Lecture #12 | Polymorphism: templates

SE271 Object-oriented Programming (2017)

Prof. Min-gyu Cho

Previously in Object-Oriented Programming

- Polymorphism: virtual functions
  - Functions with the same name may behave differently, depending on which types (i.e., classes) they are associated with

Today’s topic

- Templates

Templates

- A *template* is a class or a function that we parameterize with a set of types or values
- We represent general ideas from which we can generate specific classes or functions by providing types (e.g., int, double, or user-defined class) as parameters
- Syntax: both `class` and `typename` is valid, but the latter is recommended

```
template<typename Type> function_declaration;
template<class Type> function_declaration;

template<typename Type> class_declaration;
template<class Type> class_declaration;
```

Example: function templates

```
template<typename T>
T add(const T x, const T y) {
    return x + y;
}

int main() {
    cout << add<int>(42, 23) << endl;
    cout << add<double>(3.14, 1.68) << endl;
    cout << add(3.14, 1.68) << endl;
}
```

- By using function templates, we do not have to implement the same functions for different types
- When using function template, you may omit template argument if the required type can be unambiguously deduced

Example: class templates

```cpp
template<typename T>
class Point {
    T x;
    T y;
public:
    Point(T xx=0, T yy=0) : x(xx), y(yy) {}
    T getX() { return x; }
    T getY() { return y; }
};
int main()
{
    Point<double> pt_d {1.2, 3.4};
    cout << pt_d.getX() << endl;
    Point<int> pt_i {1, 2};
    cout << pt_i.getX() << endl;
}
```

- By using class templates, we do not have to implement the same classes with different types

Example: class templates with more than one types

```
template<typename T1, typename T2>
class Pair {
public:
    T1 first;
    T2 second;
};

int main()
{
    Pair<int, double> pair {42, 3.14};
    cout << pair.first << " ”
         << pair.second << endl;
}
```

- You may have as many template arguments, i.e., template types

Example: class template with values

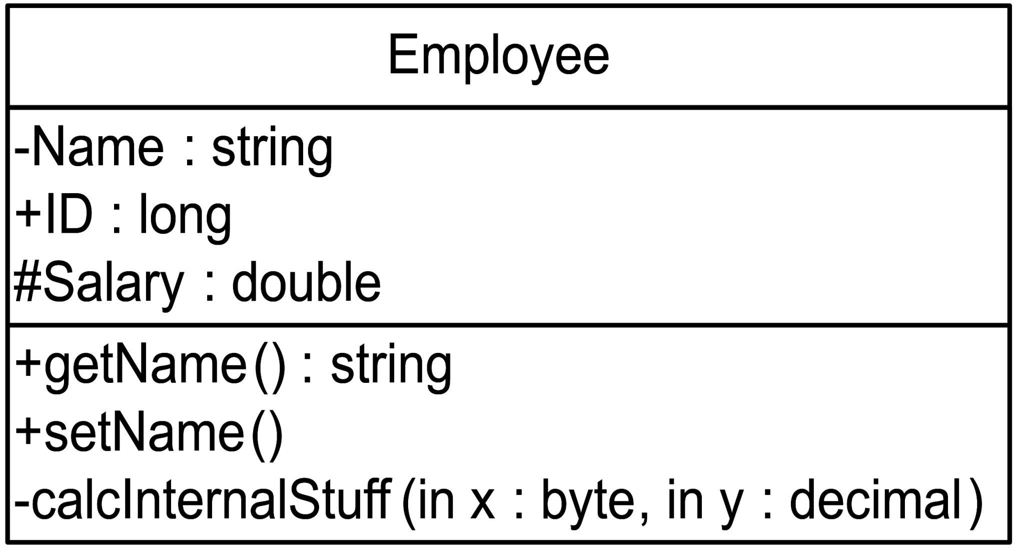
```cpp
template<typename T, int dim>
class PointND {
    T* coordinates;
public:
    PointND() { coordinates = new T[dim]; }
    ~PointND() { delete[] coordinates; }
    int getDimension() { return dim; }
};
int main() {
    PointND<double, 2> p2;
    PointND<double, 3> p3;

    cout << p2.getDimension() << endl;
    cout << p3.getDimension() << endl;
}
```

