OBJECT ORIENTED PROGRAMMING

Inheritance

(continued)

Inheritance types

Accessibility:

Private < Protected < Public

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

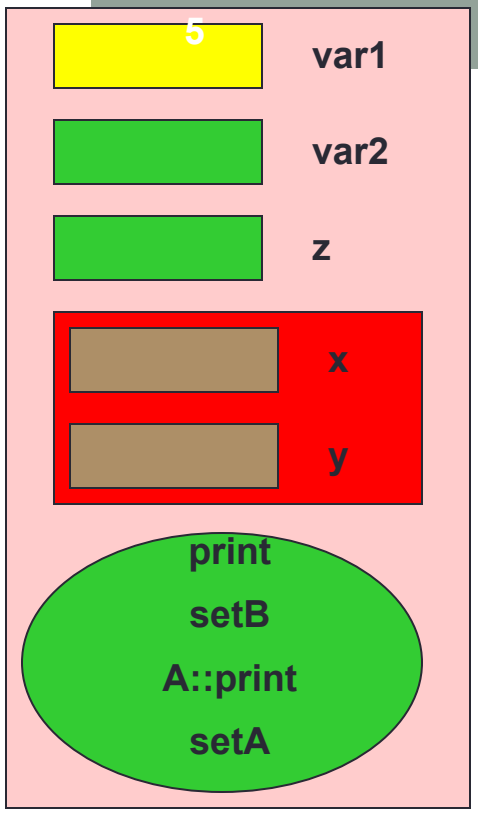
- To redefine (override) a `public` member function of a base class
  - Corresponding function in the derived class must have the same name, number, and types of parameters. Redefined
  - If the function has a different signature, this would be function overloading.

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();
}
```

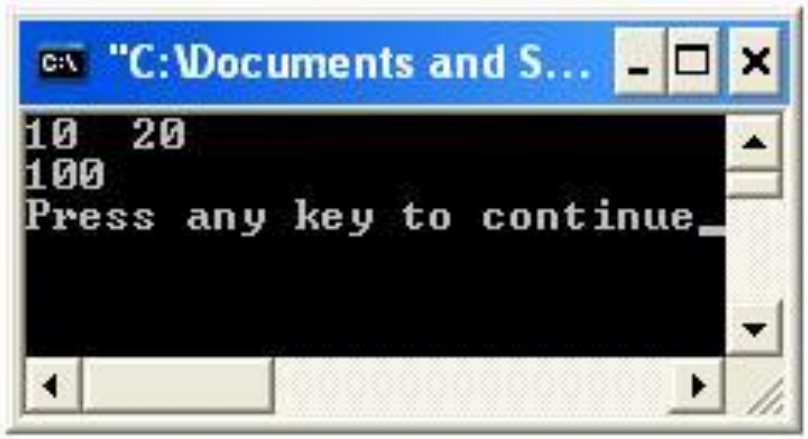


Constructors of Derived and Base Classes

- 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

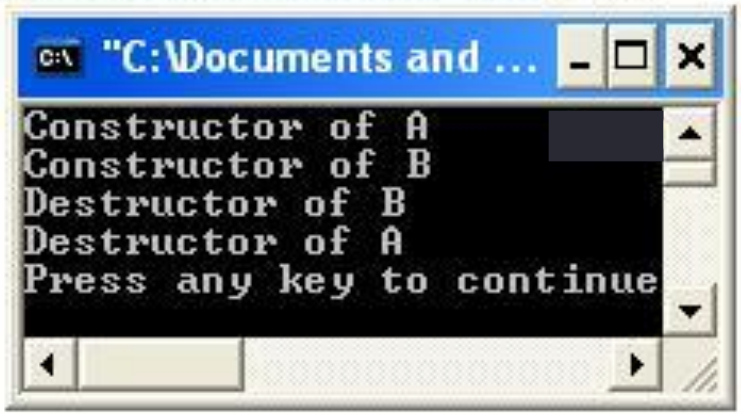
Example

