

INHERITANCE

Inheritance

- Inheritance is an “is-a” relationship
 - Example: “every employee is a person”
- Inheritance lets us create new classes from existing classes
 - New classes are called the derived classes
 - Existing classes are called the base classes
- Derived classes inherit the properties of the base classes

Examples

Base class

- Student
- Shape
- Employee

Derived classes

- Graduate Student
- Undergraduate student
- Circle
- Rectangle
- Faculty Member
- Staff member

Inheritance (continued)

- Inheritance can be viewed as a tree-like, or hierarchical, structure wherein a base class is shown with its derived classes

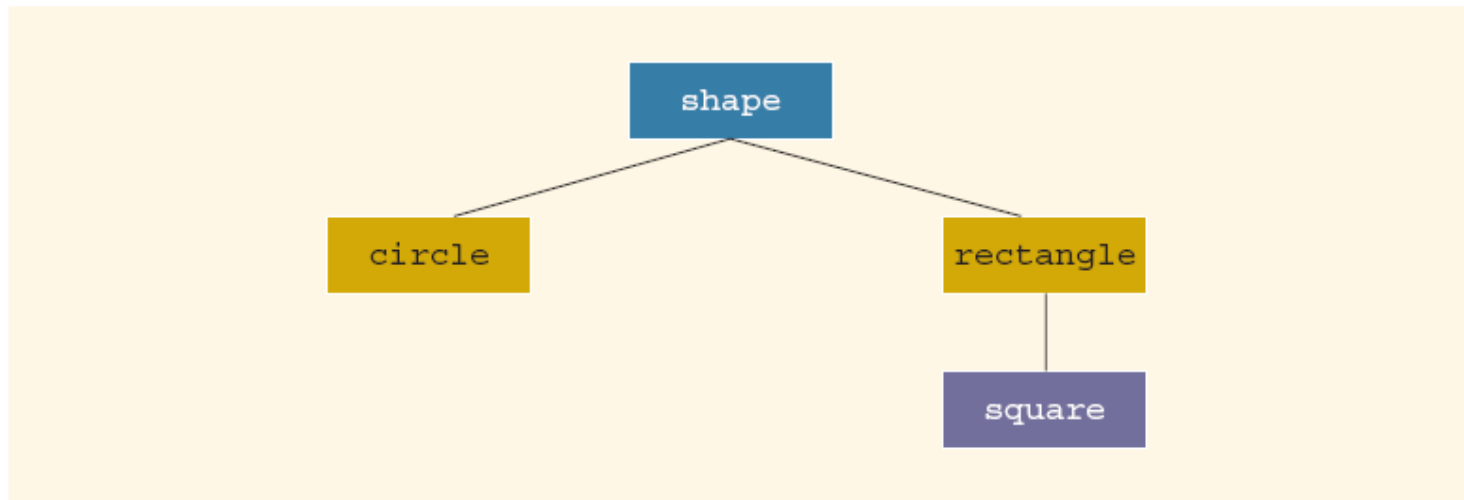


FIGURE 13-1 Inheritance hierarchy

Inheritance (continued)

- Single inheritance: derived class has a single base class
 - Ex. Circle class from Shape class
- Multiple inheritance: derived class has more than one **base class**...will not be discussed in this chapter.
 - Ex. Son class from Mother class and Father class
- Public inheritance: all public members of base class are inherited as public members by derived class

Defining a simple subclass (1)