Is-a v.s. Has-a

- Two ways we can describe some class A depending on some other class B
  - Every A object has a B object. For instance, every Vehicle has a string object (called license or name)
  - Every instance of A is a B instance. For instance, every Car is a Vehicle, as well
- ***Inheritance*** allows us to define “is-a” relationship, but it should not be used to implement “has-a” relationships
- Sometime it is not clear whether to use “is-a” or “has-a” relationship

UML Class Notation

- A class is a rectangle divided into three parts
  - Class name
  - Class attributes (i.e., member variables)
  - Class operations (i.e., member functions)
- Modifiers
  - Private: -
  - Public: +
  - Protected: #
  - Static: underlined
- Abstract class: name in italics



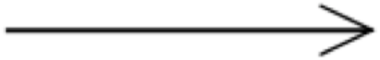
Employee

-Name : string
+ID : long
#Salary : double

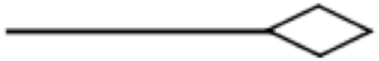
+getName() : string
+setName()
-calcInternalStuff (in x : byte, in y : decimal)

UML Class relations

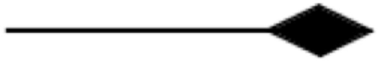
- **Association**: a straight line or arrow



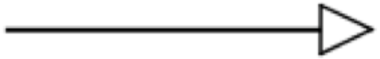
  - A relationship between instances of two classes, where one class must know about the other to do its work, e.g., client communicates to server
- **Aggregation**: an empty diamond on the side of the collection



  - An association where one class belongs to a collection, e.g., instructor part of faculty
- **Composition**: a solid diamond on the side of the collection



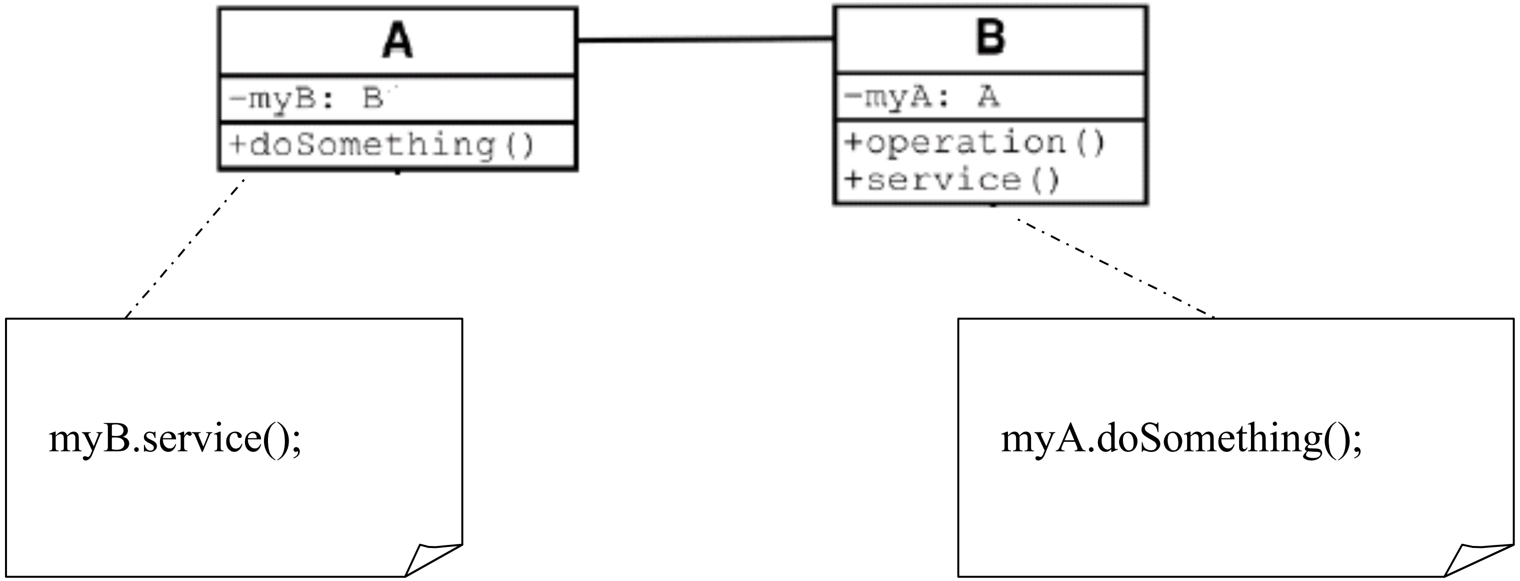
  - Strong form of aggregation
  - Lifetime control; components cannot exist without the aggregate
- **Inheritance**: a triangle pointing to superclass



