Java$^{\text{TM}}$ Education & Technology Services

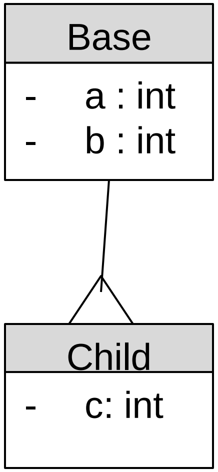
Object Oriented programming Using

C++

Class Relations

Class Relations

IV. Inheritance

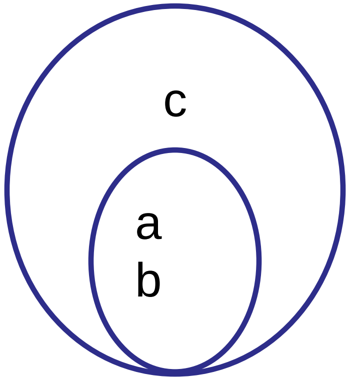


Base

- a : int
- b : int

Child

- c: int



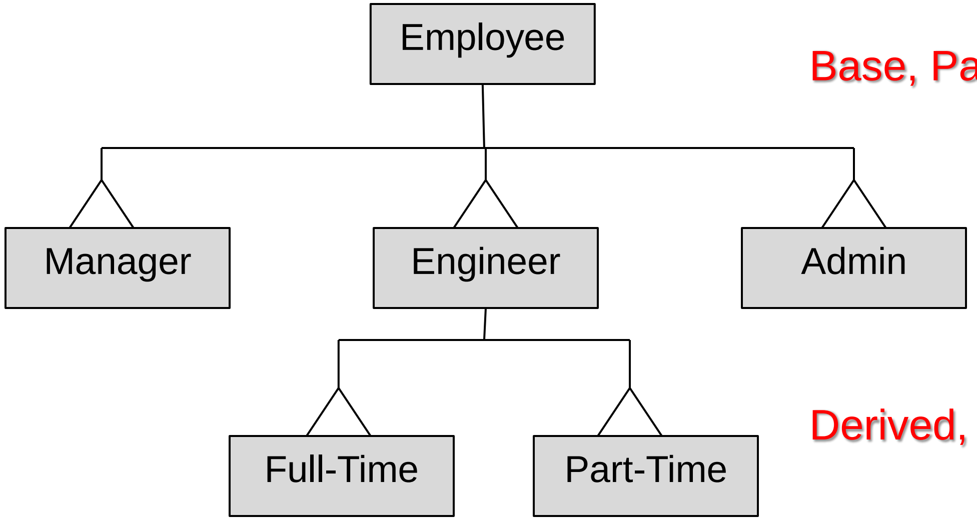
Child obj;

- It is “is-a” or “kind of”.
- It is to extend the functionality of a class.
- Used in reusability of a code.
- Add something new to the Base.
- The child class inherits all members of the parent. But it differs in the accessibility to them.
- Creating object from child = creating object from base & object from child
- Creating object from child = constructor of base is calling then constructor of child called.
- Private variables in base class = can’t be accessed from child

