Module 28: Programming in C++

Dynamic Binding (Polymorphism): Part 3

Partha Pratim Das

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

ppd@cse.iitkgp.ernet.in

Tanwi Mallick
Srijoni Majumdar
Himadri B G S Bhuyan

Module Objectives

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

Module Outline

- Virtual Destructor
- Pure Virtual Function
- Abstract Base Class

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;

    return 0;
}
```

```
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;

    return 0;
}
```

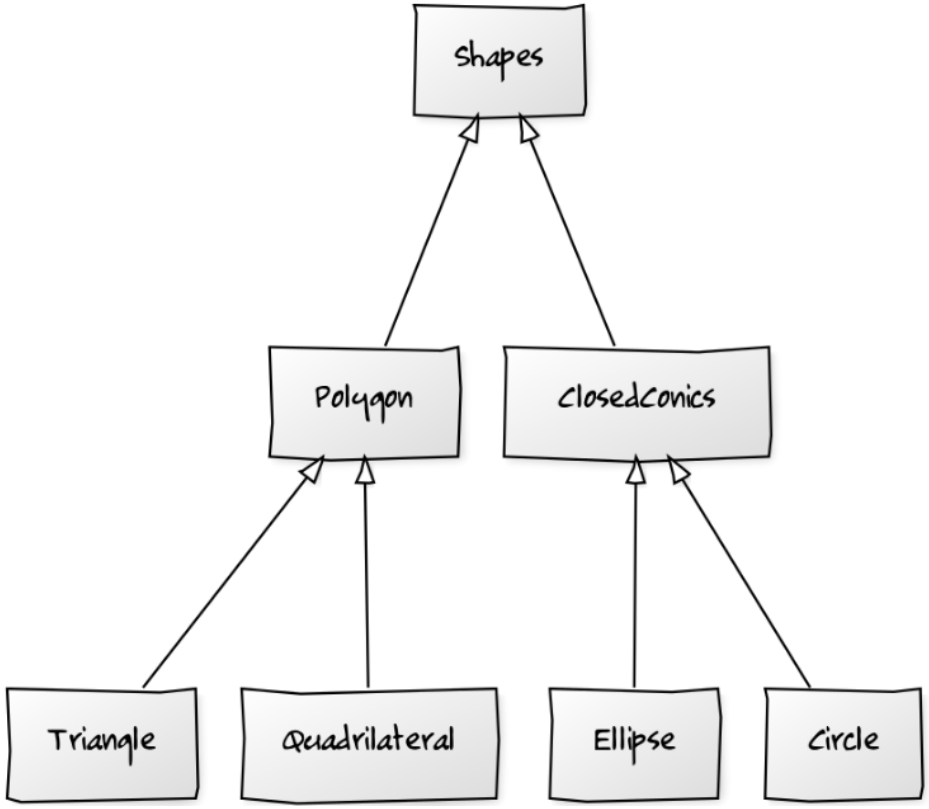
```
Output:
B()
B()
D()
2
3 5
~B()
~D()
~B()

Destructor of d (type D) is called!
```

Virtual Destructor

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

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!

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 does not have a constructor or the 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>
using namespace std;

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

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;
}
-----
Output:

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 {                          // Concrete Class
    void draw() { Shapes::draw(); cout << "Polygon: Draw by Triangulation" << endl; }
};
class ClosedConics : public Shapes {                    // 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; }
};
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;
}
```

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;
}
-----
Output:

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

Module Summary

- Discussed why destructors must be virtual
- Introduced Pure Virtual Functions
- Introduced Abstract Base Class

Instructor and TAs

Name	Mail	Mobile
Partha Pratim Das, *Instructor*	ppd@cse.iitkgp.ernet.in	9830030880
Tanwi Mallick, *TA*	tanwimallick@gmail.com	9674277774
Srijoni Majumdar, *TA*	majumdarsrijoni@gmail.com	9674474267
Himadri B G S Bhuyan, *TA*	himadribhuyan@gmail.com	9438911655