  - An inheritance link indicating one class a superclass relationship, e.g., bird is part of mammal

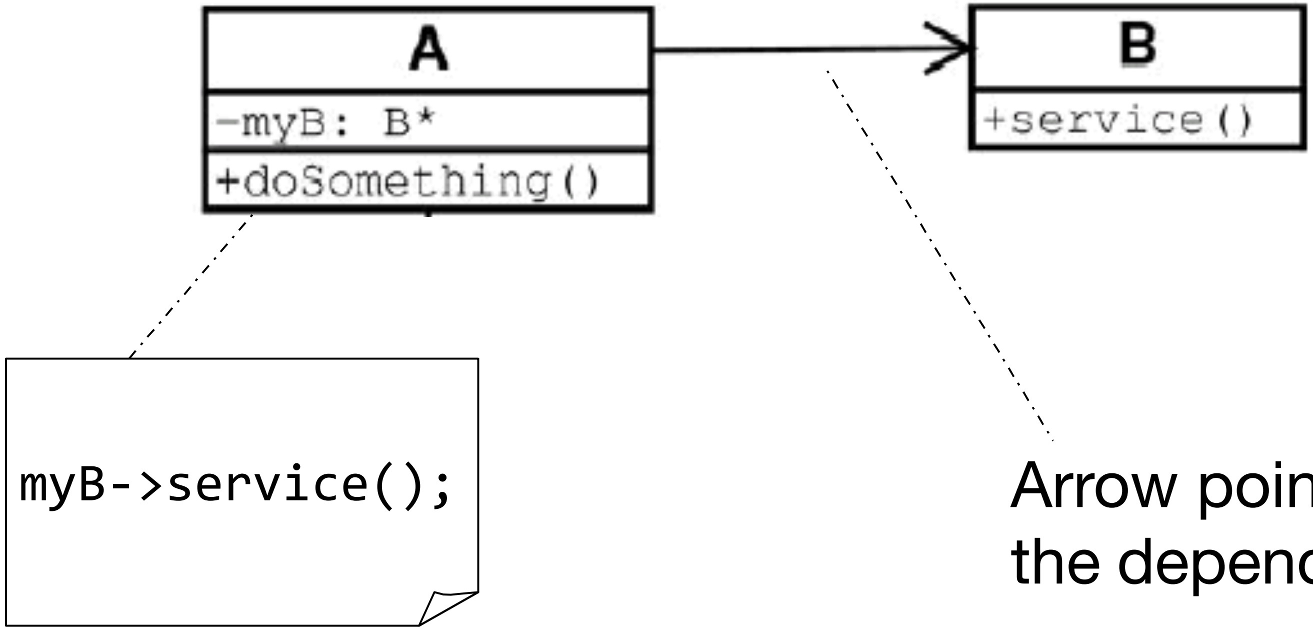
Binary Association

- Both entities “know about” each other



Unary Association

- A knows about B, but B knows nothing about A