```
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();
}
```



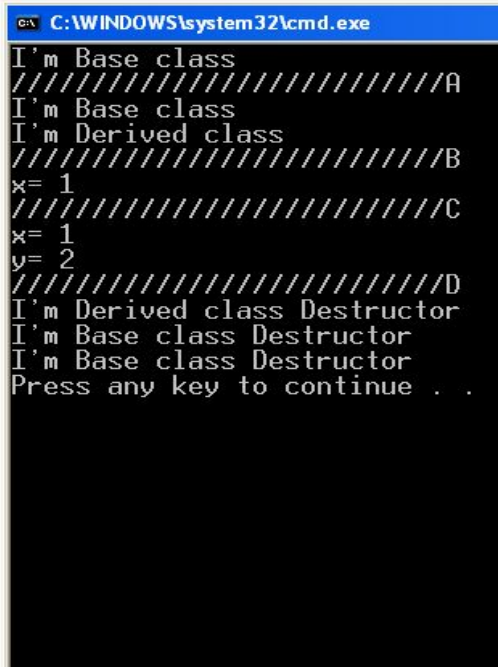
Example

```
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;
}
```



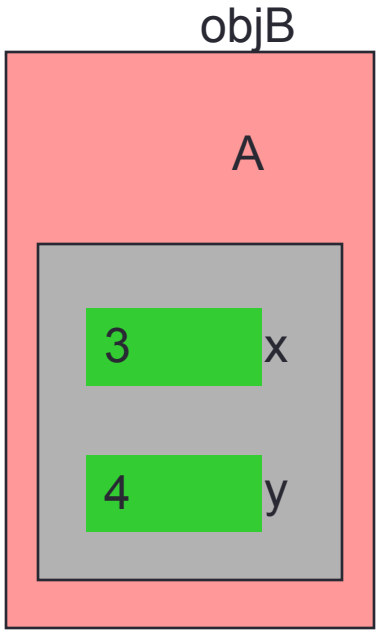
Example

```
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;
}
```



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;
//objB.y=12;   illegal
//objB.z=10;   illegal
}
```

6.1.8 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.

6.1.8 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;
}
```

Summary

- Inheritance and composition are meaningful ways to relate two or more classes
- Inheritance is an “is-a” relation
  - Single inheritance: a derived class is derived from one class, called the base class
  - Multiple inheritance: a derived class is derived from more than one base class
- Composition and aggregation are “has-a” relation

Summary (continued)

- Private members of a base class are private to the base class
- Public members of a base class can be inherited either as public or private
- Derived class can redefine function members of a base class
  - Redefinition applies only to objects of derived class

Summary (continued)

- A call to a base class constructor (with parameters) is specified in the heading of the definition of the derived class constructor
- When initializing object of a derived class, the base class constructor is executed first
- In composition
  - Class member is an object of another class
  - Call to constructor of member objects is specified in heading of the definition of class's constructor

EXAMPLES ON COMPOSITION AND INHERITENCE

Answer the following questions based on the classes' definitions shown next.

```
class first
{
    int var1;
  protected:
    int var2;
  public:
    int var3;
};
class second : private first
{    int var4;
  protected:
    int var5;
  public:
    void fun(){ }
};
class third : public second
{
  int var6;
 public:
  int var7;
};
```

OUTPUT QUESTIONS

```
class MyClass
{
public:
        int a;
        MyClass() {a=0; cout<<"First"<<endl; }
        MyClass(int i){a=i; cout<<"Second"<<endl;}
        ~MyClass(){cout<<a<<endl;}
};

class YourClass: protected MyClass
{
        int x;
        MyClass M1;
public:
        MyClass M2;
        YourClass(int i): M2(3), MyClass(i) {x=i; a=7;}
};

void main()
{
        YourClass obj(5);
}
```

```
Second
First
Second
3
0
7
```

```
class A{
   int x, y;

public:
   A(){ x = 2; y = 4; }
   A(int i){ x = i; y = x + 5; }
   void pr(){ cout << x << endl <<y<<endl;}
};

class B{
   int z;
   A a1, a2;
public:
   B(): a2(2) { z = 5; pr(); }
   B(int i): a2(5) { z = i + 10; }
   void pr(){ a1.pr(); a2.pr(); cout<<z<< endl;}
};


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

```
2
4
2
7
5
```

```
class A
{
public:
        A(){cout<<"First"<<endl;}
        A(int a){cout<<"Second"<<endl;}
        A(int a, int b){cout<<"Third"<<endl;}
};
class B: protected A
{
public:
        A obj1, obj2;
        B(): obj2(9), A(2,2){cout<<"Fourth"<<endl;}
};

int main()
{
        B obj;
        return 0;
}
```