Class Relations

IV. Inheritance

- **Examples:**
  - car is vehicle, bus is vehicle

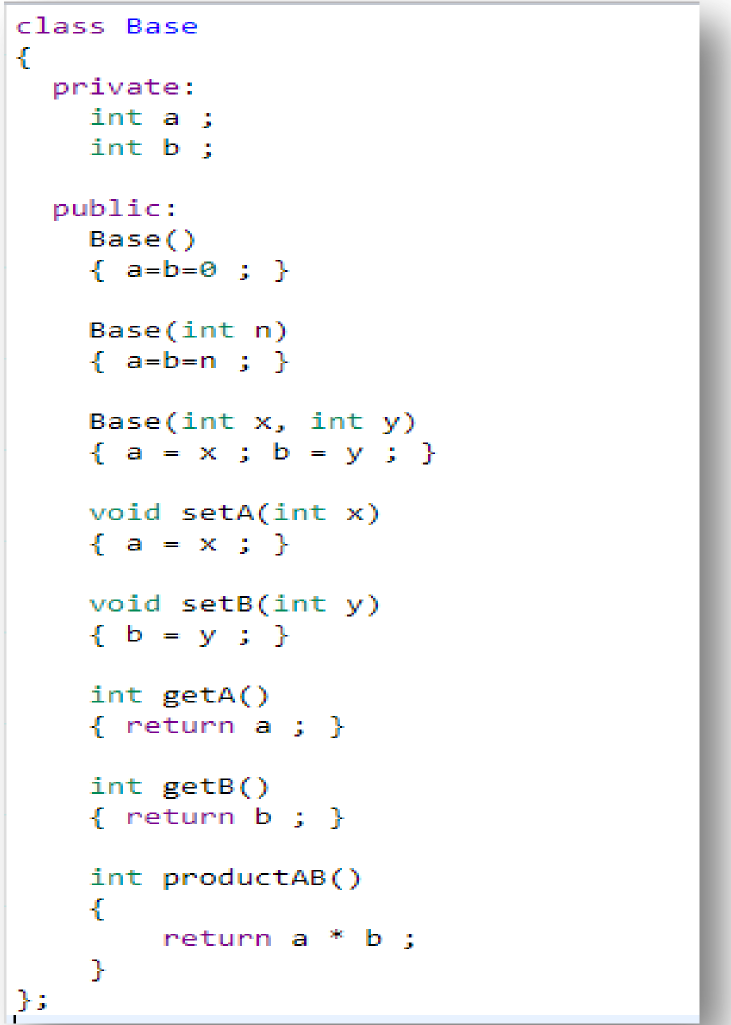


Base, Parent , Super

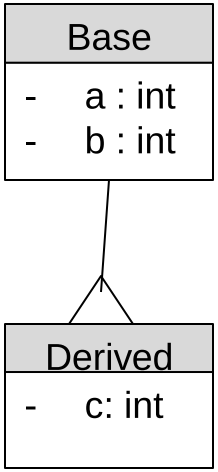
Derived, child, sub

Class Relations

IV. Inheritance

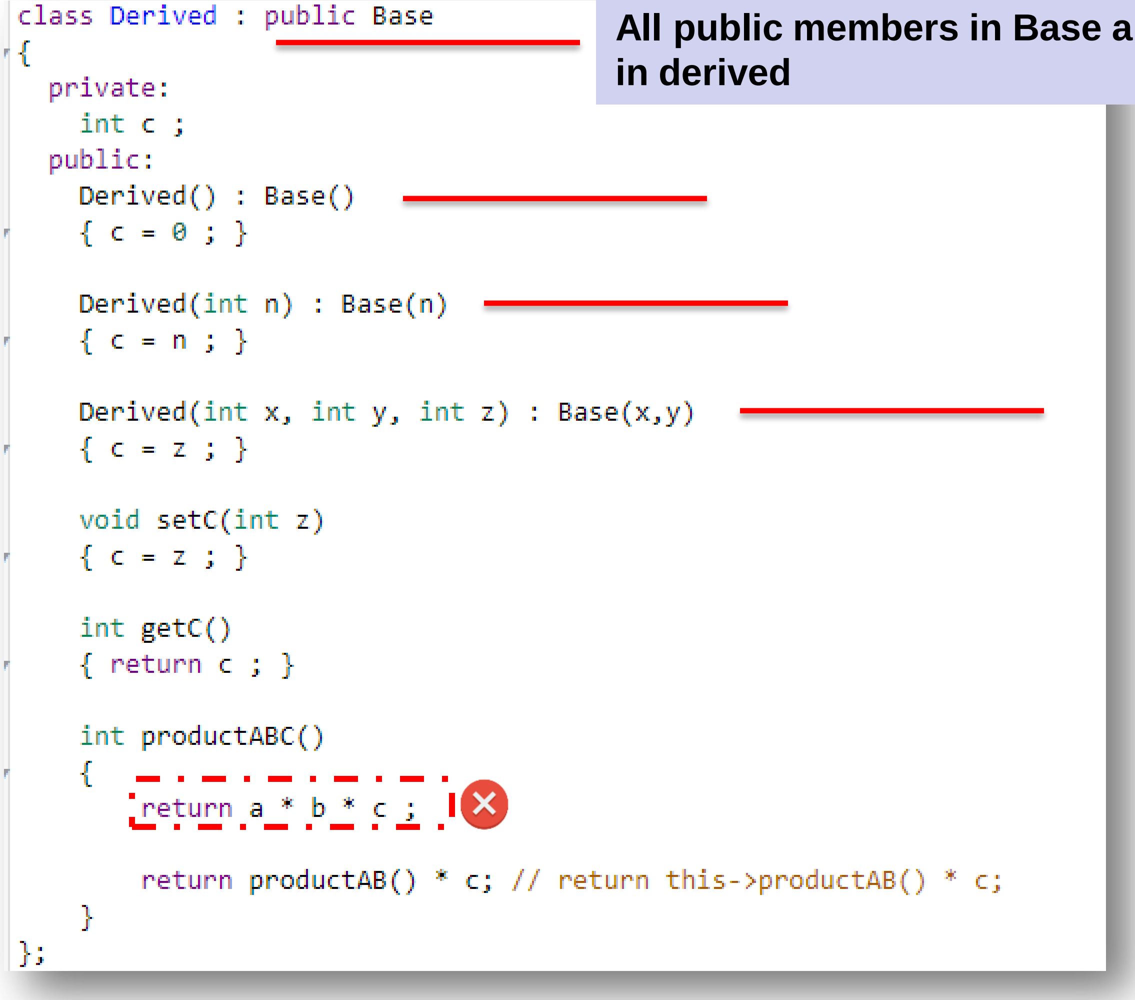


```
class Base
{
  private:
    int a ;
    int b ;

  public:
    Base()
    { a=b=0 ; }

    Base(int n)
    { a=b=n ; }

    Base(int x, int y)
    { a = x ; b = y ; }

    void setA(int x)
    { a = x ; }

    void setB(int y)
    { b = y ; }

    int getA()
    { return a ; }

    int getB()
    { return b ; }

    int productAB()
    {
        return a * b ;
    }
};
```



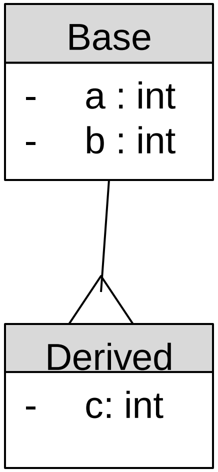
Class Relations

IV. Inheritance



```
class Derived : public Base
{
  private:
    int c ;
  public:
    Derived() : Base()
    { c = 0 ; }

    Derived(int n) : Base(n)
    { c = n ; }

    Derived(int x, int y, int z) : Base(x,y)
    { c = z ; }

    void setC(int z)
    { c = z ; }

    int getC()
    { return c ; }

    int productABC()
    {
        return a * b * c ;

        return productAB() * c; // return this->productAB() * c;
    }
};
```

