Programming in Modern C++

Module M28: Polymorphism: Part 3: Abstract Base Class

Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

ppd@cse.iitkgp.ac.in

All url's in this module have been accessed in September, 2021 and found to be functional

Module Recap

- Discussed Static and Dynamic Binding
- Polymorphic type introduced

Module Objectives

- Understand why destructor must be `virtual` in a class hierarchy
- Learn to work with class hierarchy

Module Outline

1. Virtual Destructor
   - Slicing
2. Pure Virtual Function
3. Abstract Base Class
   - Shape Hierarchy
     - Pure Virtual Function with Body
4. Module Summary

Virtual Destructor

Virtual Destructor

Virtual Destructor

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

class B { int data_; public:
    B(int d) :data_(d) { cout << "B()" << endl; }
    ~B() { cout << "~B()" << endl; }
     virtual void Print() { cout << data_; }
};
class D: public B { int *ptr_; public:
    D(int d1, int d2) :B(d1), ptr_(new int(d2)) { cout << "D()" << endl; }
    ~D() { cout << "~D()" << endl; delete ptr_; }
    void Print() { B::Print();  cout << " " << *ptr_; }
};
int main() {
    B *p = new B(2);
    B *q = new D(3, 5);

    p->Print(); cout << endl;
    q->Print(); cout << endl;

    delete p;
    delete q;
}
```

```
Output:
B()
B()
D()
2
3 5
~B()
~B()
```

Destructor of d (type D) not called!

Virtual Destructor

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

class B { int data_; public:
    B(int d) :data_(d) { cout << "B()" << endl; }
    virtual ~B() { cout << "~B()" << endl; }        // Destructor made virtual
    virtual void Print() { cout << data_; }
};
class D: public B { int *ptr_; public:
    D(int d1, int d2) :B(d1), ptr_(new int(d2)) { cout << "D()" << endl; }
    ~D() { cout << "~D()" << endl; delete ptr_; }
    void Print() { B::Print();  cout << " " << *ptr_; }
};
int main() {
    B *p = new B(2);
    B *q = new D(3, 5);

    p->Print(); cout << endl;
    q->Print(); cout << endl;

    delete p;
    delete q;
}
```

```
Output:
B()
B()
D()
2
3 5
~B()
~D()
~B()
```

Destructor of d (type D) is called!

Virtual Destructor: Slicing

- Slicing is where we assign an object of a derived class to an instance of a base class, thereby losing part of the information - some of it is sliced away

```
#include <iostream>
using namespace std;
class Base { protected: int i; public:
    Base(int a)      i = a;
    virtual void display() { cout << "I am Base class object, i = " << i << endl; }
};
class Derived : public Base { int j; public:
    Derived(int a, int b) : Base(a) { j = b; }
    virtual void display() { cout<< "I am Derived class object, i = " << i << ", j = " << j <<endl; }
};
// Global method, Base class object is passed by value
void somefunc (Base obj) { obj.display(); }
int main() { Base b(33); Derived d(45, 54);
    somefunc(b);
    somefunc(d);  // Object Slicing, the member j of d is sliced off
}
```

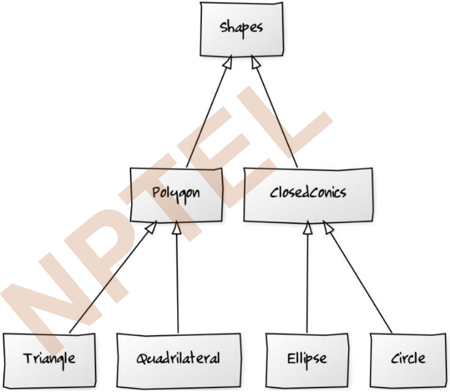
```
I am Base class object, i = 33
I am Base class object, i = 45
```

- If the destructor is not `virtual` in a polymorphic hierarchy, it leads to Slicing
- **Destructor must be declared `virtual` in the base class**

Pure Virtual Function

Pure Virtual Function

Hierarchy of Shapes



- We want to have a polymorphic `draw()` function for the hierarchy
- `draw()` will be overridden in every class based on the drawing algorithms
- What is the `draw()` function for the root `Shapes` class?

