ref);
};
```

Example: OBSort3.cpp



UML Example (C++): Composition 1



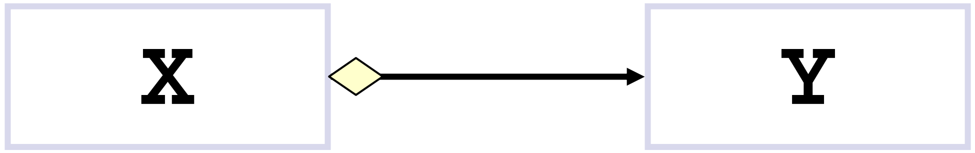
```
class X {
...
   Y a;            // 1; Composition
   Y b[10];        // 0..10; Composition
   vector<Y> c;    // ??
};
```

Java?

Composition of vector<Y>

NOT Composition of Y

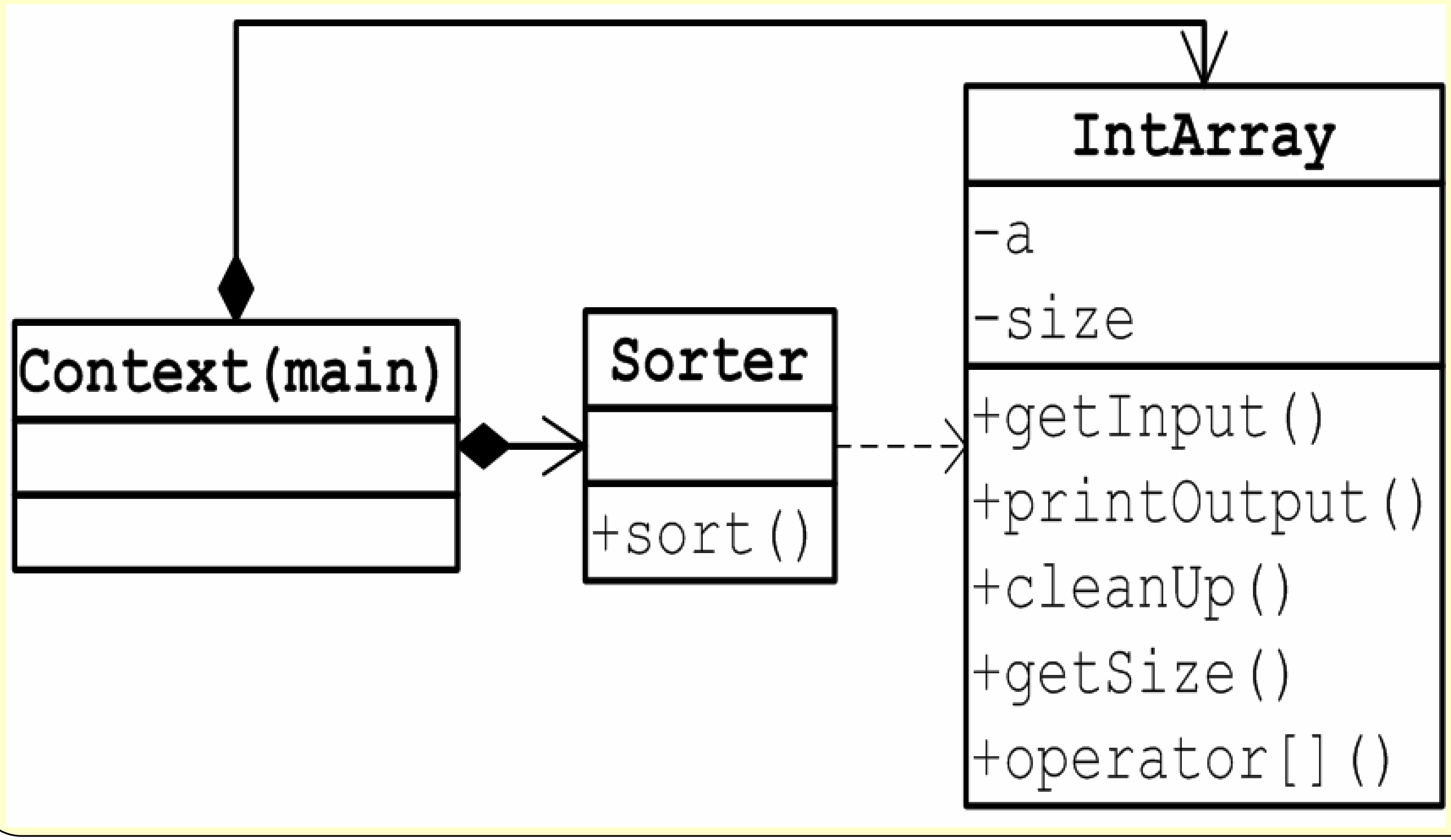