All public members in Base are public in derived



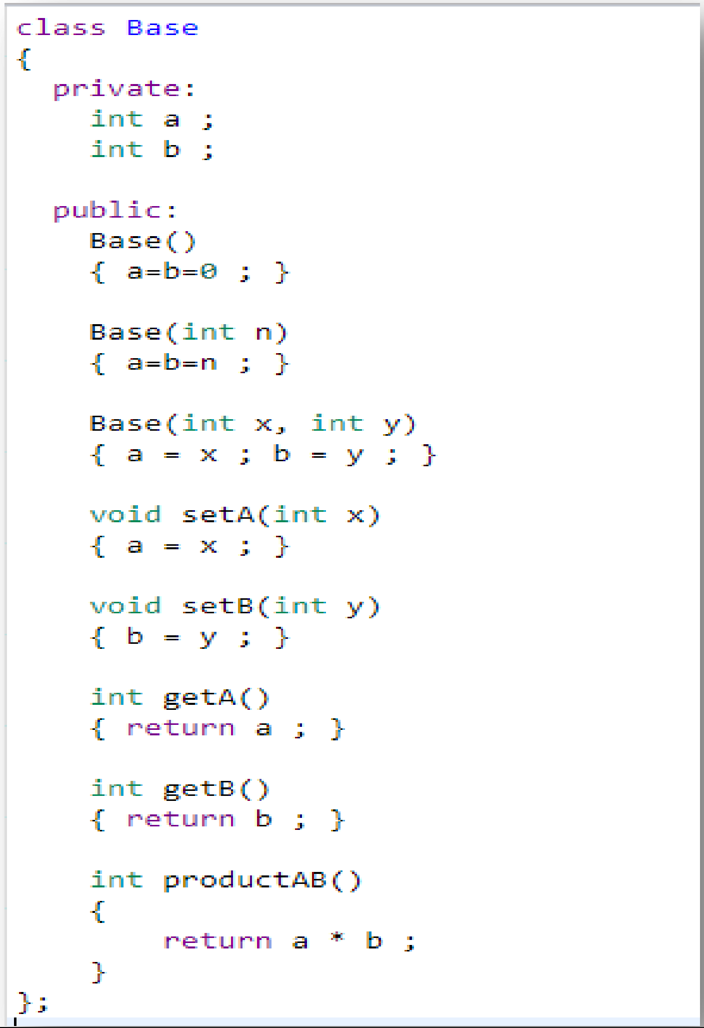
Class Relations

IV. Inheritance

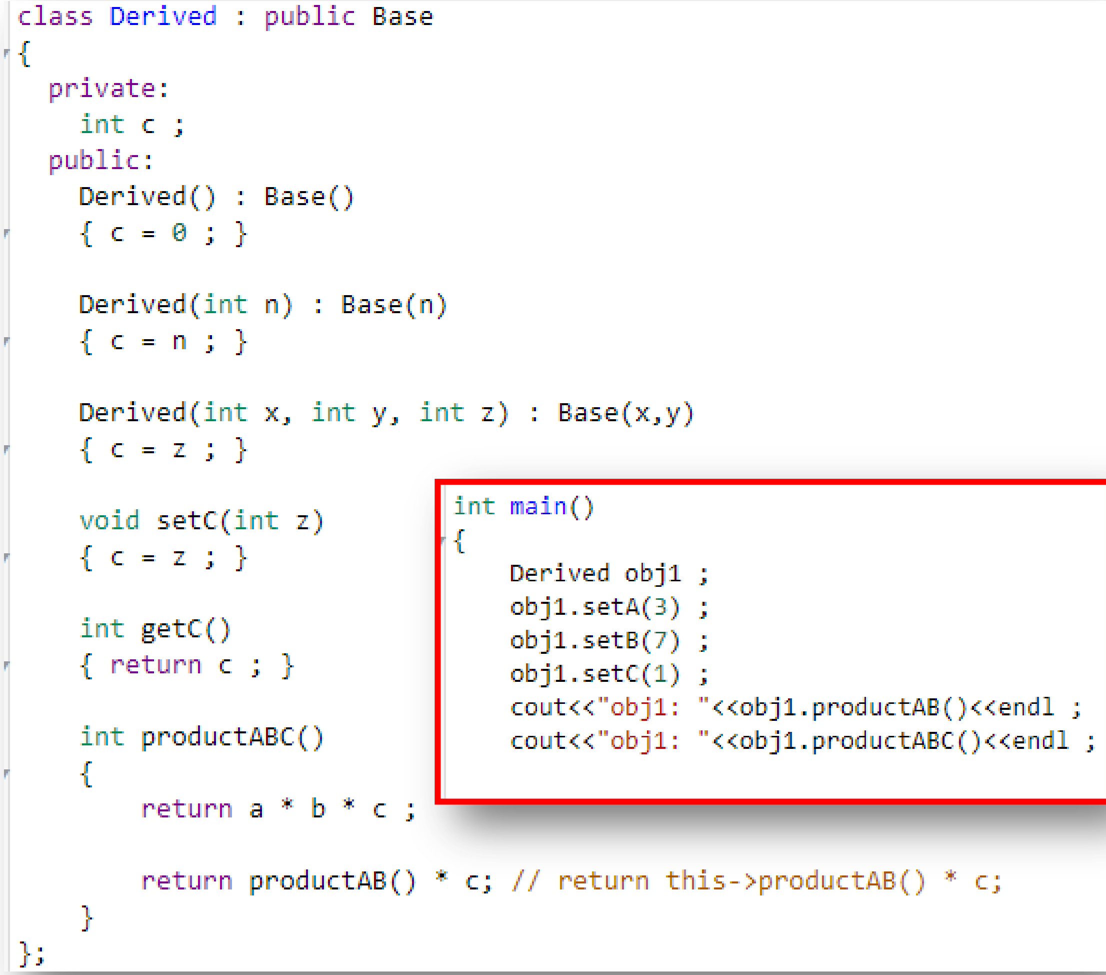
```
Manager:: Manager( ) : Employee() , b1(3), b2(4,3) {
:
:
}

// parent Employee constructor and then parts b1 and b2 constructor and
finally Manager Constructor body
```

Class Relations

IV. Inheritance



```
class Base
{
   private:
      int a ;
      int b ;

   public:
      Base()
      { a=b=0 ; }

      Base(int n)
      { a=b=n ; }

      Base(int x, int y)
      { a = x ; b = y ; }

      void setA(int x)
      { a = x ; }

      void setB(int y)
      { b = y ; }

      int getA()
      { return a ; }

      int getB()
      { return b ; }

      int productAB()
      {
         return a * b ;
      }
};
```