Pure Virtual Function

- For the polymorphic hierarchy of Shapes, we need draw() to be a virtual function
- draw() must be a member of Shapes class for polymorphic dispatch to work
- But we cannot define the body of draw() function for the root Shapes class as we do not have an algorithm to draw an arbitrary share. In fact, we cannot even have a representation for shapes in general!
- Pure Virtual Function solves the problem
- A Pure Virtual Function has a signature but no body!
- Example:

```
class Root { public:
    void f();              // Non-Virtual Function
    virtual void g();      // Virtual Function
    virtual void h() = 0; // Pure Virtual Function
};
```

Abstract Base Class

Abstract Base Class

Abstract Base Class

- A class containing at least one Pure Virtual Function is called an Abstract Base Class
- Pure Virtual Functions may be inherited or defined in the class
- No instance can be created for an Abstract Base Class
- Naturally it may not have a constructor or a `virtual` destructor
- An Abstract Base Class, however, may have other `virtual` (non-pure) and non-`virtual` member functions as well as data members
- Data members in an Abstract Base Class should be protected. Of course, private and public data are also allowed
- Member functions in an Abstract Base Class should be public. Of course, private and protected methods are also allowed
- A Concrete Class must override and implement all Pure Virtual Functions so that it can be instantiated

Shape Hierarchy

```
#include <iostream>  // Abstract Base Class shown in red
using namespace std; // Concrete Class shown in green

class Shapes { public:                                        // Abstract Base Class
    virtual void draw() = 0; // Pure Virtual Function
};
class Polygon: public Shapes { public: void draw() { cout<< "Polygon: Draw by Triangulation" <<endl; } };
class ClosedConics: public Shapes { public:                   // Abstract Base Class
    // draw() inherited - Pure Virtual
};
class Triangle: public Polygon { public: void draw() { cout << "Triangle: Draw by Lines" << endl; } };
class Quadrilateral: public Polygon { public:
    void draw() { cout << "Quadrilateral: Draw by Lines" << endl; }
};
class Circle: public ClosedConics { public:
    void draw() { cout << "Circle: Draw by Breshenham Algorithm" << endl; }
};
class Ellipse: public ClosedConics { public: void draw() { cout << "Ellipse: Draw by ..." << endl; } };
int main() {
    Shapes *arr[] = { new Triangle, new Quadrilateral, new Circle, new Ellipse };

    for (int i = 0; i < sizeof(arr) / sizeof(Shapes *); ++i)
        arr[i]->draw();
    // ...
}
```

Shape Hierarchy

```
int main() {
    Shapes *arr[] = { new Triangle, new Quadrilateral, new Circle, new Ellipse };

    for (int i = 0; i < sizeof(arr) / sizeof(Shapes *); ++i)
        arr[i]->draw();
    // ...
    return 0;
}
```

```
Triangle: Draw by Lines
Quadrilateral: Draw by Lines
Circle: Draw by Breshenham Algorithm
Ellipse: Draw by ...
```

- Instances for class Shapes and class ClosedConics cannot be created

Shape Hierarchy: A Pure Virtual Function may have a body!

```
#include <iostream>
using namespace std;
class Shapes { public:                                  // Abstract Base Class
    virtual void draw() = 0 // Pure Virtual Function
    { cout << "Shapes: Init Brush" << endl; }
};
class Polygon: public Shapes { public:                  // Concrete Class
    void draw() { Shapes::draw(); cout << "Polygon: Draw by Triangulation" << endl; }
};
class ClosedConics: public Shapes { public:             // Abstract Base Class
    // draw() inherited - Pure Virtual
};
class Triangle: public Polygon { public:                // Concrete Class
    void draw() { Shapes::draw(); cout << "Triangle: Draw by Lines" << endl; }
};
class Quadrilateral: public Polygon { public:           // Concrete Class
    void draw() { Shapes::draw(); cout << "Quadrilateral: Draw by Lines" << endl; }
};
class Circle: public ClosedConics { public:             // Concrete Class
    void draw() { Shapes::draw(); cout << "Circle: Draw by Breshenham Algorithm" << endl; }
};
class Ellipse: public ClosedConics { public:            // Concrete Class
    void draw() { Shapes::draw(); cout << "Ellipse: Draw by ..." << endl; }
};
```

Shape Hierarchy

```
int main() {
    Shapes *arr[] = { new Triangle, new Quadrilateral, new Circle, new Ellipse };

    for (int i = 0; i < sizeof(arr) / sizeof(Shapes *); ++i)
        arr[i]->draw();
}
```

```
Shapes: Init Brush
Triangle: Draw by Lines
Shapes: Init Brush
Quadrilateral: Draw by Lines
Shapes: Init Brush
Circle: Draw by Breshenham Algorithm
Shapes: Init Brush
Ellipse: Draw by ...
```

- Instances for class Shapes and class ClosedConics cannot be created
- Some compilers do not allow to inline the function body for a pure virtual function

```
class Shapes { public: virtual void draw() = 0 { cout << "Shapes: Init Brush" << endl; } };
```

Outline the function body:

```
class Shapes { public: virtual void draw() = 0; };
void Shapes::draw() { cout << "Shapes: Init Brush" << endl; }
```

Module Summary

- Discussed why destructors must be `virtual` in a polymorphic hierarchy
- Introduced Pure Virtual Functions
- Introduced Abstract Base Class