CSCCORE1I

Object Oriented Software Development

with C++

Lecture 16: Inheritance (II)

Dr Kapil Dev

devk@hope.ac.uk

Learning Objectives

- Inheritance Hierarchies
  - Single Inheritance
  - Multiple Inheritance
  - Levels of Inheritance
- UML and Inheritance
- Inheritance and Program Development

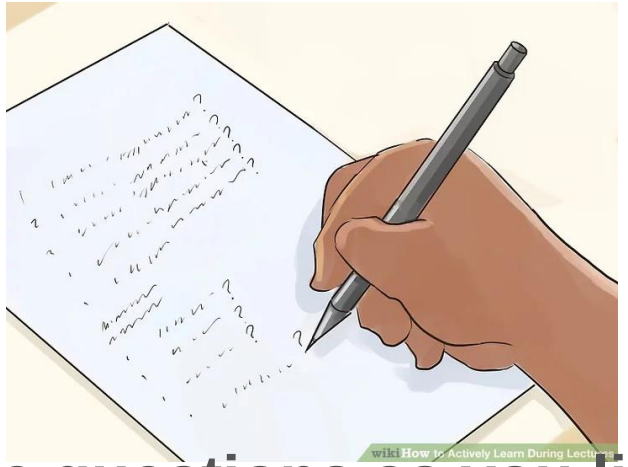
Learning Actively in Online Classes

Create a Workspace

Reach Out

Eliminate Distractions

Jot down main ideas

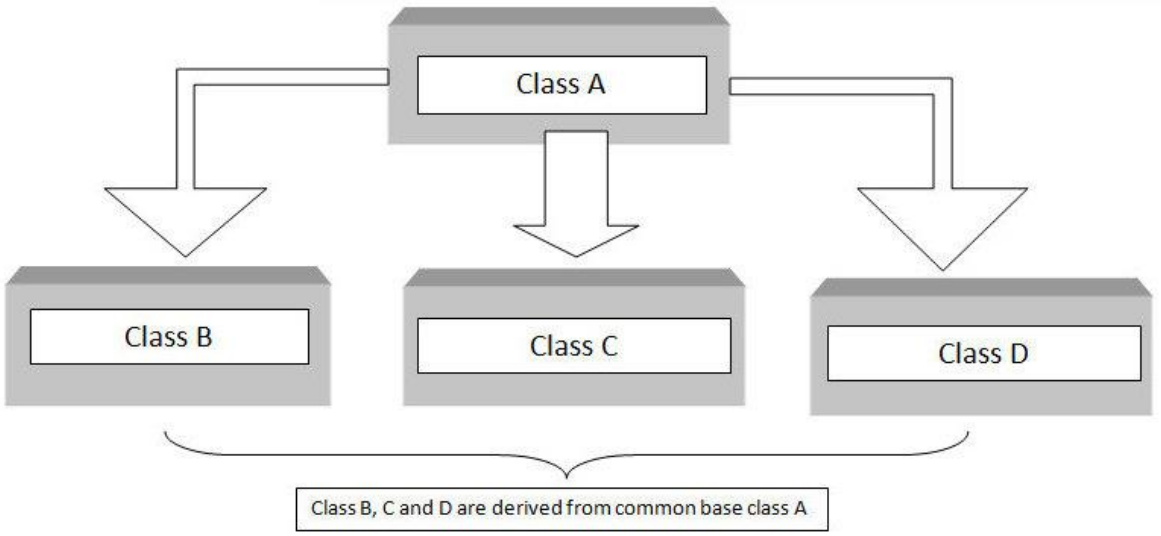


Write questions as you listen

Stay focused

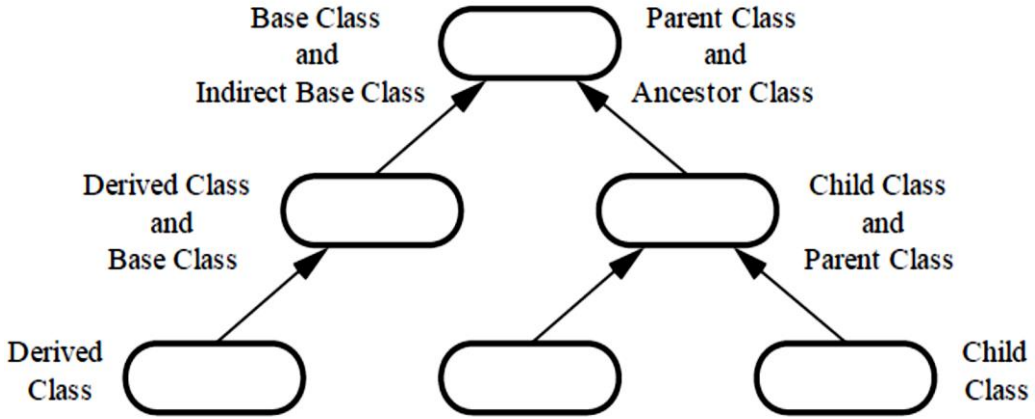
Inheritance

- We know that Inheritance provides a powerful way to extend the capabilities of existing classes, and to design programs using hierarchical relationships.
- In Inheritance, a class, called the derived class, can inherit the features of another class, called the base class.



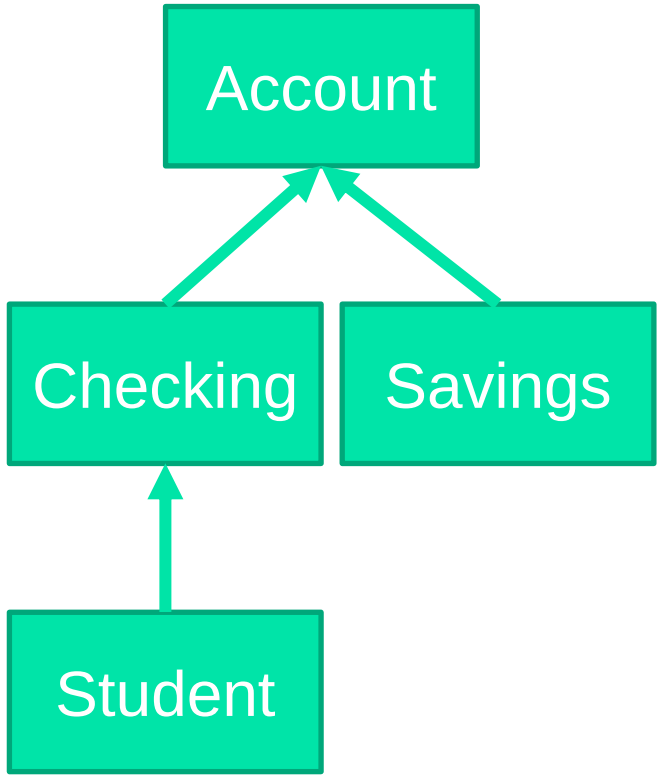
Inheritance Hierarchies

- One class' design can be based on existing
- Can be thought of as using building blocks
- We use base classes to establish the common attributes and behaviour
- Called an ***"is a"*** relationship because a derived class "is a" base class



Single Inheritance