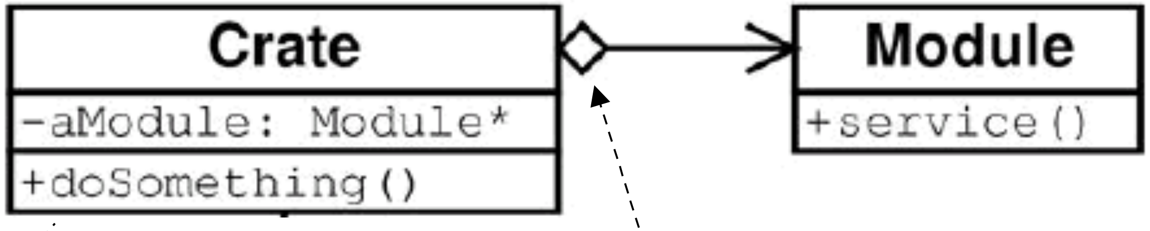


```
myB->service();
```

Arrow points in direction of the dependency

Aggregation

- Aggregation is an association with a “collection-member” relationship



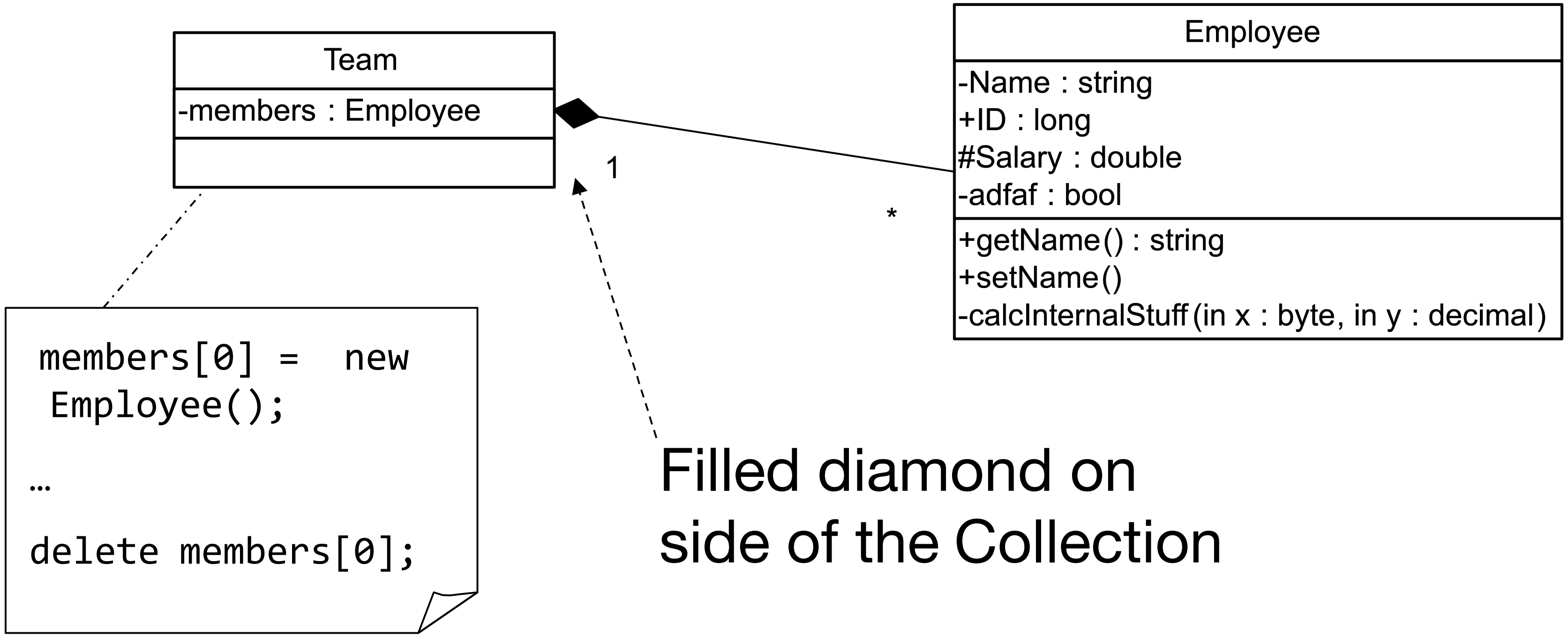
```
void doSomething() {
  aModule->service();
}
```