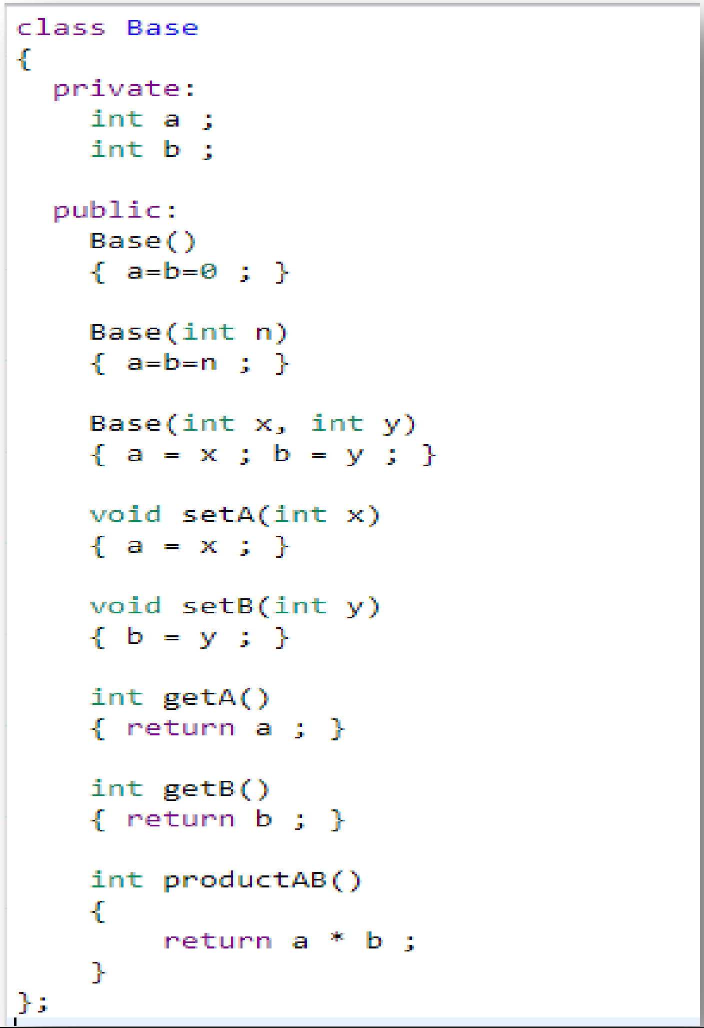


```
class Derived : public Base
{
  private:
    int c ;
  public:
    Derived() : Base()
    { c = 0 ; }

    Derived(int n) : Base(n)
    { c = n ; }

    Derived(int x, int y, int z) : Base(x,y)
    { c = z ; }

    void setC(int z)
    { c = z ; }

    int getC()
    { return c ; }

    int productABC()
    {
        return a * b * c ;

        return productAB() * c; // return this->productAB() * c;
    }
};
```

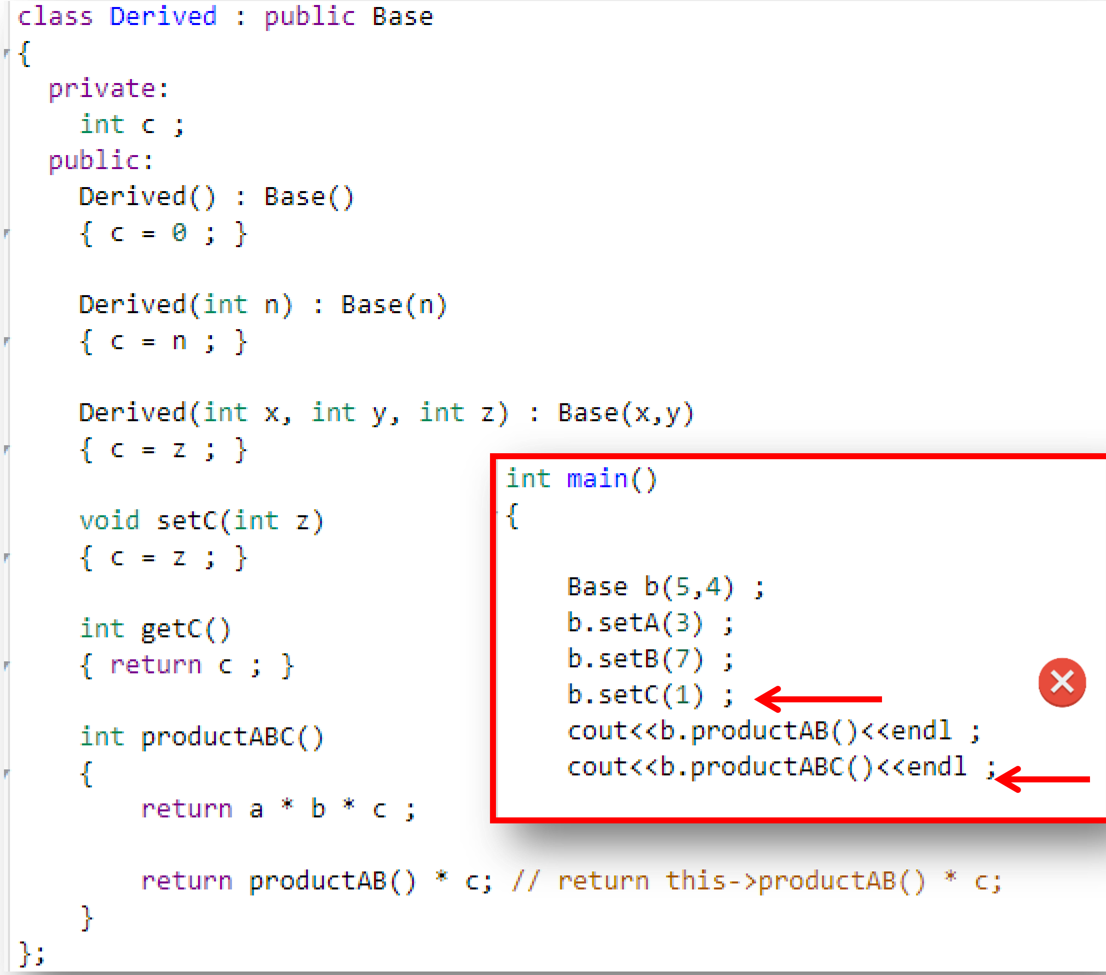
```
int main()
{
     Derived obj1 ;
     obj1.setA(3) ;
     obj1.setB(7) ;
     obj1.setC(1) ;
     cout<<"obj1: "<<obj1.productAB()<<endl ;
     cout<<"obj1: "<<obj1.productABC()<<endl ;
```

Class Relations

IV. Inheritance



```
class Base
{
  private:
    int a ;
    int b ;

  public:
    Base()
    { a=b=0 ; }

    Base(int n)
    { a=b=n ; }

    Base(int x, int y)
    { a = x ; b = y ; }

    void setA(int x)
    { a = x ; }

    void setB(int y)
    { b = y ; }

    int getA()
    { return a ; }

    int getB()
    { return b ; }

    int productAB()
    {
        return a * b ;
    }
};
```