- When a class is derived only from one base class
- For example, in the figure:
  - the base class is account
  - other classes are derived directly or indirectly from account



Single Inheritance Example

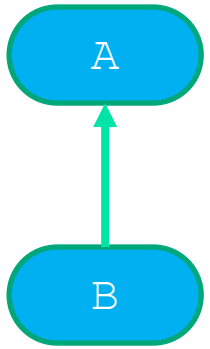
```
class A {
protected:
int aInt;
public:
A(int num): aInt(num)  {
cout << "A::constructor"<<endl;}
~A() {
  cout <<"A::destructor"<<endl;}
};

class B: public A {
public:
B(int numa, int numb)
 : A(numa), bInt(numb) {
cout << "B::constructor"<<endl;}
~B() {
  cout << "B::destructor"<<endl;}
protected:
int bInt;
};
```

```
int main (){
  B b(2, 3);
  return 0;
}
```

Output:

```
A::constructor
B::constructor
B::destructor
A::destructor
```



- Order of execution
  - constructors: base class first then derived class
  - destructors: reverse of constructors

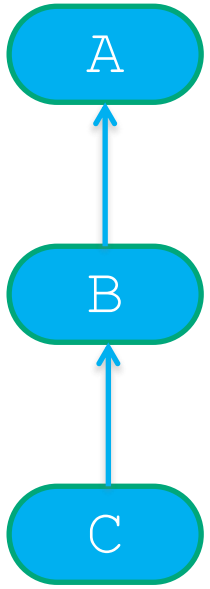
Self-test Exercise

What is the output of the following?

```
// A and B are as declared
// in the previous slide

class C: public B {
public:
C(int numa, int numb, string s )
: B(numa, numb), cStr(s) {
  cout << "C::constructor"<<endl;}
~C() {
  cout << "C::destructor"<<endl;}
protected:
  string cStr;
};

int main () {
  C c(4, 7, "hello");
  return 0;
}
```