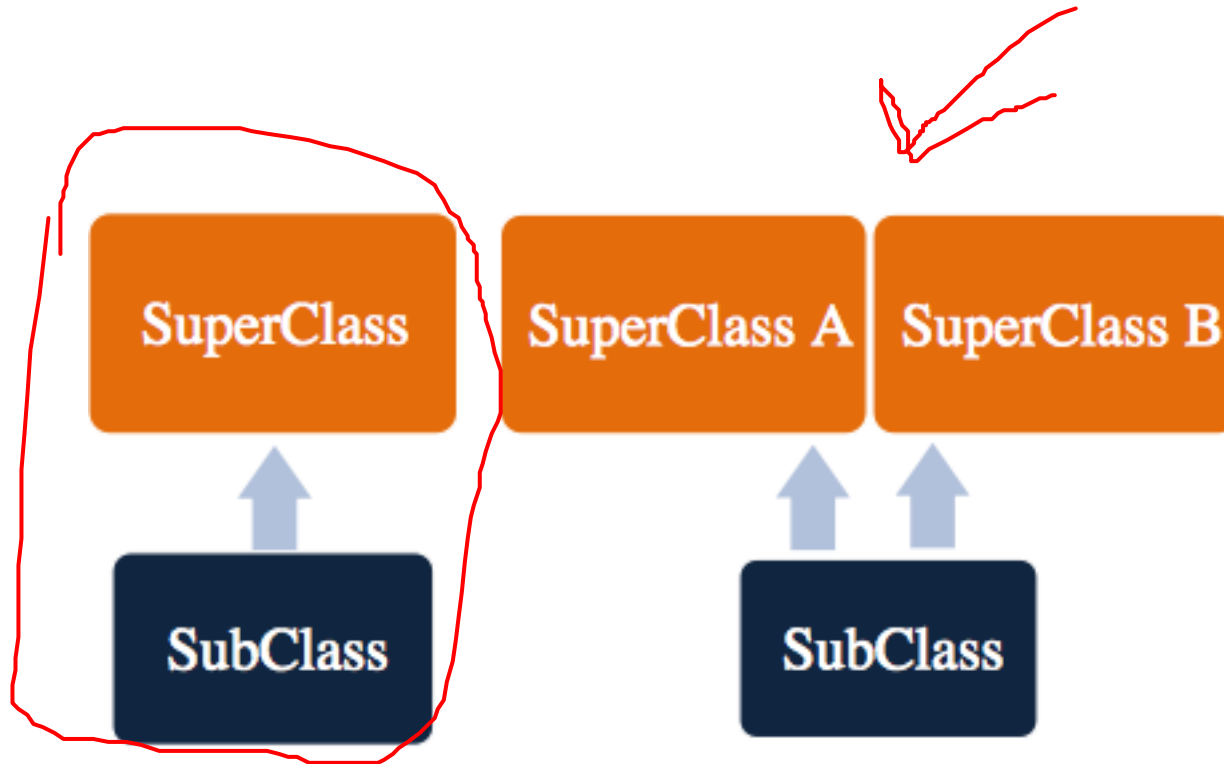
We can use each class as a base (or a foundation) to define or build another class (**a subclass**). It's also possible to use **more than one class to define a subclass**. You can see both of these cases on the right →

Note that **the arrows always point to the superclass(es)**.

The left diagram illustrates a “**single inheritance**”, and the right one a “**multiple inheritance**” or “**multi-inheritance**”.

We'll show you some examples of both types of inheritance.

We can also write about super classes as **base** classes, and subclasses as **derived** classes.



Defining a simple subclass (2)

The class on the right → will serve as a **superclass**. Analyse its structure – it's not difficult, we promise.

The program emits the following text:

101

```
#include <iostream>
using namespace std;
class Super {
private:
    int storage;
public:
    void put(int val) { storage = val; }
    int get(void) { return storage; }
};

int main(void) {
    Super object;

    object.put(100);
    object.put(object.get() + 1);
    cout << object.get() << endl;
    return 0;
}
```


Inheritance (continued)

- General syntax of a derived class:

```
class className: memberAccessSpecifier baseClassName
{
    member list
};
```

- Where memberAccessSpecifier is `public`, `protected`, or `private` (default)
- The `private` members of a base class are private to the derived class
 - Derived class cannot directly access them
 - Ex. `Class SonClass: public FatherClass`
`{` `};`

Defining a simple subclass (3)

If we want to **define a class named Y as a subclass of a superclass named X**, we use the following syntax →

```
class Y : {visibility specifier} X { ... };
```

The difference from the notation we used before is in the fact that we have to:

place **a colon** after the subclass class name

optionally place a so-called **visibility specifier** (we'll return to this soon)

add **a superclass name**

If there's more than one superclass, we have to enlist them all using commas as separators, like this:

```
class A : X, Y, Z { ... };
```

Let's start with the simplest possible case.

Defining a simple subclass (4)

Take a look here → We've defined a class named *Sub*, which is a subclass of a class named *Super*. We may also say that the *Sub* class is derived from the *Super* class.

The *Sub* class introduces neither new variables nor new functions. Does this mean that any object of the *Subclass* inherits all the traits of the *Super* class, being in fact a copy of the *Super* class's objects?

No. It doesn't.

```
class Sub : Super {  
};
```

```
int main(void) {  
    Sub object;
```

```
    object.put(100);  
    object.put(object.get() + 1);  
    cout << object.get() << endl;  
    return 0;
```

```
}
```

Defining a simple subclass (4) cont'd

If we compile the following code, we'll get nothing but compilation errors saying that the *put* and *get* methods are inaccessible. Why?

When we **omit the visibility specifier**, the compiler assumes that we're going to apply a "**private inheritance**". This means that **all public superclass components turn into private access, and private superclass components won't be accessible** at all. It consequently means that you're not allowed to use the latter inside the subclass.

This is exactly what we want now.