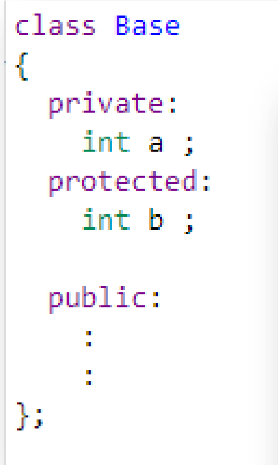


```
class Derived : public Base
{
  private:
    int c ;
  public:
    Derived() : Base()
    { c = 0 ; }

    Derived(int n) : Base(n)
    { c = n ; }

    Derived(int x, int y, int z) : Base(x,y)
    { c = z ; }

    void setC(int z)
    { c = z ; }

    int getC()
    { return c ; }

    int productABC()
    {
        return a * b * c ;

        return productAB() * c; // return this->productAB() * c;
    }
};
```

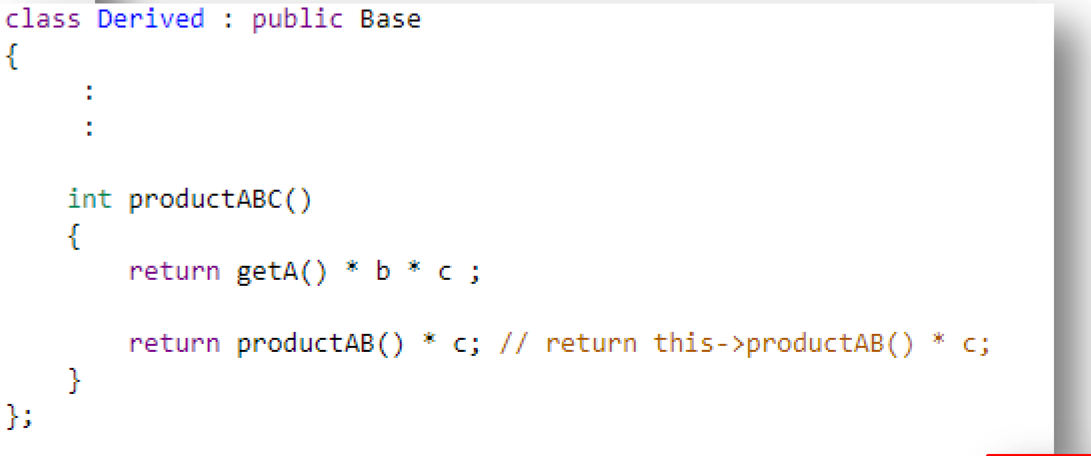
```
int main()
{

    Base b(5,4) ;
    b.setA(3) ;
    b.setB(7) ;
    b.setC(1) ;
    cout<<b.productAB()<<endl ;
    cout<<b.productABC()<<endl ;
```

Class Relations

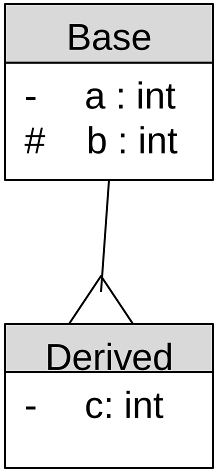
IV. Inheritance [Protected Access Specifier]

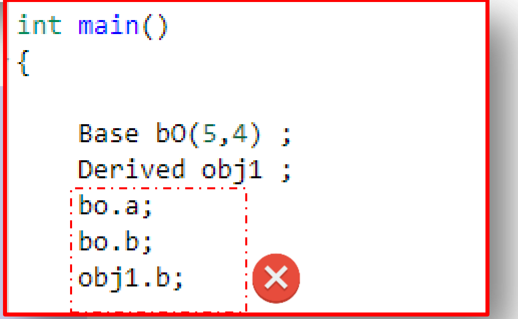


```
class Base
{
   private:
      int a ;
   protected:
      int b ;

   public:
      :
      :
};
```



```
class Derived : public Base
{
       :
       :

     int productABC()
     {
         return getA() * b * c ;

         return productAB() * c; // return this->productAB() * c;
     }
};
```





```
int main()
{

     Base b0(5,4) ;
     Derived obj1 ;
     bo.a;
     bo.b;
     obj1.b;
```

- In Derived, because of public inheritance :
  - all Base public members become public in derived.
  - all Base protected members become protected in derived.
- In Main: only public member of any class can be accessed.

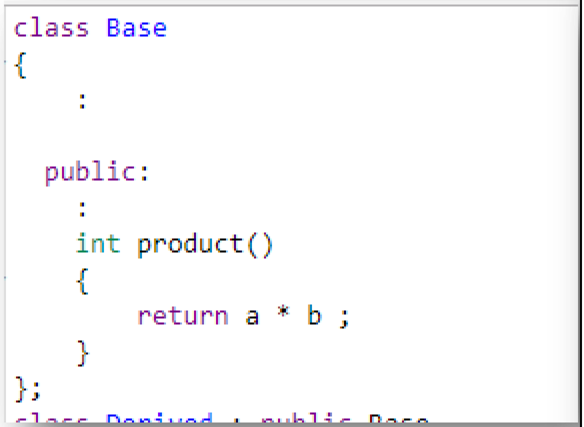
Polymorphism

- Function Overloading :
  - Many functions in one class with the same name but with different function signatures

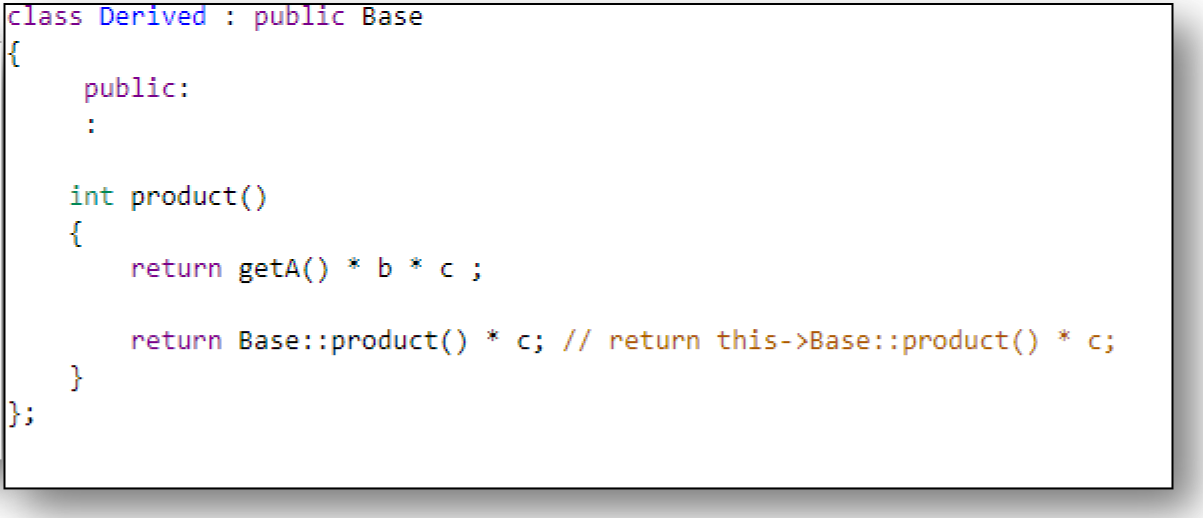
Polymorphism

– Function Overriding :