Hollow diamond on the Collection side

No sole ownership implied

Composition

- Composition is aggregation with
  - The whole-part relationship
  - Lifetime control (owner controls construction & destruction)

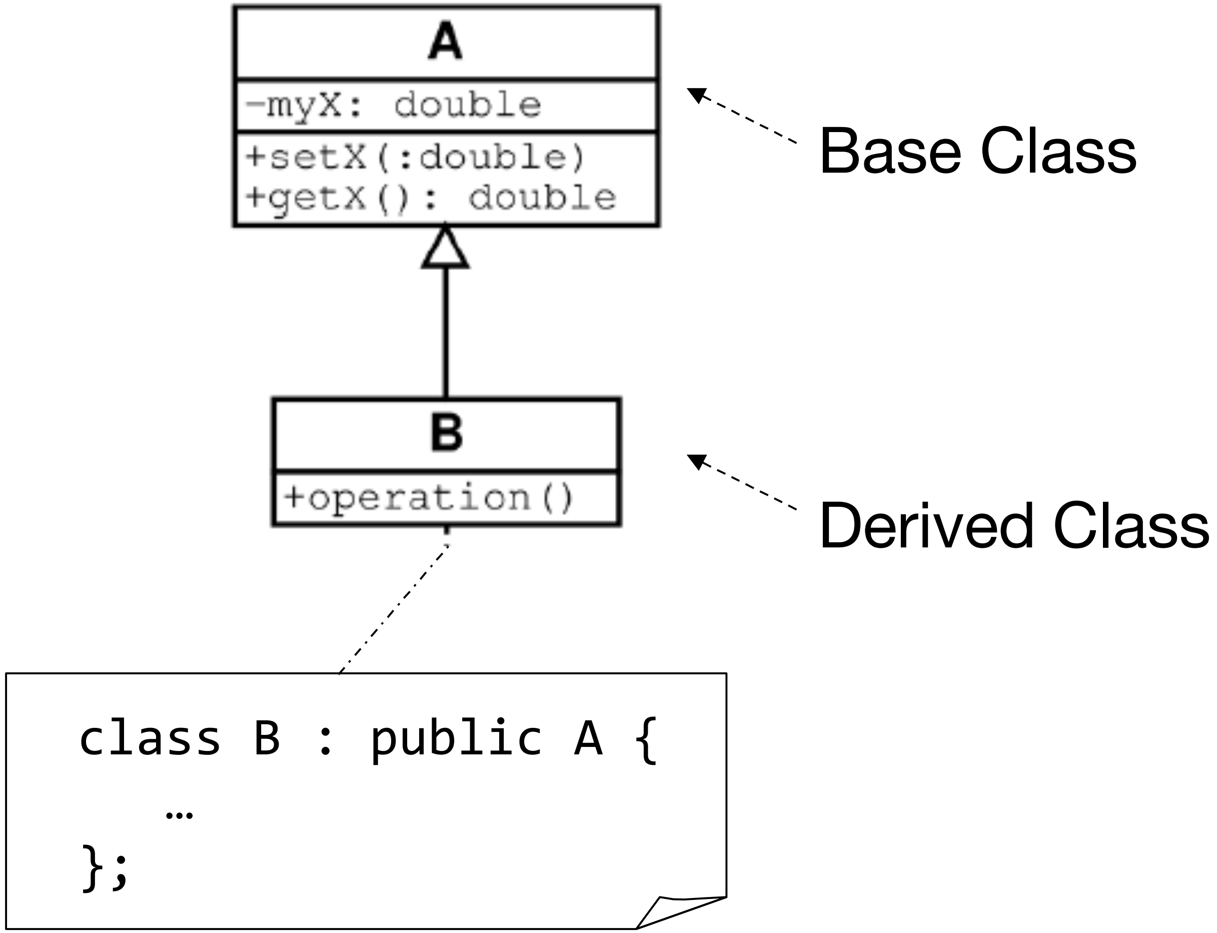


```
members[0] =  new
 Employee();

…

delete members[0];
```