UML Example (C++): Composition 2



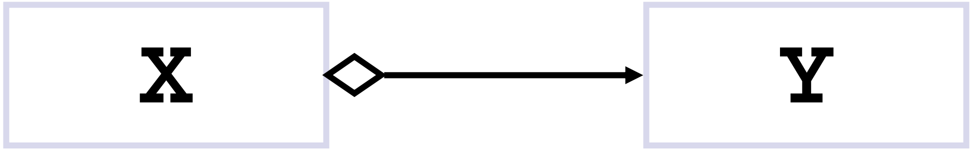
```
class X {
   X() { a = new Y[10]; }
   ~X(){ delete [] a; }
...
   Y *a;          // 0..10; Composition
};
```

NOT Association

UML Example: OBSort3.cpp



UML Example (C++): Aggregation 1

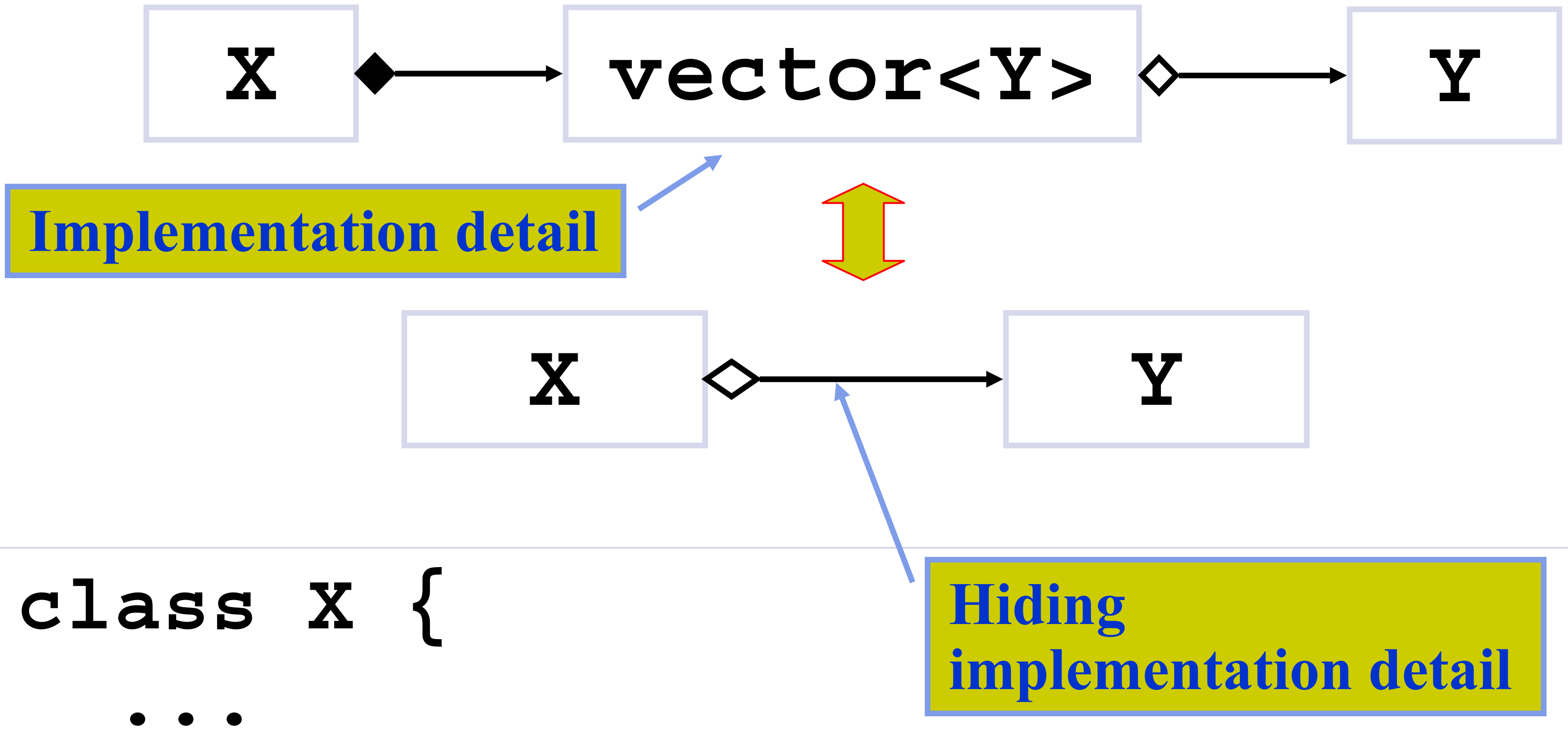