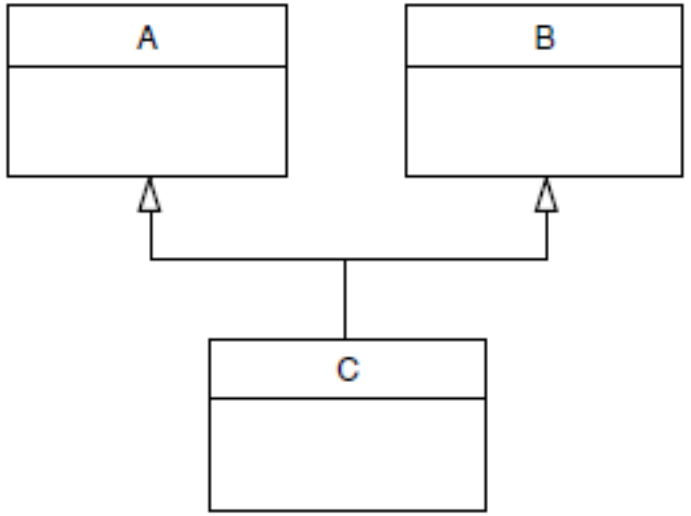
Output:

```
A::constructor
B::constructor
C::constructor
C::destructor
B::destructor
A::destructor
```

Multiple Inheritance

- A class can inherit characteristics and features from more than one parent class

```
class A                          // base class A
   {
   };
class B                          // base class B
   {
   };
class C : public A, public B     // C is derived from A and B
   {
   };
```



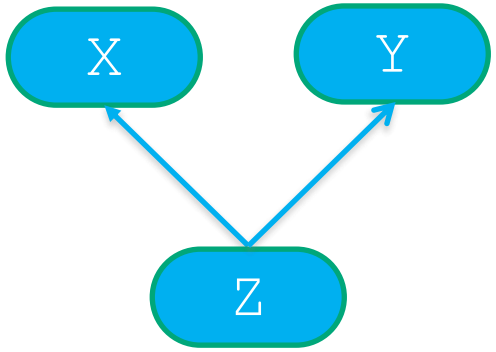
Multiple Inheritance Example

```
class X {
public:
X(): xChar('D')  {cout <<
"X::default constructor"<<endl;;}
X(char c): xChar(c)  {cout <<
"X::constructor"<<endl;;}
~X() {
cout << "X::destructor"<<endl;}
protected:
char xChar;
};
```

```
class Y {
public:
Y(char c): yChar(c)  {
cout << "Y::constructor"<<endl;;}
~Y() {cout <<
"Y::destructor"<<endl;}
protected:
char yChar;
};
```

```
class Z : public X, public Y {
public:
Z(char xC,char yC, int num)
 : X(xC), Y(yC), zInt(num) {
  cout << "Z::constructor"<<endl;}
~Z() {
  cout << "Z::destructor"<<endl;}
protected:
  int zInt;
};
```

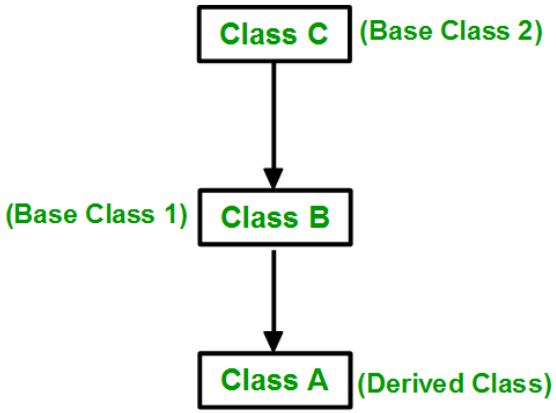
```
int main () {
Z zObj('z', 'b', 8);
return 0; }
```



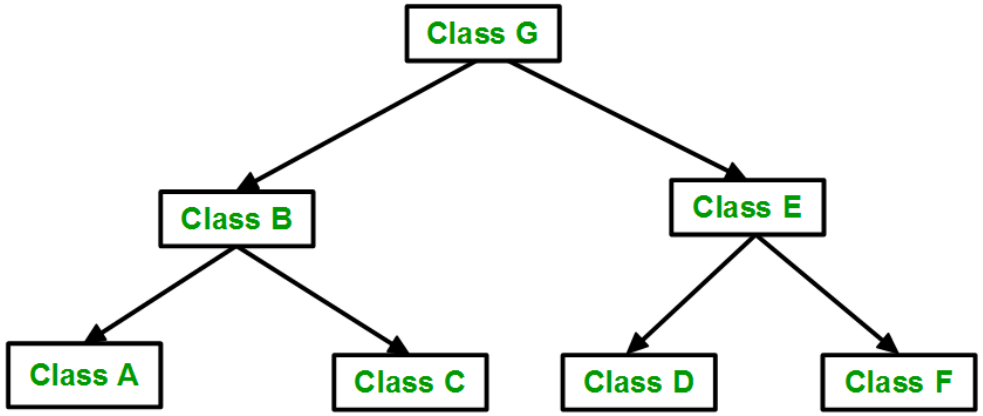
Output:

```
X::constructor
Y::constructor
Z::constructor
Z::destructor
Y::destructor
X::destructor
```