Filled diamond on
side of the Collection

Inheritance

- Standard concept of inheritance



Base Class

Derived Class

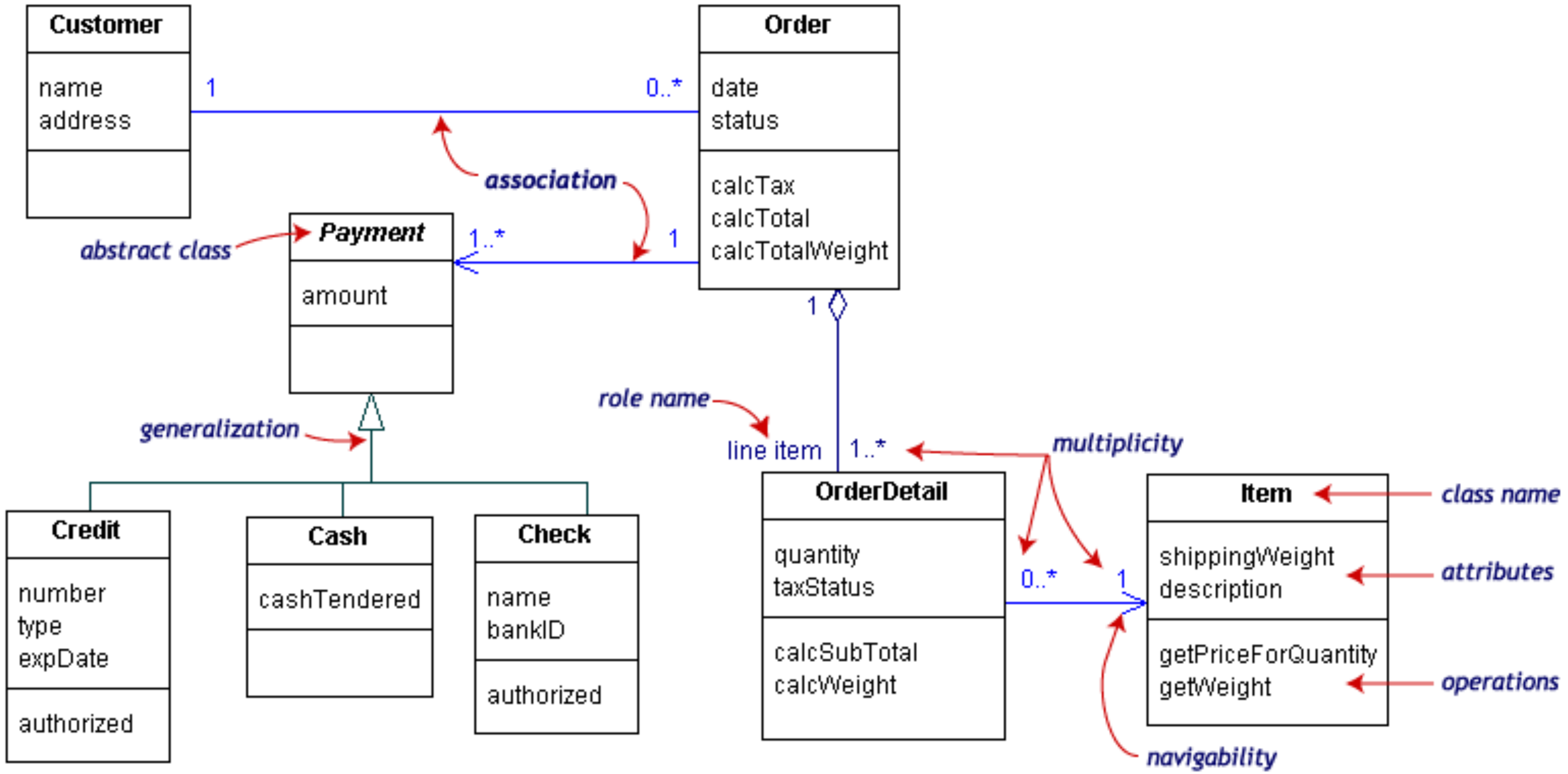
```