```
class X {
   X() { a = new Y[10]; }
   ~X(){ delete [] a; }
...
   Y *a;          // 0..n; Aggregation
   vector<Y> b;// Y's are instantiated
               // and destroyed by X
};
```

The same as composition?

May be considered as aggregation of Y

UML Example (C++): Aggregation 2



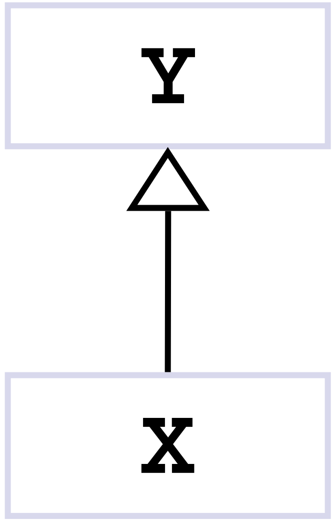
```
class X {
   ...
   vector<Y> b;
};
```

UML Example (C++): Inheritance

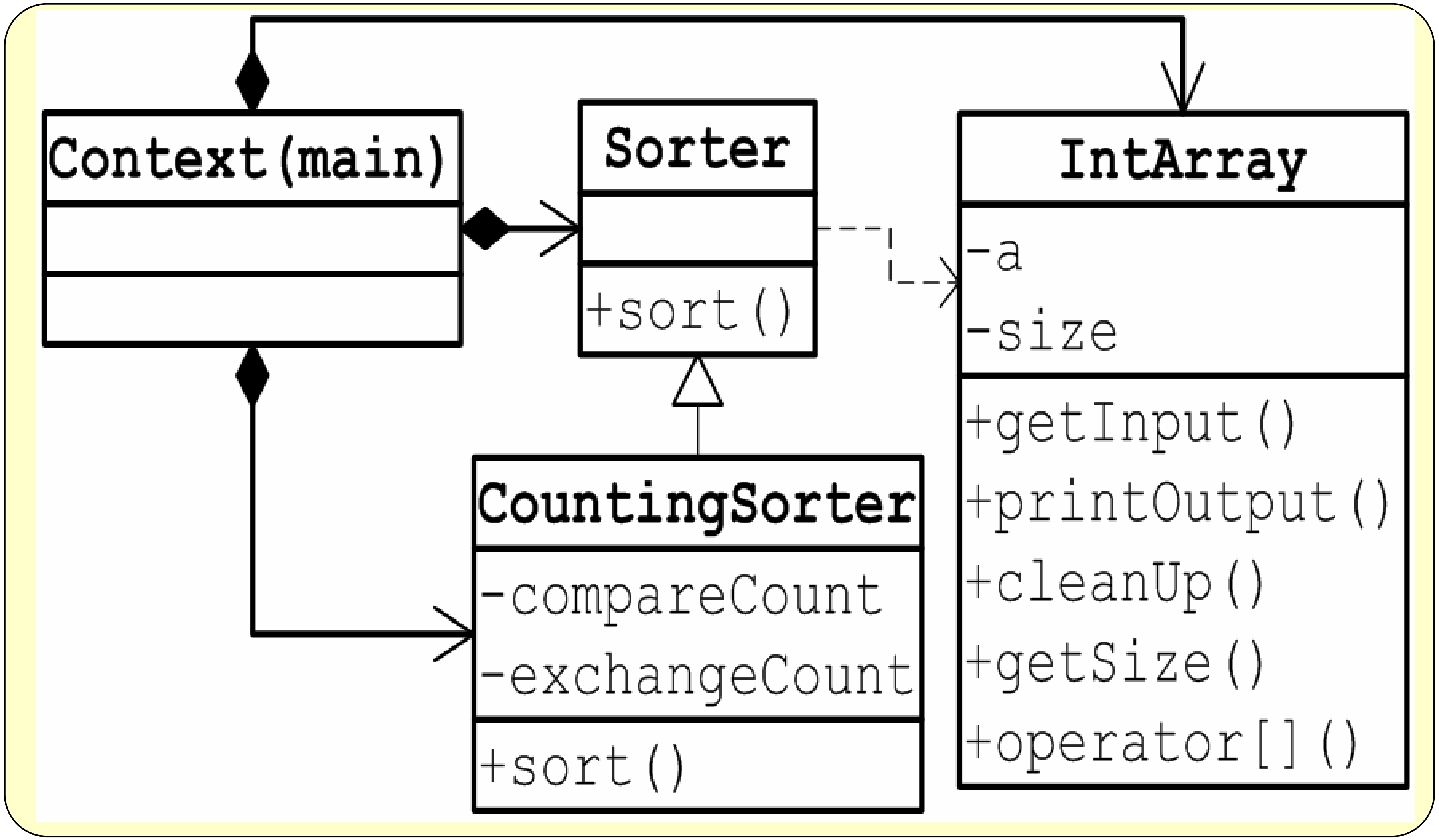
```
class Y {
...
};