Defining a simple subclass (4) cont'd

We have to tell the compiler that **we want to preserve the previously used access policy**. We do this by using a “public” visibility specifier:

```
class Sub : public Super {  };
```

Don't be misled: this doesn't mean that the private components of the *Superclass* (like the *storagevariable*) will magically turn into public ones. Private components will remain private, public components will remain public.

Defining a simple subclass (5)

Objects of the *Sub* class may do almost the same things as their older siblings created from the *Superclass*. We use the word 'almost' because being a subclass also means that **the class has lost access to the private components of the superclass**. We cannot write a member function of the *Sub* class which would be able to directly manipulate the *storage* variable.

```
#include <iostream>

using namespace std;

class Super {
private:
    int storage;
public:
    void put(int val) { storage = val; }
    int get(void) { return storage; }
};

class Sub : public Super {
};

int main(void) {
    Sub object;

    object.put(100);
    object.put(object.get() + 1);
    cout << object.get() << endl;
    return 0;
}
```

Defining a simple subclass (5) cont'd

This is a very serious restriction. Is there any workaround?
Yes.

There's the third access level we haven't mentioned yet. It's called "**protected**".

The keyword *protected* means that any component marked with it **behaves like a public component when used by any of the subclasses and looks like a private component to the rest of the world.**

We should add that this is true only for publicly inherited classes (like the *Super* class in our example previous example)

Let's make use of the keyword right now.

Defining a simple subclass (6)

As you can see in the example code → we've added some new functionality to the *Sub* class.

We've added the *print* function. It isn't especially sophisticated, but it does one important thing: it accesses the *storage* variable from the Superclass. This wouldn't be possible if the variable was declared as **private**.

In the *main* function scope, the variable remains hidden anyway. You mustn't write anything like this:

```
object.storage = 0;
```

The compiler will be very stubborn about this.

We almost forgot to mention that our new program will produce the following output:
storage = 101

```
#include <iostream>

using namespace std;

class Super {
protected:
    int storage;
public:
    void put(int val) { storage = val; }
    int get(void) { return storage; }
};

class Sub : public Super {
public:
    void print(void) { cout << "storage = " << storage << endl; }
};

int main(void) {
    Sub object;

    object.put(100);
    object.put(object.get() + 1);
    object.print();
    return 0;
}
```



```
#include <iostream>
using namespace std;

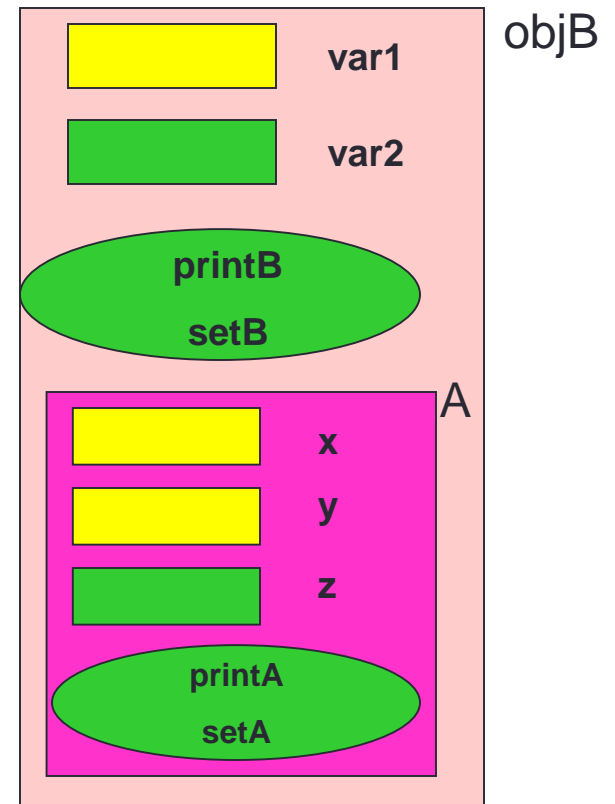
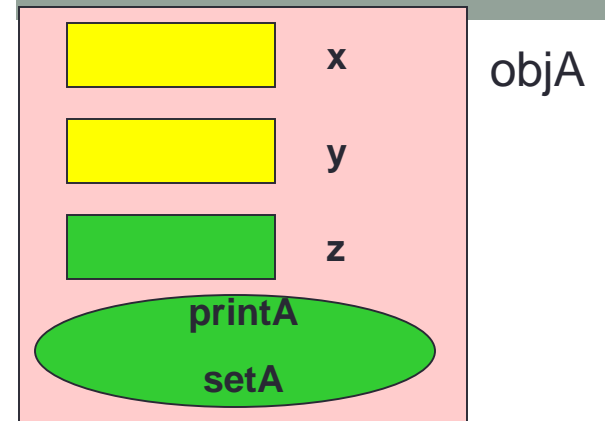
class A
{
    int x;
    int y;
public:
    int z;
    void printA(){cout<<x<<" "<<y<<" "<<endl;}
    void setA(int a, int b, int c){x=a; y=b; z=c;}
};
```