- Among functions in inheritance tree. [Not in the same class]
- Derived class has the same Base function (the same signature) but with different implementation.
- when run the code, it looks in caller class first for the function and then in its Base class.
- Can increase the accessibility not worth [from protected to public]



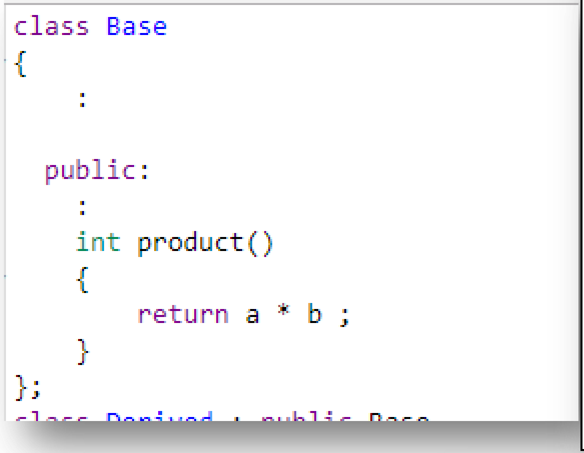
```
class Base
{
    :

  public:
    :
    int product()
    {
        return a * b ;
    }
};
class Derived : public Base
```



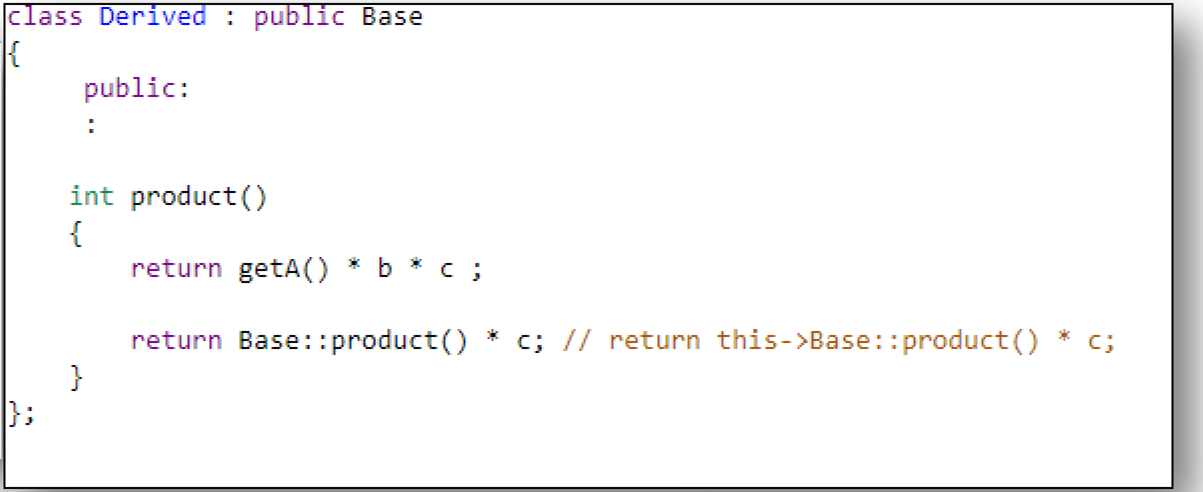
```
class Derived : public Base
{
     public:
      :

    int product()
    {
        return getA() * b * c ;

        return Base::product() * c; // return this->Base::product() * c;
    }
};
```

Polymorphism

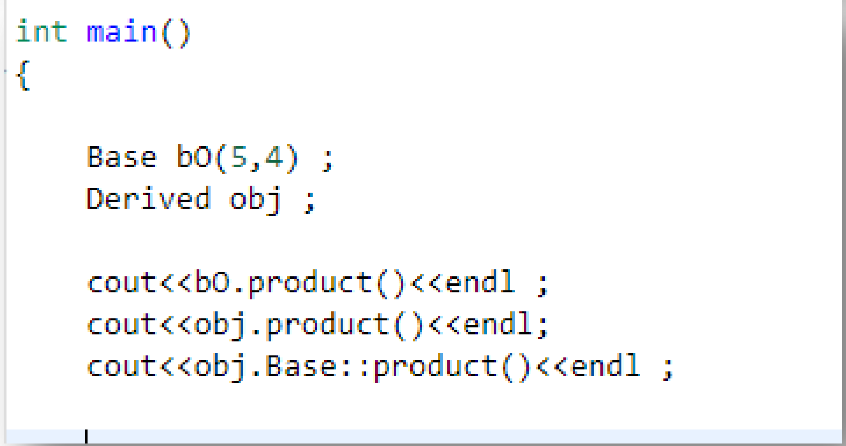
– Function Overriding :



```
class Base
{
    :

  public:
    :
    int product()
    {
        return a * b ;
    }
};
class Derived : public Base
```



```
class Derived : public Base
{
     public:
     :

    int product()
    {
        return getA() * b * c ;

        return Base::product() * c; // return this->Base::product() * c;
    }
};
```