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

“is a” relationship

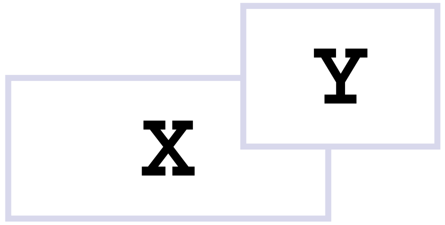


Example: OOSort2.cpp

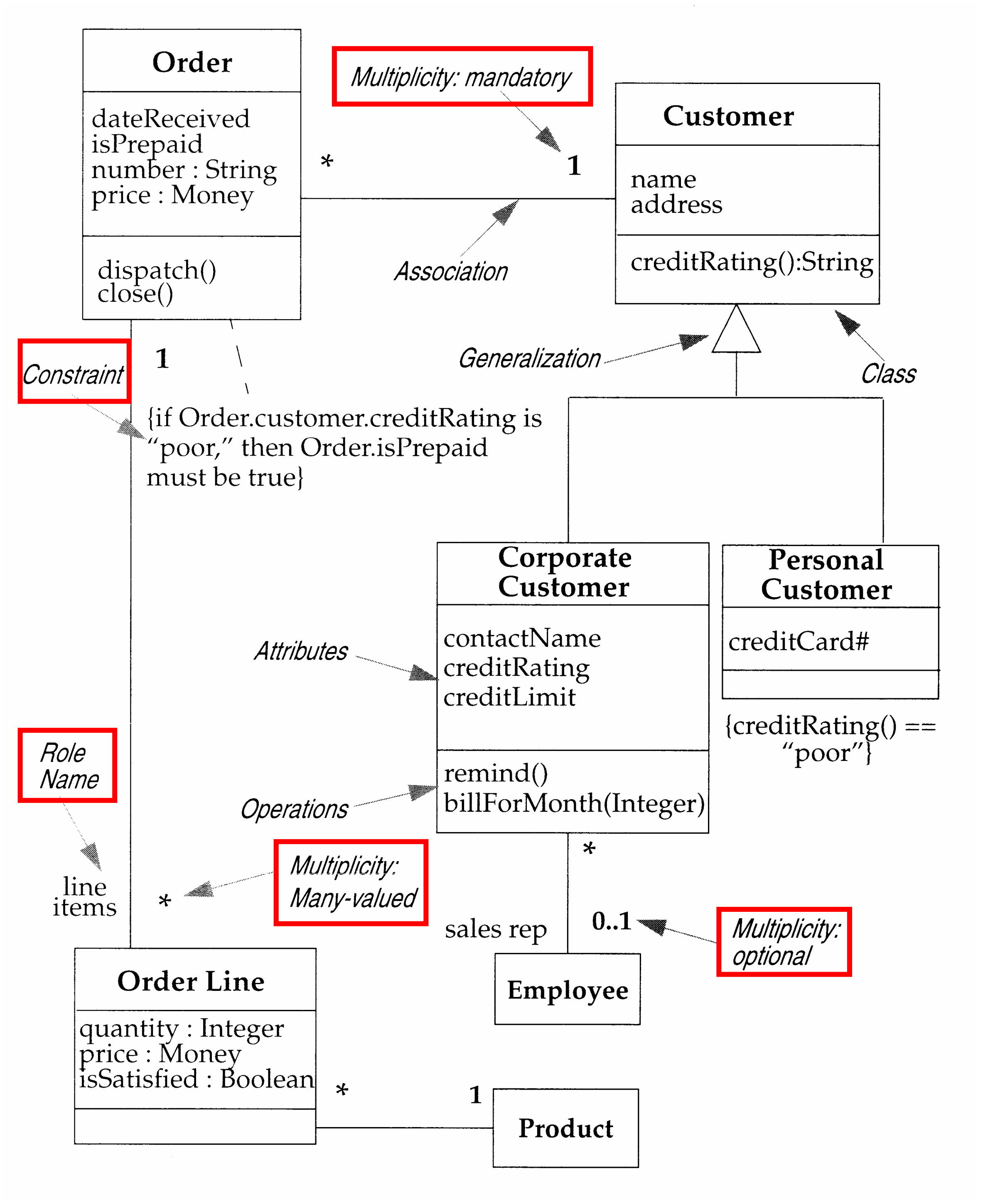


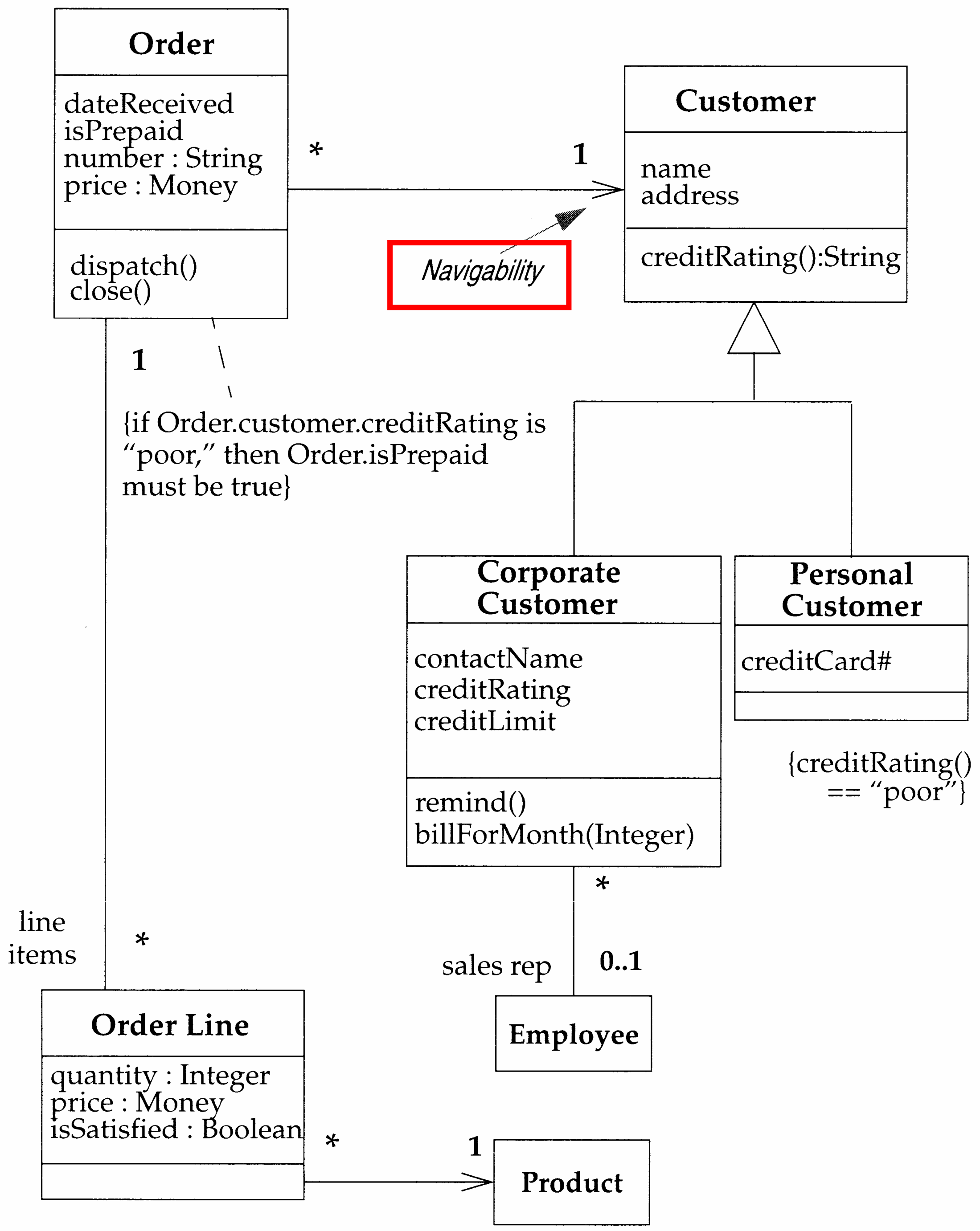