Base

```
class B: public A
{
    int var1;
public:
    int var2;
    void setB(int a, int b) {var1=a; var2=b;}
    void printB(){cout<<var1<<" "<<var2<<endl;}
};
```

derived

```
void main()
{
    A objA;
    B objB;
}
```



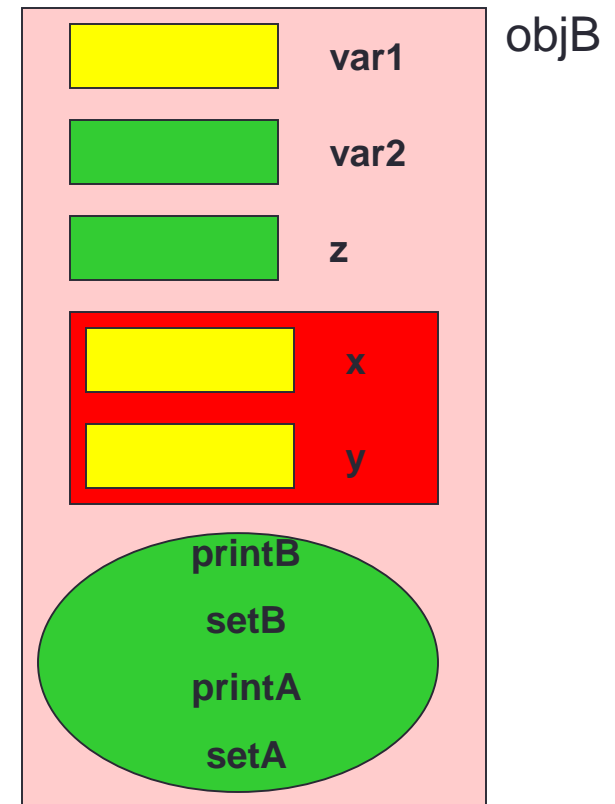
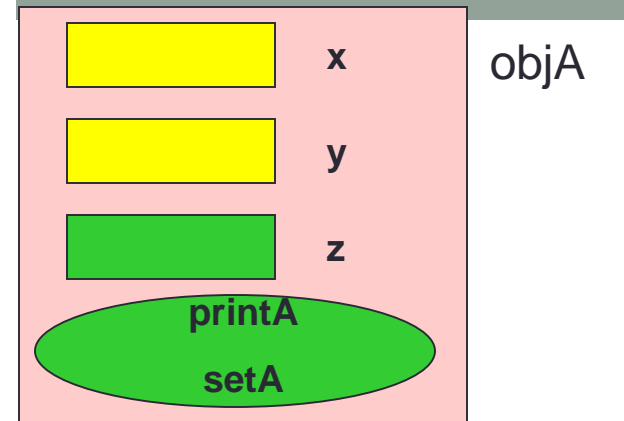
```
#include <iostream>
using namespace std;;

class A
{
    int x;
    int y;
public:
    int z;
    void printA(){cout<<x<<" "<<y<<" "<<endl;}
    void setA(int a, int b, int c){x=a; y=b; z=c;}
};
```

```
class B: public A
{
    int var1;
public:
    int var2;
    void setB(int a, int b) {var1=a; var2=b;}
    void printB(){cout<<var1<<" "<<var2<<endl;}
};
```

```
void main()
{
    A objA;
    B objB;
}
```