Third
First
Second
Fourth

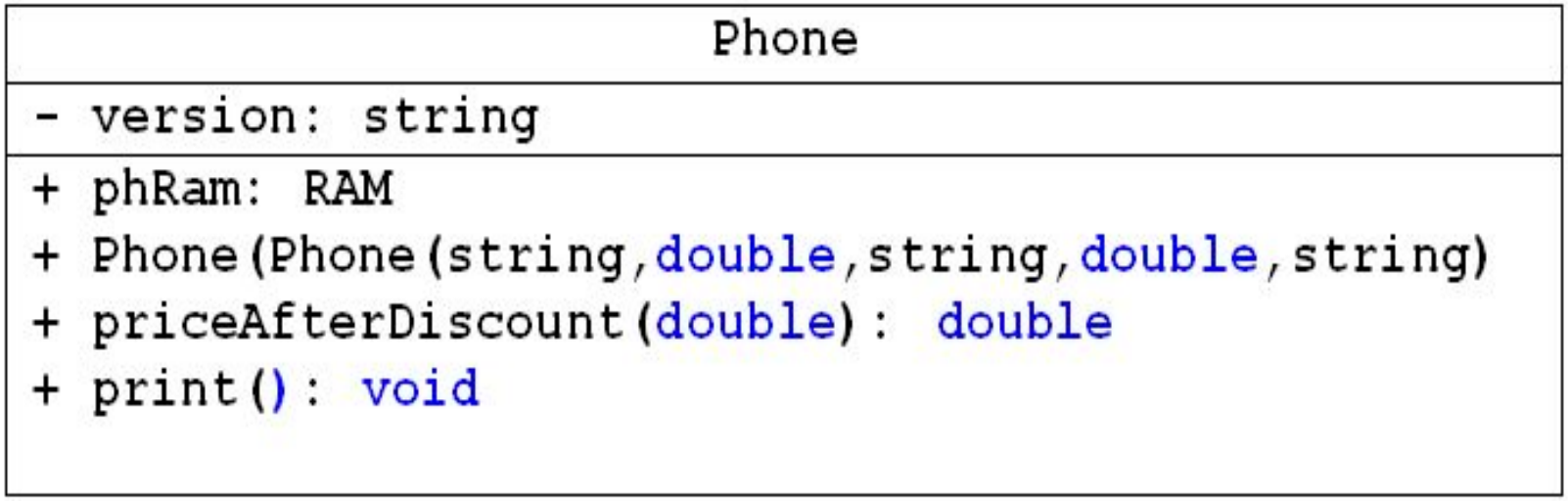
WRITING CODE

Consider the following two classes and answer the questions below

```
class Device{
      string company;
      double price;
  protected:
       double getPrice()
       { return price; }
  public:
    Device(string c)
      { company = c; }
    void setPrice(double p)
      {  price = p; }
    void print()
      { cout<<"Company:  "<<company<<endl;
      cout<<"Price:  "<<price<<endl;}};
```

```
class RAM{
      string id;
      double size;
 public:
  RAM(string i,double s)
      { id = i;   size = s;}
  void print()
  { cout<<"RAM id:  "<<id<<endl;
cout<<"RAM size:  "<<size<<endl;}
};
```

1) Write the header (the definition) of the new **class Phone** which ***publically inherits*** the properties of the class **Device** with the following additional members as shown in the below UML class diagram. (**Don't** write the implementation of the member functions inside the header)



```
class Phone:public Device {
      string version;
public:
      RAM phRam;
      Phone(string ,double ,string ,double ,string);
      double priceAfterDiscount(double x);
      void print();
};
```

2) Write the implementation of the Constructor function (**outside the class**) that takes five parameters to set the company and the price for the **Device**, the id and the size for the phRam and the version for the Phone.

```
Phone::Phone(string a,double b,string c,double d,string e):
 Device(a), phRam(c,d)
{    setPrice(b);
      version = e;
}
```

3) Write the implementation of **priceAfterDiscount** function that receives one double value which represents the discount for the phone and return the price after the discount.

```
double Phone::priceAfterDiscount(double x)
     {
         return getPrice()- x; }
```

4) Write the implementation of the **print** function (outside the class) to print the company and the price for the **Device**, the id and the size for the phRam and the version for the Phone.

```
void Phone::print()
{ Device::print();
  phRam.print();
  cout<<"Version: "<<version<<endl;

}
```