UML Example (C++): Template Class

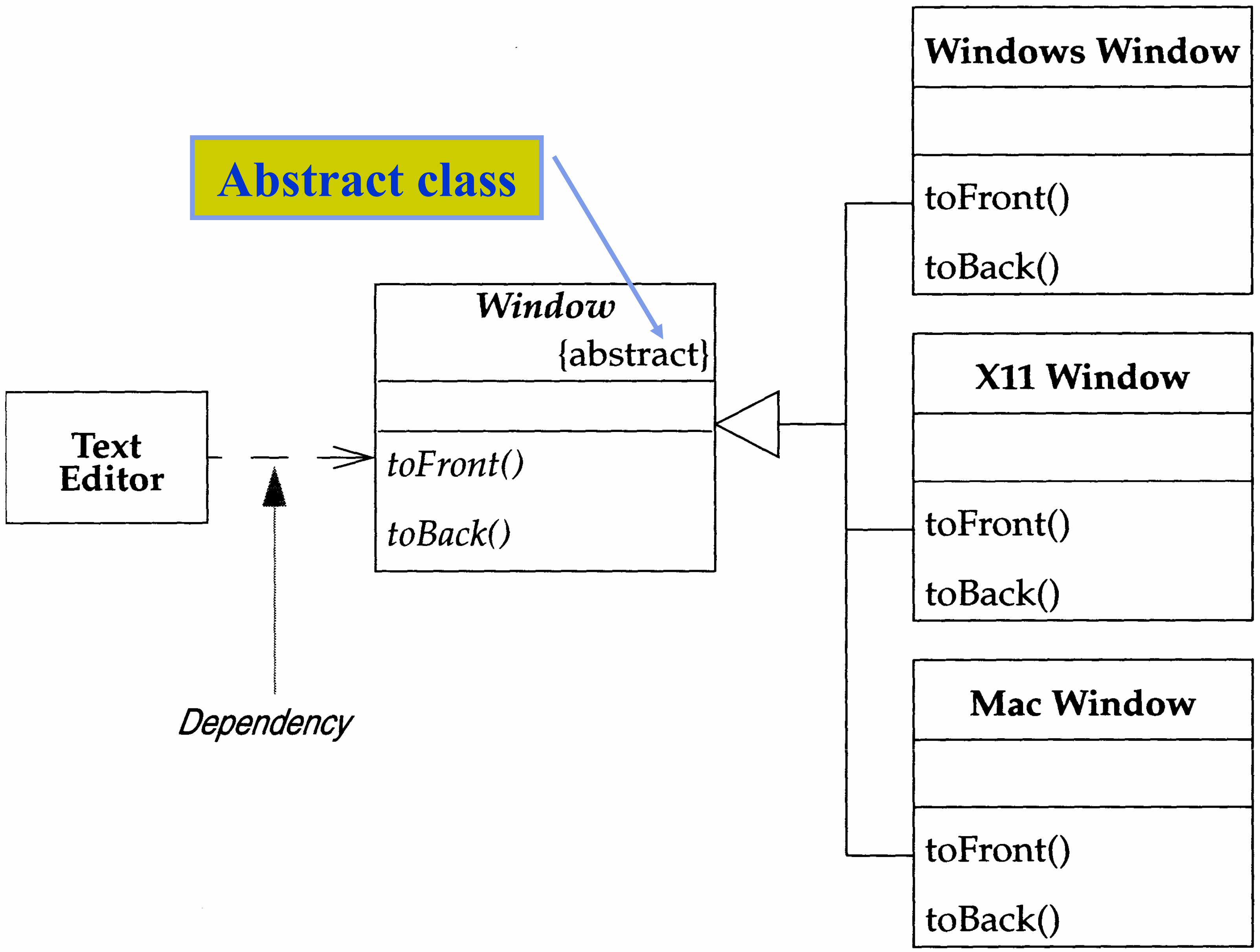
```
template <class T>
class X {
...
...
...
};
```