Can not access x, y

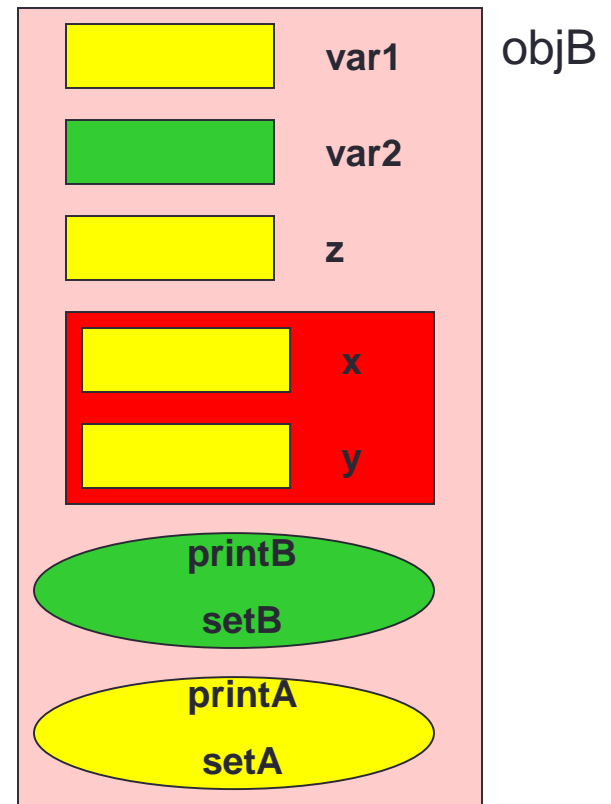
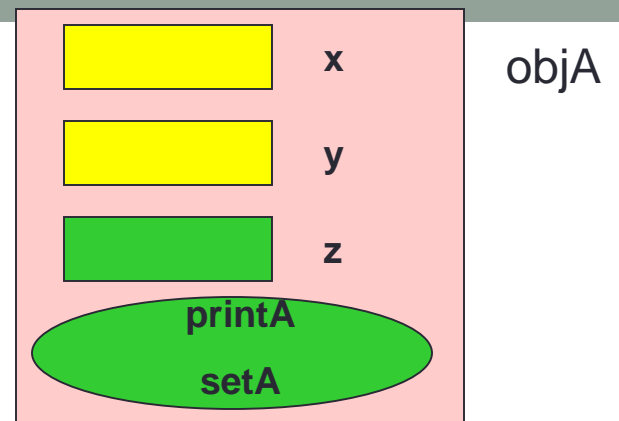


```
#include <iostream>
using namespace std;;

class A
{
    int x;
    int y;
public:
    int z;
    void printA(){cout<<x<<" "<<y<<" "<<endl;}
    void setA(int a, int b, int c){x=a; y=b; z=c;}
};
```

```
class B: private A ★
{
    int var1;
public:
    int var2;
    void setB(int a, int b) {var1=a; var2=b;}
    void printB(){cout<<var1<<" "<<var2<<endl;}
};
```

```
void main()
{
    A objA;
    B objB;
}
```



```

#include <iostream>
using namespace std;
class A
{
    int x;
    int y;
public:
    int z;
    void printA(){cout<<x<<"    "<<y<<"    "<<z<<endl;}
    void setA(int a, int b, int c){x=a; y=b; z=c;}
};
class B:public A
{
    int var1;
public:
    int var2;
    void setB(int a, int b) {var1=a; var2=b;}
    void printB(){cout<<var1<<"    "<<var2<<endl;}
};

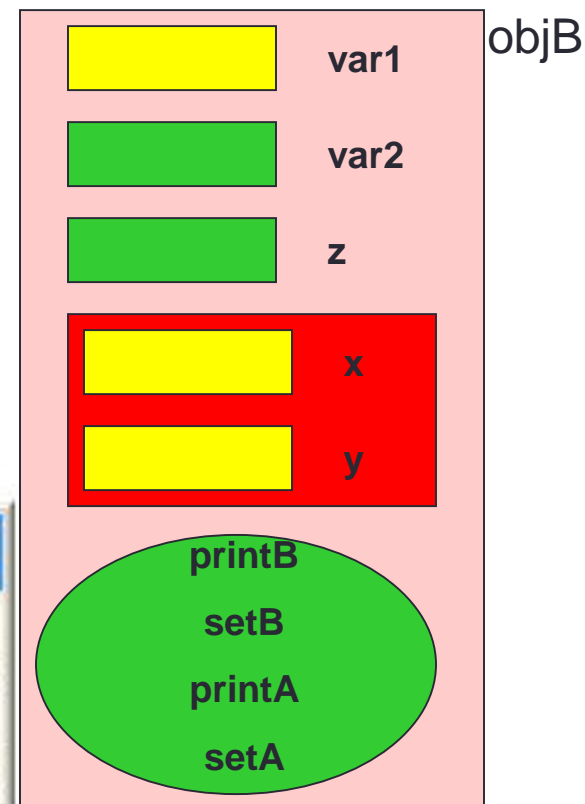
void main()
{
    B objB;
    objB.setA(3,4,5);
    objB.setB(7,8);
    objB.printA();
    objB.printB();
}

```

```

3    4    5
7    8
Press any key to continue

```



Defining a simple subclass (7)

Now's a good opportunity to do a little summarising here. We know that any component of the class may be declared as:

- public
- private
- Protected

These three keywords may also be used in a completely different context to specify the visibility inheritance model. So far, we've talked about public and private keywords used in such a case. It should be no surprise to you that the protected keyword can be employed in this role, too.