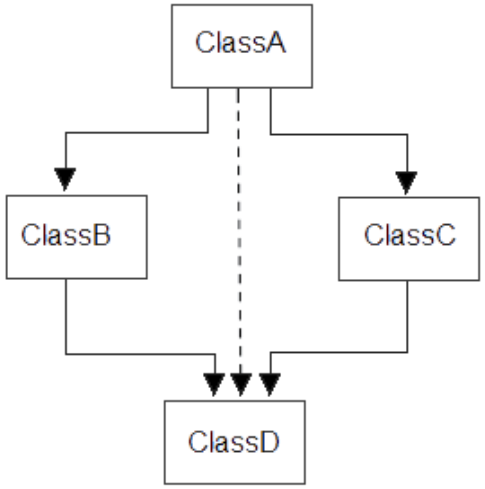
More Possibilities



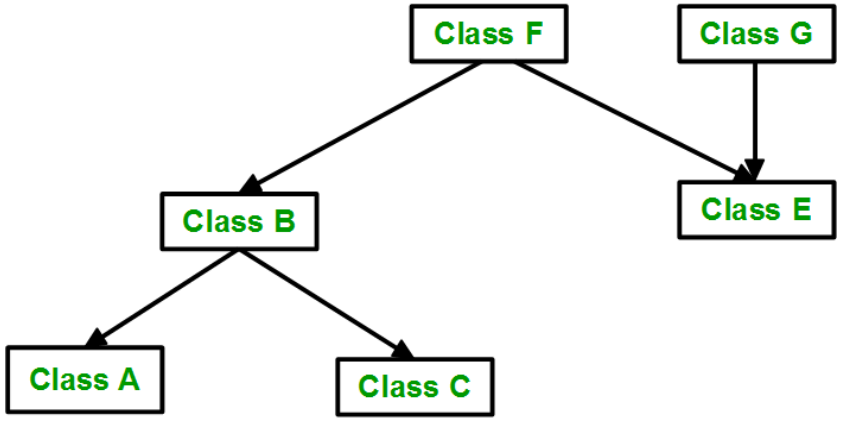
Multilevel Inheritance



Hierarchical Inheritance



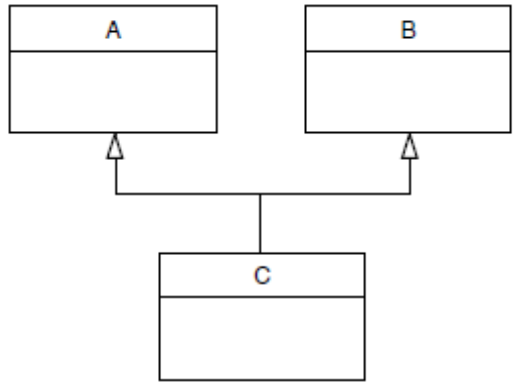
Multipath inheritance



Hybrid (Virtual) Inheritance

Ambiguity in Multiple Inheritance

```
class A
   {
   public:
      void show()  { cout << "Class A\n"; }
   };
class B
   {
   public:
      void show()  { cout << "Class B\n"; }
   };
class C : public A, public B
   {
   };
```

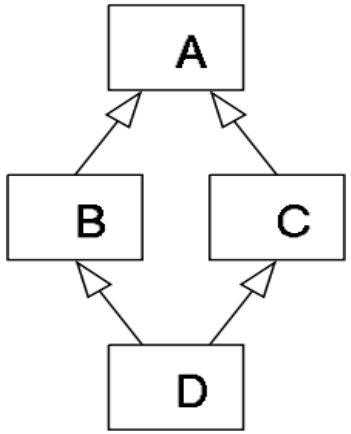



```
int main()
   {
   C objC;            //object of class C
// objC.show();       //ambiguous--will not compile
   objC.A::show();    //OK
   objC.B::show();    //OK
   return 0;
   }
```