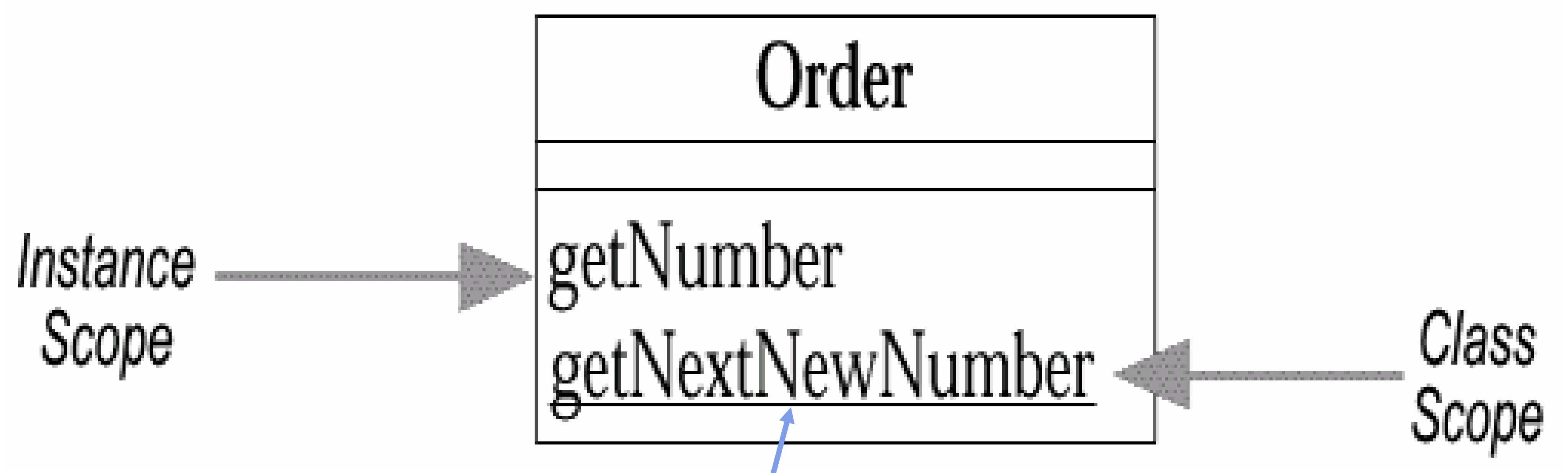


```
...
X<Y> a;
...
```


Order
dateReceived
isPrepaid
number : String
price : Money
dispatch()
close()
Multiplicity: mandatory
*
1
Customer
name
address
creditRating():String
Association
Constraint
1
{if Order.customer.creditRating is "poor," then Order.isPrepaid must be true}
Generalization
Class
Corporate Customer
contactName
creditRating
creditLimit
remind()
billForMonth(Integer)
Personal Customer
creditCard#
{creditRating() == "poor"}
Attributes
Operations
Role Name
line items
*
Multiplicity: Many-valued
*
sales rep
0..1
Multiplicity: optional
Employee
Order Line
quantity : Integer
price : Money
isSatisfied : Boolean
*
1
Product


Order
dateReceived
isPrepaid
number : String
price : Money
dispatch()
close()
*
1
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"}
line items
*
*
sales rep
0..1
Employee
Order Line
quantity : Integer
price : Money
isSatisfied : Boolean
*
1
Product


Abstract class
Window
{abstract}
toFront()
toBack()
Text Editor
Dependency
Windows Window
toFront()
toBack()
X11 Window
toFront()
toBack()
Mac Window
toFront()
toBack()



C++ static member