Inheritance (continued)

- `public` members of base class can be inherited as `public` or `private` members
- The derived class can include additional members--data and/or functions
- The derived class can redefine the `public` member functions of the base class
- All members of the base class are also member variables of the derived class

Defining a simple subclass (7) cont'd

Take a look at the table here →
It gathers all of the possible combinations of the component declaration and inheritance model, presenting the resulting access to the components when the subclass is completely defined.
It reads in the following way (take a look at the first row): if a component is declared as public and its class is inherited as public, the resulting access is public.

Familiarize yourself with the table – it's a basic tool to resolve all the issues regarding the inheritance of the class components.

When the component is declared as:	When the class is inherited as:	The resulting access inside the subclass is:
public	public	Public
protected		protected
private		none
public	protected	protected
protected		protected
private		none
public	private	private
protected		private
private		none

Redefining (Overriding) Member Functions of the Base Class

- To redefine (override) a `public` member function of a base class
 - Corresponding function in the derived class must have the same name, number, and types of parameters. Redefined
 - If the function has a different signature, this would be function overloading.

Redefining (Overriding) Member Functions of the Base Class (continued)

If derived class overrides a `public` member function of the base `class`, then to call the base class function, specify:

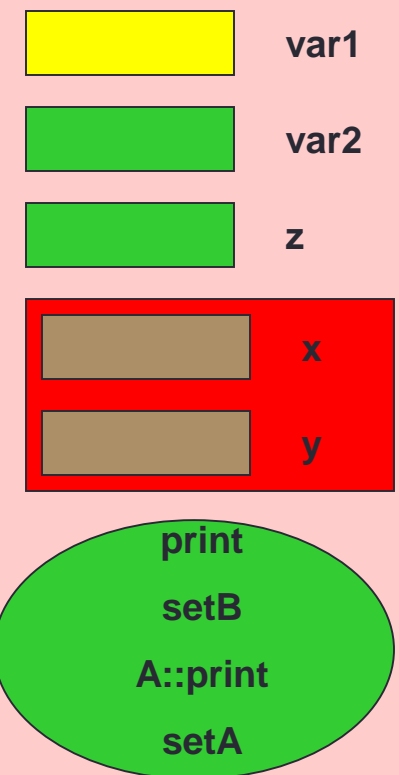
- Name of the base class
- Scope resolution operator (`::`)
- Function name with the appropriate parameter list

```

#include <iostream>
using namespace std;;
class A
{
    int x;
    int y;
public:
    int z;
    void setA(int a, int b, int c){x=a; y=b; z=c;}
    void print(){cout<<x<<" "<<y<<" "<<z<<endl;}
};
class B:public A
{
    int var1;
public:
    int var2;
    void setB(int a, int b) {var1=a; var2=b; }
    void setB(int a) {var1=a; var2=a;}
    void print(){cout<<var1<<" "<<var2<<endl; A::print(); }
};

void main()
{
    B objB;
    objB.setB(7,8);
    objB.setA(1,2,3);
    objB.print();
    objB.A::print();
}

```



```

C:\WINDOWS\system32\cmd.exe
7 8
1 2 3
1 2 3
Press any key to continue .

```

Constructors of Derived and Base Classes

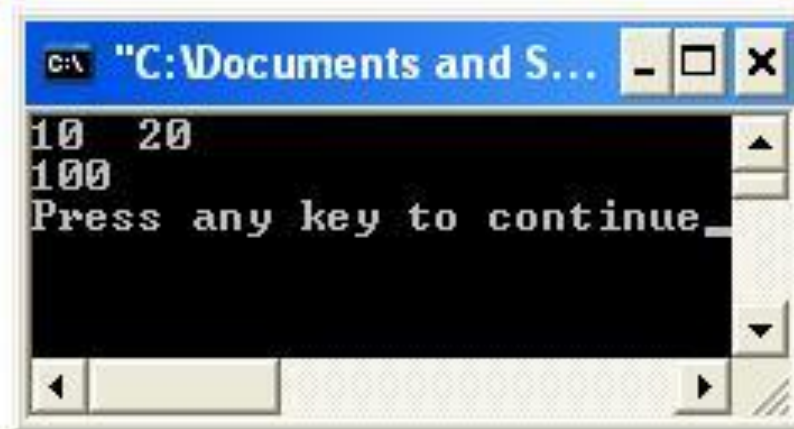
- Derived class constructor cannot directly access `private` members of the base class
- Derived class can directly initialize only `public` member variables of the base class
- When a derived object is declared
 - It must execute one of the base class constructors
- Call to base class constructor is specified in heading of derived class constructor definition