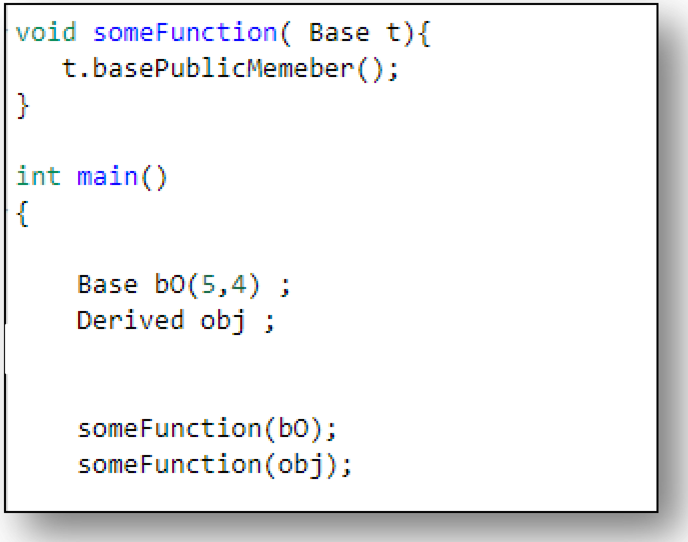


```
int main()
{

    Base b0(5,4) ;
    Derived obj ;

    cout<<b0.product()<<endl ;
    cout<<obj.product()<<endl;
    cout<<obj.Base::product()<<endl ;
```

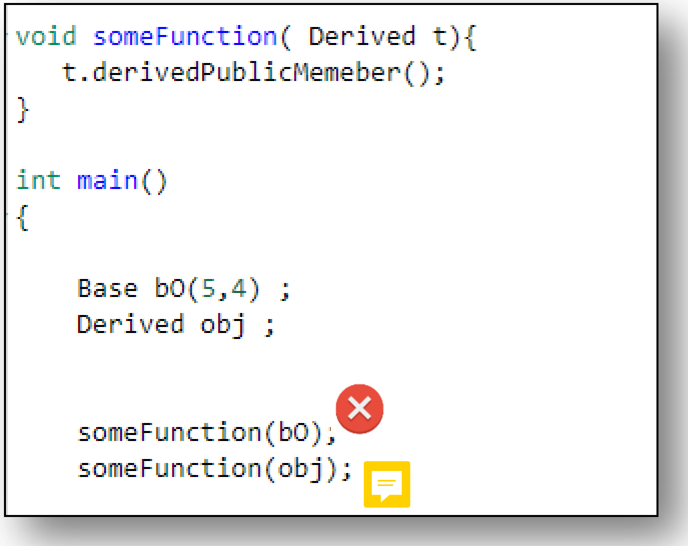
Polymorphism

– Function Overriding :

- If function takes a Base Type the Base or Derived object can be sent to it.



```
void someFunction( Base t){
   t.basePublicMemeber();
}

int main()
{

    Base b0(5,4) ;
    Derived obj ;


    someFunction(b0);
    someFunction(obj);
```

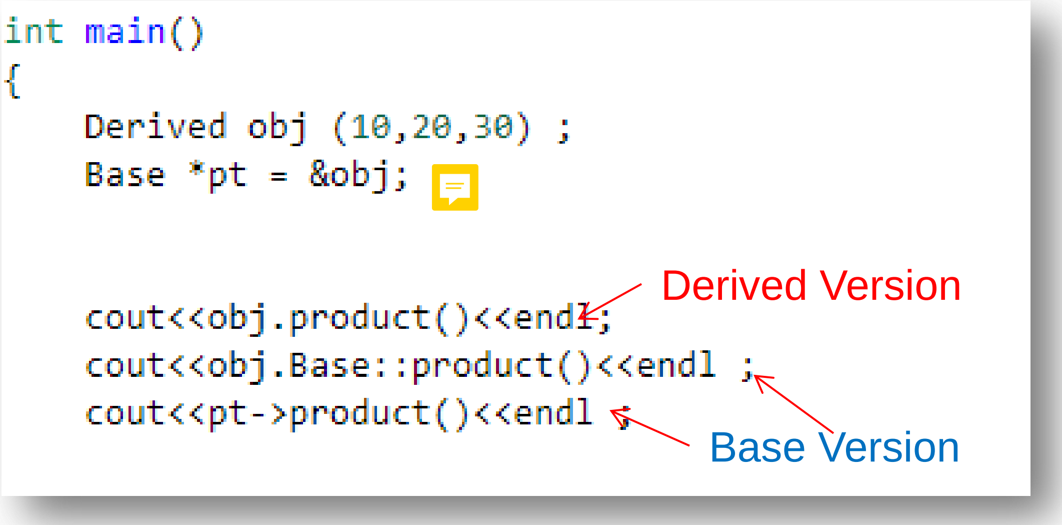


```
void someFunction( Derived t){
   t.derivedPublicMemeber();
}

int main()
{

    Base b0(5,4) ;
    Derived obj ;


    someFunction(b0);✗
    someFunction(obj);
```

- If function takes a Derived Type the only Derived object can be sent to it.

Polymorphism

– Function Overriding :