class B : public A {
   …
};
```

UML Multiplicities

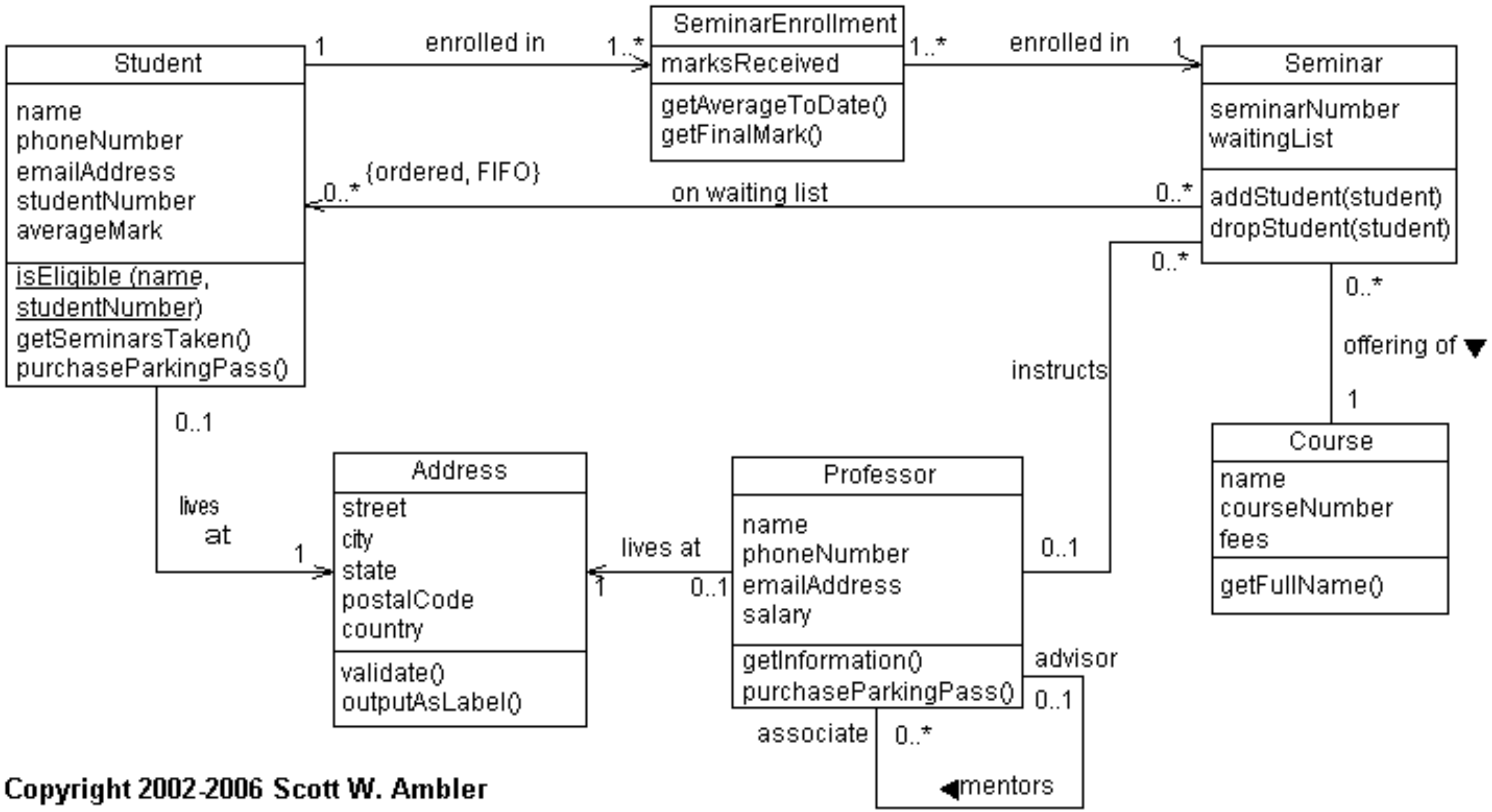
- Links on associations to specify more details about the relationship