Example

```
class A
{
public:
    int x;
    void print()
    {cout<<x<<"    "<<y<<endl;}
    A() { x=10; y=20;}
private:
    int y;
};

class B:public A
{
public:
    int z;
    void print() {A::print(); cout<<z<<endl;}
    B(){ z=100;}
};

void main()
{
    B objB;
    objB.print();
}
```

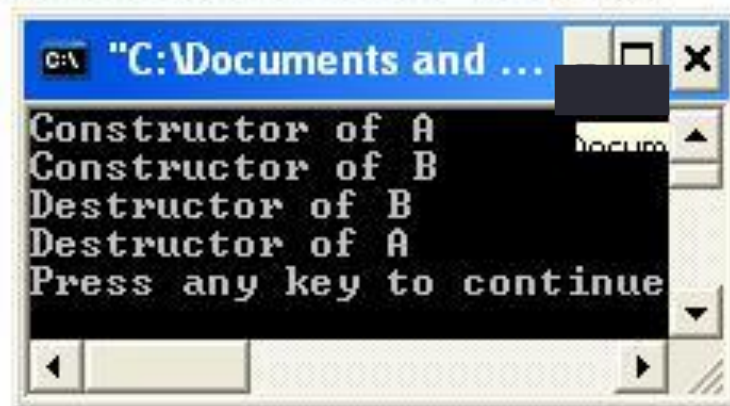


Example

```
class A
{
public:
    int x;
    A() { x=10; y=20; cout<< "Constructor of A"<<endl;}
    ~A(){cout<<"Destructor of A"<<endl;}
private:
    int y;
};

class B:public A
{
public:
    int z;
    B(){z=100; cout<<"Constructor of B"<<endl;}
    ~B(){cout<<"Destructor of B"<<endl;}
};

void main()
{
    B objB;
}
```

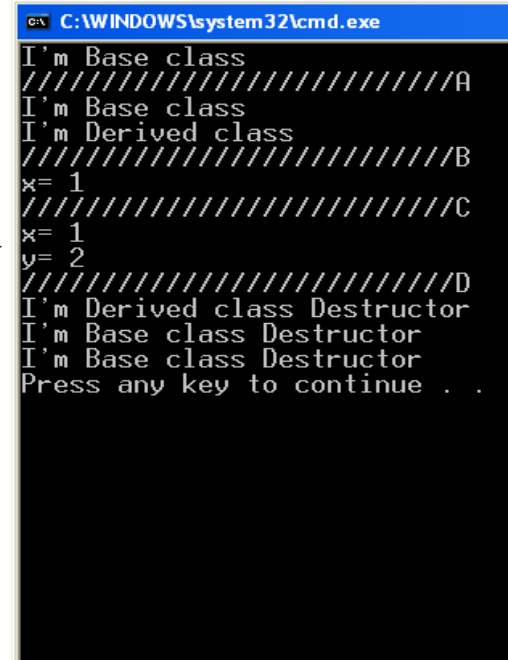


Example

```
class Base{
    int x;
public:
    Base(){ x=1; cout<<"I'm Base class\n"; }
    ~Base(){ x=1; cout<<"I'm Base class Destructor\n"; }
    void print(){cout<<"x= "<<x<<endl;}
};

class Derived: public Base{
    int y;
public:
    Derived(){ y=2; cout<<"I'm Derived class\n";
        // x=1; illegal
    }
    ~Derived(){ y=2; cout<<"I'm Derived class Destructor\n";}
    void print(){ Base::print(); cout<<"y= "<<y<<endl; }
};

int main(){
    Base obj1;
    cout<<"////////////////////////A\n";
    Derived obj2;
    cout<<"////////////////////////B\n";
    obj1.print();
    cout<<"////////////////////////C\n";
    obj2.print();
    cout<<"////////////////////////D\n";
    return 0;
}
```



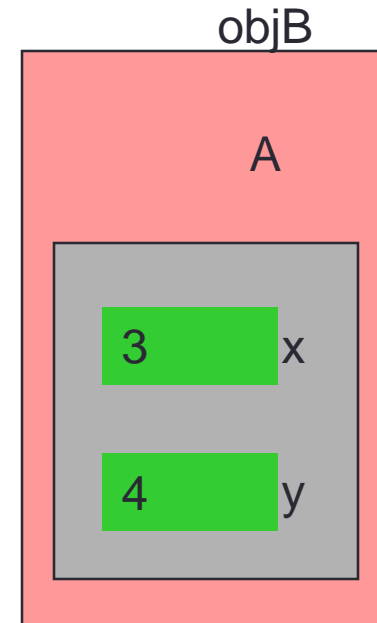
```
C:\WINDOWS\system32\cmd.exe
I'm Base class
////////////////////////A
I'm Base class
I'm Derived class
////////////////////////B
x= 1
////////////////////////C
x= 1
y= 2
////////////////////////D
I'm Derived class Destructor
I'm Base class Destructor
I'm Base class Destructor
Press any key to continue ..
```

Initialization List and Inheritance

```
class A
{
public:
    int x;
    A(int a,int b) { x=a; y=b;}
private:
    int y;
};

class B:public A
{
public:
    B(int a,int b): A(a,b){}
};

void main()
{
    B objB(3,4);
}
```



Initialization list goes with constructor of derived class and uses class name to pass parameters to base class constructor.