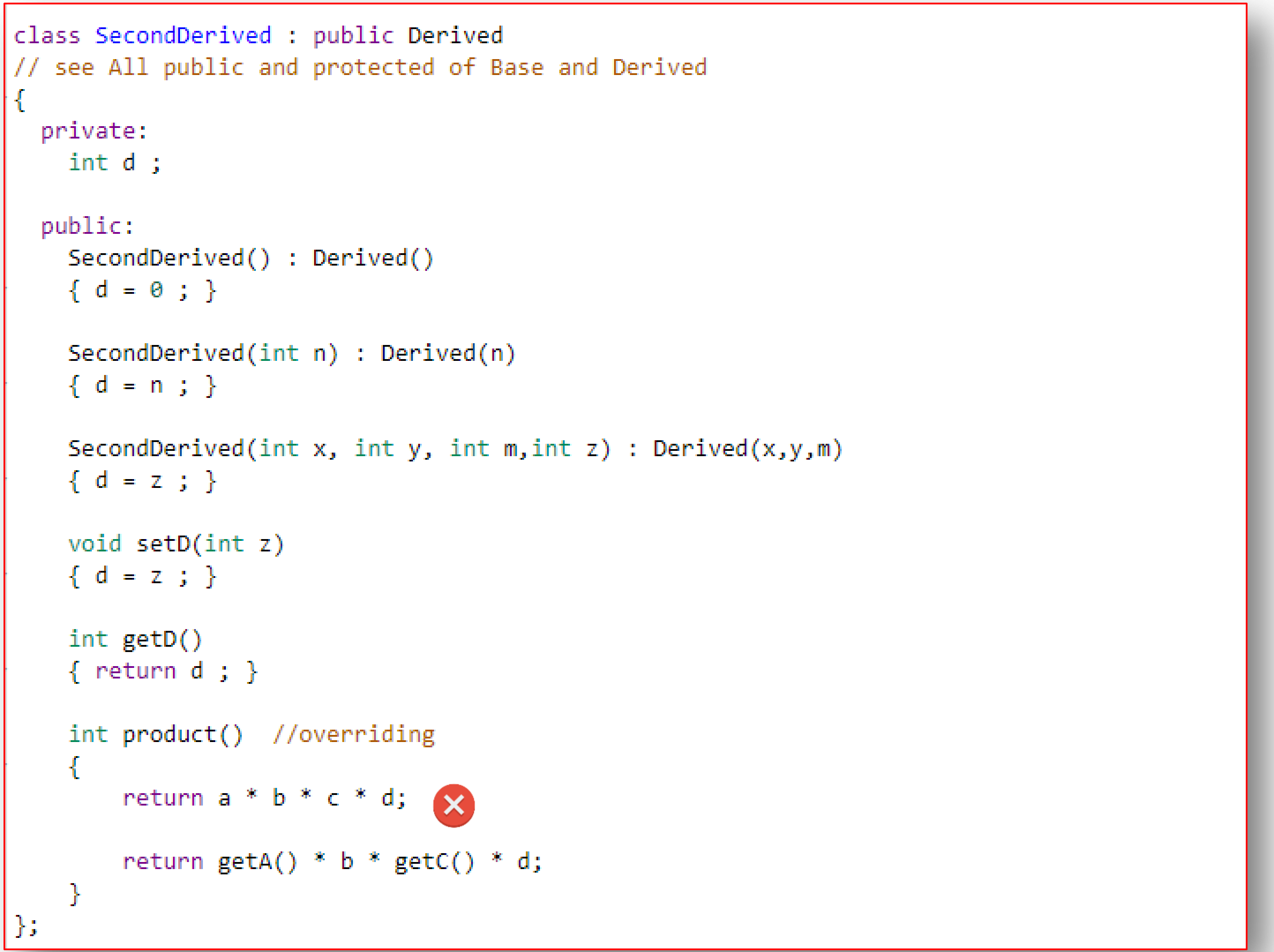
```
int main()
{
    Derived obj (10,20,30) ;
    Base *pt = &obj;


    cout<<obj.product()<<endl;
    cout<<obj.Base::product()<<endl ;
    cout<<pt->product()<<endl ;
```

Derived Version

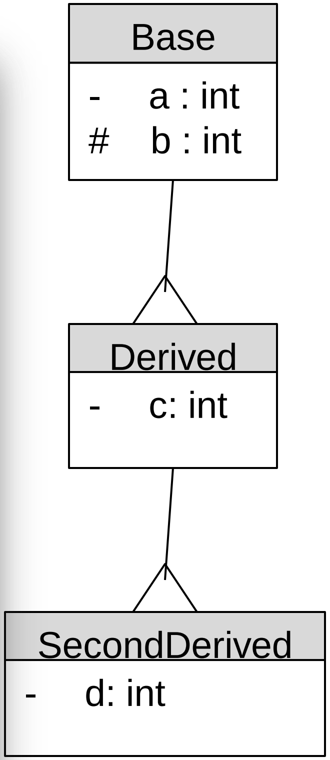
Base Version

Polymorphism

– Add new level to the tree:



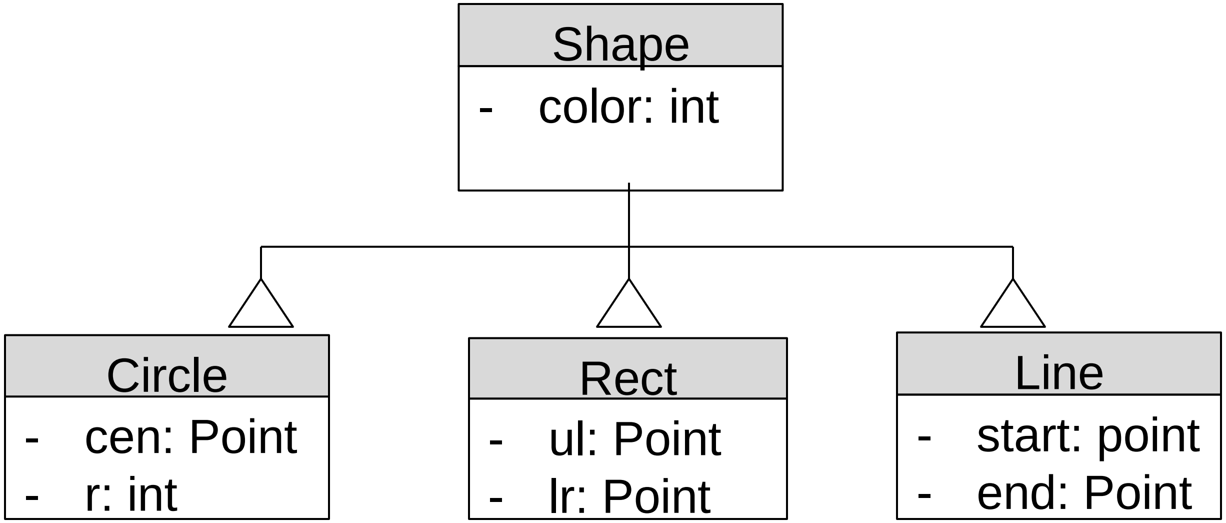
```
class SecondDerived : public Derived
// see All public and protected of Base and Derived
{
  private:
    int d ;

  public:
    SecondDerived() : Derived()
    { d = 0 ; }

    SecondDerived(int n) : Derived(n)
    { d = n ; }

    SecondDerived(int x, int y, int m,int z) : Derived(x,y,m)
    { d = z ; }

    void setD(int z)
    { d = z ; }

    int getD()
    { return d ; }

    int product()  //overriding
    {
        return a * b * c * d; ✖

        return getA() * b * getC() * d;
    }
};
```



Lab Exercise

Lab Exercise

- 1st Assignment :
  - Base, Derived, and SecondDerived [Test all cases at the main].
  - Update picture application by add class Shape as a parent to lines , Rectangles and circles with color attribute.


Shape
- color: int
Circle
- cen: Point
- r: int
Rect
- ul: Point
- lr: Point
Line
- start: point
- end: Point