Multiplicities	Meaning
n..m	n to m instances
*	no limit on the number of instances (including none)
1	exactly one instance
1..*	at least one instance

Example: UML class diagram



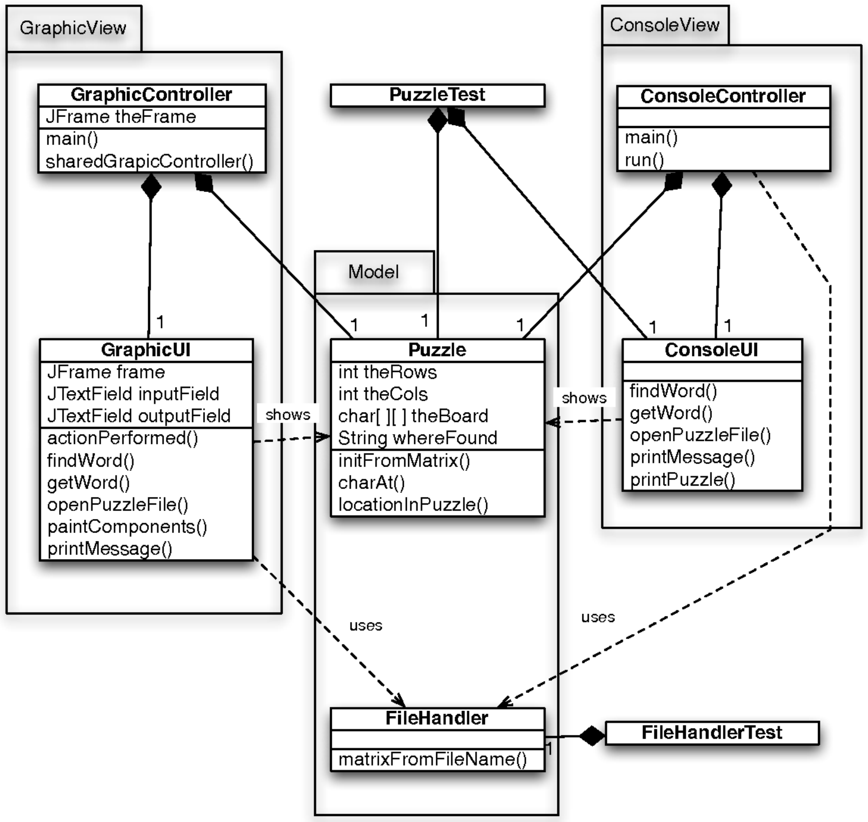
Example: UML class diagram



Copyright 2002-2006 Scott W. Ambler

Example: UML class diagram (word search)

Word Search UML Class Diagram



GraphicView

GraphicController
JFrame theFrame
main()
sharedGrapicController()

GraphicUI
JFrame frame
JTextField inputField
JTextField outputField
actionPerformed()
findWord()
getWord()
openPuzzleFile()
paintComponents()
printMessage()

PuzzleTest

Model

Puzzle
int theRows
int theCols
char[ ][ ] theBoard
String whereFound
initFromMatrix()
charAt()
locationInPuzzle()

FileHandler
matrixFromFileName()

FileHandlerTest

ConsoleView

ConsoleController
main()
run()

ConsoleUI
findWord()
getWord()
openPuzzleFile()
printMessage()
printPuzzle()

Reading list

- Learn C++
  - `this`, static member variables/functions: Ch. 8.8-12
  - template: Ch. 13
- Reference on character encodings
  - Templates: https://isocpp.org/wiki/faq/templates

ANY QUESTIONS?