Protected Members of a Class

- Private members of a class cannot be directly accessed outside the class
- For a base class to give derived class access to a `private` member
 - Declare that member as `protected`
- The accessibility of a `protected` member of a class is in between `public` and `private`
 - A derived class can directly access the `protected` member of the base class

Inheritance as `public`, `protected`, or `private`

```
class B: memberAccessSpecifier A
```

- If `memberAccessSpecifier` is `public`:
 - `public` members of A are `public` members of B and can be directly accessed in class B
 - `protected` members of A are `protected` members of B and can be directly accessed by member functions (and friend functions) of B
 - `private` members of A are hidden in B and can be accessed by member functions of B through `public` or `protected` members of A

Inheritance as `public`, `protected`, or `private` (continued)

```
class B: memberAccessSpecifier A
{
```

- If `memberAccessSpecifier` is `protected`:
 - `public` members of `A` are `protected` members of `B` and can be accessed by the member functions (and friend functions) of `B`
 - `protected` members of `A` are `protected` members of `B` and can be accessed by the member functions (and friend functions) of `B`
 - `private` members of `A` are hidden in `B` and can be accessed by member functions of `B` through `public` or `protected` members of `A`

Inheritance as `public`, `protected`, or `private` (continued)

```
class B: memberAccessSpecifier A
{
```

- If `memberAccessSpecifier` is `private`:
 - `public` members of `A` are `private` members of `B` and can be accessed by member functions of `B`
 - `protected` members of `A` are `private` members of `B` and can be accessed by member functions (and friend functions) of `B`
 - `private` members of `A` are hidden in `B` and can be accessed by member functions of `B` through `public/protected` members of `A`

```

class A
{
public:
    int x;
private:
    int y;
protected:
    int z;
};

class B:public A
{
public:
    void fun()
    {
        x=3;
        // y=4; illegal
        z=5;
    }
protected:
    int w;
};

void main()
{
    B objB;
    objB.fun();
    //objB.w=7;    illegal
    objB.x=5;
    //objB.y=12;    illegal
    //objB.z=10;    illegal
}

```

```

class A
{
public:
    int x;
private:
    int y;
protected:
    int z;
};

class B:protected A
{
public:
    void fun()
    {
        x=3;
        // y=4; illegal
        z=5;
    }
protected:
    int w;
};

void main()
{
    B objB;
    objB.fun();
    //objB.w=7;    illegal
    //objB.x=5;    illegal
    //objB.y=12;    illegal
    //objB.z=10;    illegal
}

```

```

class A
{
public:
    int x;
private:
    int y;
protected:
    int z;
};

class B:private A
{
public:
    void fun()
    {
        x=3;
        // y=4; illegal
        z=5;
    }
protected:
    int w;
};

void main()
{
    B objB;
    objB.fun();
    //objB.w=7;    illegal
    //objB.x=5;    illegal
    //objB.y=12;    illegal
    //objB.z=10;    illegal
}

```

Defining a simple subclass (8)

We'll finish our current topic with a very simple example demonstrating multi-inheritance. We need to emphasize that using this technique is commonly recognized as error-prone and obfuscating class hierarchy.

Any solution that avoids multi-inheritance is generally better and in fact many contemporary object programming languages don't offer multi-inheritance at all. We think it's a good argument to consider when you're making design assumptions.

Defining a simple subclass (8) cont'd

This example should be clear (we hope).

The program will produce the following output:

storage = 3

safe = 5

```
#include <iostream>
using namespace std;
class SuperA {
protected:
    int storage;
public:
    void put(int val) { storage = val; }
    int get(void) { return storage; }
};
class SuperB {
protected:
    int safe;
public:
    void insert(int val) { safe = val; }
    int takeout(void) { return safe; }
};
class Sub : public SuperA, public SuperB {
public:
    void print(void) {
        cout << "storage = " << storage << endl;
        cout << "safe  = " << safe << endl;
    }
};
int main(void) {
    Sub object;

    object.put(1);    object.insert(2);
    object.put(object.get() + object.takeout());
    object.insert(object.get() + object.takeout());

    object.print();
    return 0;
}
```