Resolved using the scope-resolution operator

```
objC.A::show();
objC.B::show();
```

Ambiguity in Multiple Inheritance



Diamond problem

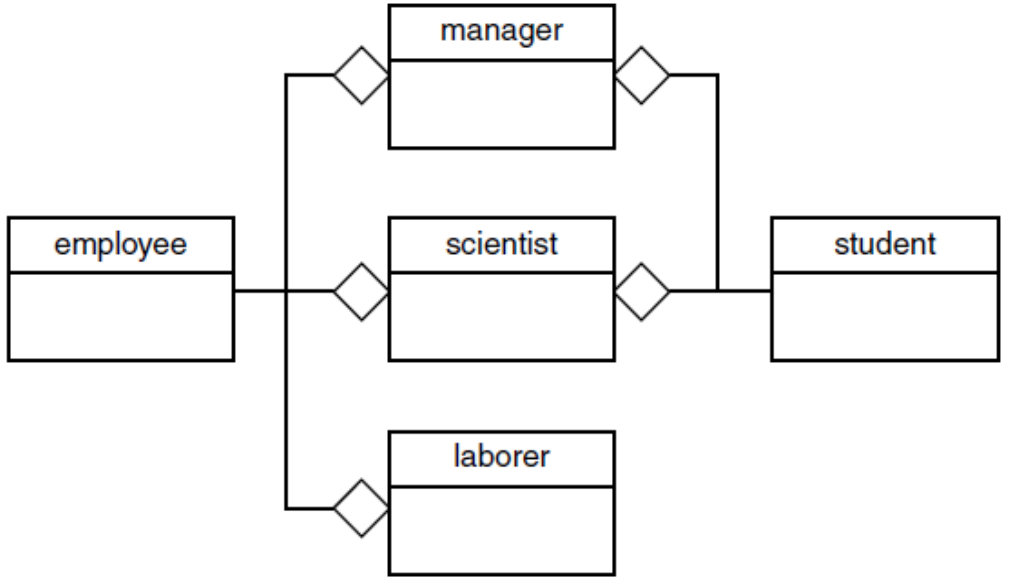
```
class A
   {
   public:
      void func();
   };
class B : public A
   {  };
class C : public A
   {  };
class D : public B, public C
   {  };
```

```
int main()
   {
   D objD;
   objD.func();  //ambiguous: won't compile
   return 0;
   }
```

- Trouble starts if you try to access a member function in class A from an object of class D. In this example objD tries to access func(). However, both B and C contain a copy of func(), inherited from A.
- The compiler can’t decide which copy to use, and signals an error.
- There are various advanced ways of coping with this problem,
- Experts recommend avoiding multiple inheritance altogether

Aggregation: Classes Within Classes

```
class student
   {};
class employee
   {};
class manager
   {
   student stu;     // stu is an object of class student
   employee emp;    // emp is an object of class employee
   };
class scientist
   {
   student stu;     // stu is an object of class student
   employee emp;    // emp is an object of class employee
   };
class laborer
   {
   employee emp;    // emp is an object of class employee
   };
```



Inheritance and Program Development

- The program-development process is being fundamentally altered by object-oriented programming.
  - Programmer A creates a class, say Shape
  - Programmer B likes the A's class and creates a new class, say Circle
  - Programmers C and D then write applications that use the Circle class
- Programmer B may not have access to the source code for the Shape member functions, and programmers C and D may not have access to the source code for Circle. Yet, because of the software reusability feature of C++, B can modify and extend the work of A, and C and D can make use of the work of B (and A).

Self-test Exercise

- Suppose ***Child*** is a class derived from the class ***Parent***, and the class ***Grandchild*** is a class derived from the class ***Child***. This question is concerned with the constructors and destructors for the three classes ***Parent***, ***Child***, and ***Grandchild***. When a constructor for the class ***Grandchild*** is invoked, what constructors are invoked and in what order? When the destructor for the class ***Grandchild*** is invoked, what destructors are invoked and in what order?

The constructors are called in the following order: first Parent, then Child, and finally Grandchild.

The destructors are called in the reverse order: first Grandchild, then Child, and finally Parent.

Inheritance Summary

- A class, called the derived class, can inherit the features of another class, called the base class.
- The derived class can add other features of its own, so it becomes a specialized version of the base class.
- Inheritance provides a powerful way to extend the capabilities of existing classes, and to design programs using hierarchical relationships.
- A class can be derived from more than one base class. This is called multiple inheritance.
- A class can also be contained within another class.

Thank You!