

7COM1025

Programming for Software Engineers

Lecture 14

INHERITANCE

```
#include <iostream>
#include <string>
using namespace std;
class TwoDShape{
public:
    double width;
    double height;
    void showDim(){cout<<"Width: "<<width<<" height:
"<<height<<endl;}};
class Triangle : public TwoDShape{
public:
    string style;
    double area(){
        return width * height / 2.0;}
    void showStyle(){cout<<"Triangle is "<<style<<endl;}};
int main(){
    Triangle t1;
    t1.width=4.0; t1.height=4.0;
    t1.style="isosceles";
    t1.showStyle();
    t1.showDim();
    return 0;
}
```

TwoDShape = Base class

Triangle = derived class

class Triangle : public TwoDShape

Means that all public members of the base class will also be public members of the derived class.

You could use private so the public members of the base class will be private members of the derived class.

Derived classes cannot access base class private members

PROTECTED MEMBERS

```
#include <iostream>
#include <string>
using namespace std;
class B{
protected:
    int i, j;//private to B, but accessible in derived class
public:
    void set(int a, int b){i=a;j=b;}
    void show(){cout<<i<<" "<<j<<endl;}
};
class D: public B{
    int k;
public:
    void setk(){k=i*j;}//D can access i and j because they are protected, not private
    void showk(){cout<<k<<endl;}
};
int main(){
    D ob;
    ob.set(2,3);
    ob.show();
    ob.setk();
    ob.showk();
    return 0;
}
```

class D: public B{

If you use protected in the above:
all public and protected members of the
base class become protected members of
the derived class.

PROBLEM 14.1

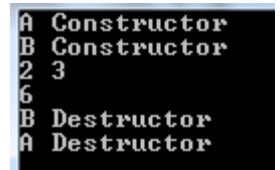
Using the TwoDShape class as base, substitute the Triangle class by a Rectangle class.
Your class should have:
A method to calculate the area.
Getters and setters for all member variables.

CONSTRUCTORS AND INHERITANCE

```
#include <iostream>
#include <string>
using namespace std;
class A{
protected:
    int i, j;
public:
    A(){cout<<"A Constructor"<<endl;}
    ~A(){cout<<"A Destructor"<<endl;}
    void set(int a, int b){i=a;j=b;}
    void show(){cout<<i<<" "<<j<<endl;}
};
```

```
class B: public A{
    int k;
public:
    B(int p,int q){i=p;j=q; cout<<"B Constructor"<<endl;}
    ~B(){cout<<"B Destructor"<<endl;}
    void setk(){k=i*j;}
    void showk(){cout<<k<<endl;}
};

int main(){
    B ob(2,3);
    ob.show();
    ob.setk();
    ob.showk();
    return 0;
}
```



```
A Constructor
B Constructor
2 3
B Destructor
A Destructor
```

CONSTRUCTORS WITH PARAMETERS

```
#include <iostream>
#include <string>
using namespace std;
class TwoDShape{
    double width, height;
public:
    TwoDShape(double w, double h)
        {width =w; height=h;}
    void showDim()
    {cout<<"Width: "<<width<<" height: "<<height<<endl;}
    double getWidth(){return width;}
    double getHeight(){return height;}
    void setWidth(double w){width=w;}
    void setHeight(double h){height=h;}
};
```

```
class Triangle : public TwoDShape{
    string style;
public:
    Triangle(string str, double w, double h): TwoDShape(w,h)
    {style=str;}
    double area()
        {return getWidth() * getHeight() / 2.0;}
    void showStyle()
        {cout<<"Triangle is "<<style<<endl;}
};

int main(){
    Triangle t1("isosceles", 4.0,4.0);
    t1.showStyle();
    t1.showDim();
    cout<<"Area: "<<t1.area()<<endl;
    return 0;
}
```

PROBLEM 14.2

Update your code for the previous problem so that width and height can be passed via a constructor.

POINTERS TO DERIVED TYPES

Normally a pointer of one type cannot point to an object of a different type.

Base class pointers and derived objects are an exception to this rule.

A base class pointer can also be used to point to an object of any derived class!

```
B *p;  
B B_ob;  
D D_ob;  
p=&B_ob;  
p=&D_ob;
```

(In the above B is the base class and D is the derived class)

Similarly, a base class reference can be used to refer to an object of a derived type.

VIRTUAL FUNCTIONS

They are used to implement polymorphism.

Virtual functions can be redefined in one or more derived classes

Each derived class can have its own version of a virtual function!

Classes with virtual functions are called polymorphic classes

A virtual function in the derived class must have the definition as in the base class (data types/quantities)

You can have virtual destructors (but not constructors)

Polymorphism only happens when a virtual function is accessed through a pointer (or reference)

VIRTUAL FUNCTION EXAMPLE

```
#include <iostream>
#include <string>
using namespace std;
class B{
public:
    virtual void who(){
        cout<<"Base"<<endl;}};
class D1:public B{
public:
    void who(){
        cout<<"First Derivation"<<endl;}};
class D2:public B{
public:
    void who(){
        cout<<"Second derivation"<<endl;}};
```

```
int main(){
    B base_obj;
    B *p;
    D1 D1_obj;
    D2 D2_obj;
    p=&base_obj;
    p->who();
    p=&D1_obj;
    p->who();
    p=&D2_obj;
    p->who();
    return 0;
}
```

VIRTUAL FUNCTIONS CAN BE INHERITED

```
#include <iostream>
#include <string>
using namespace std;
class B{
public:
    virtual void who(){
        cout<<"Base"<<endl;}};
class D1:public B{
public:
    void who(){
        cout<<"First Derivation"<<endl;}};
class D2:public B{};
//Who is not defined
```

```
int main(){
    B base_obj;
    B *p;
    D1 D1_obj;
    D2 D2_obj;
    p=&base_obj;
    p->who();
    p=&D1_obj;
    p->who();
    p=&D2_obj;
    p->who();
    return 0;
}
```

PURE VIRTUAL FUNCTIONS AND ABSTRACT CLASSES

A pure virtual function is declared in the base class but has no definition (ie. no code between {}).

These should be defined in the derived classes only.

```
virtual type name(parameter-list)=0;
```

Abstract classes are those that only declare pure virtual function.

They declare only a generalised form that will be shared by all its derived classes.

However, each derived class should define the implementation.

PROBLEM 14.3

Update your code so that it has three classes: The twoDShape base class, your Rectangle class, and the Triangle class.