

Виртуальные функции и полиморфизм

Тема 6

Полиморфизм -

это возможность для объектов разных классов, связанных с помощью наследования, реагировать различным образом при обращении к одной и той же функции-элементу.

Говорят, что полиморфизм - это **один интерфейс и множество методов**

Полиморфизм в C++

- ▶ Наследование (public)
- ▶ Виртуальные функции

Виртуальные функции

Интерфейсные функции. Их можно переопределить в каждом производном классе

```
virtual void draw() const;
```

Реализуют **позднее** связывание

Функция, объявленная виртуальной в базовом классе, остается виртуальной до конца иерархии

Полиморфный класс - класс, в котором есть хотя бы одна виртуальная функция

Абстрактный класс -

класс, объекты которого не могут быть созданы

Чтобы класс стал абстрактным, в нем должна быть хотя бы одна **чистая виртуальная** функция

```
virtual void draw() const = 0;
```

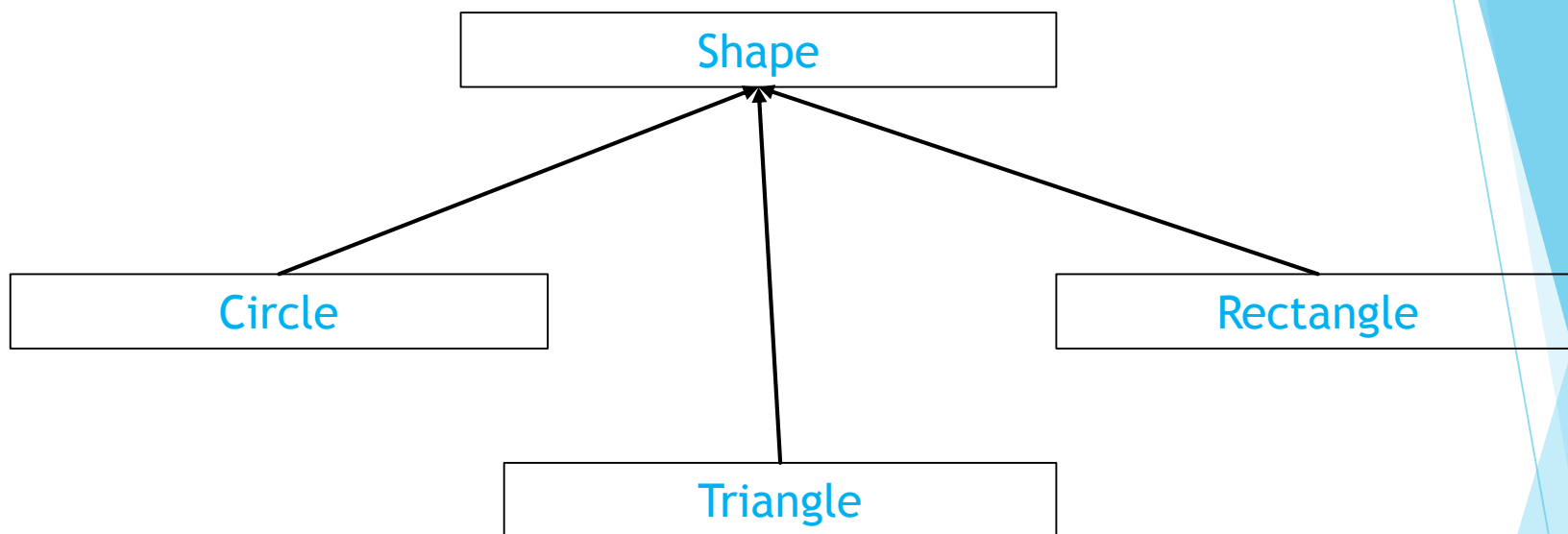
Чистая виртуальная функция - это функция, тело которой равно 0.

Если в производном классе не переопределена хотя бы одна чистая виртуальная функция, класс останется абстрактным

Виртуальный деструктор

Если у класса имеются виртуальные функции, то классу необходимо создать **виртуальный деструктор**. В этом случае, если объект в иерархии уничтожен явным использованием операции **delete**, примененной к указателю базового класса на объект производного класса, то будет вызван деструктор соответствующего класса

Учебный пример



Класс Shape

```
class Shape
{
public:
    Shape() ;
    virtual ~Shape() ;
    virtual void Print()const =0;
    virtual double Area()const = 0;
    virtual double Perimetr()const =0;
};
```


Определение функций класса Shape

```
Shape::Shape ()  
{  
    std::cout << "Shape constructor" << std::endl;  
}  
Shape::~~Shape ()  
{  
    std::cout << "Shape destructor" << std::endl;  
}
```

Класс Circle

```
class Circle: public Shape {  
public:  
    Circle(double) ;  
    virtual ~Circle() ;  
    virtual void Print() const;  
    virtual double Area() const;  
    virtual double Perimetr() const;  
private:  
    double radius;  
};
```

Определение функций класса Circle

```
Circle::Circle(double rad):radius(rad) {  
    std::cout << "Circle constructor" << std::endl;}
```

```
Circle::~~Circle() {  
    std::cout << "Circle destructor" << std::endl; }
```

```
void Circle::Print() const {  
    std::cout << "*****Circle*****" << std::endl;  
    std::cout << "radius = " << radius << std::endl;  
    std::cout << "Length of circle = " << Perimetr() << std::endl;  
    std::cout << "Area = " << Area() << std::endl;  
    std::cout << "*****" << std::endl;}
```

```
double Circle::Area() const {  
    return M_PI * pow(radius, 2);  
}
```

```
double Circle::Perimetr() const {  
    return 2 * M_PI * radius;  
}
```

Для работы с M_PI необходимо определить макрос и включить библиотеку (она нужна для и функции pow)

```
#define _USE_MATH_DEFINES  
#include <math.h>
```

Класс Rectangle

```
class Rectangle: public Shape {  
public:  
    Rectangle(double, double);  
    virtual ~Rectangle();  
    virtual void Print() const;  
    virtual double Area() const;  
    virtual double Perimetr() const;  
private:  
    double a, b;  
};
```

Определение функций класса Rectangle

```
double Rectangle::Area() const {return a*b;}
```

Определение функций класса Rectangle

```
Rectangle::Rectangle(double _a, double _b):a(_a), b(_b){  
    std::cout << "Rectangle constructor" << std::endl; }
```

```
Rectangle::~~Rectangle(){  
    std::cout << "Rectangle destructor" << std::endl; }
```

```
void Rectangle::Print() const{  
    std::cout << "*****Rectangle*****" << std::endl;  
    std::cout << "a = " << a << std::endl;  
    std::cout << "b = " << b << std::endl;  
    std::cout << "Perimetr = " << Perimetr() << std::endl;  
    std::cout << "Area = " << Area() << std::endl;  
    std::cout << "*****" << std::endl; }
```

```
double Rectangle::Area() const {  
    return a * b;  
}
```

```
double Rectangle::Perimetr() const {  
    return 2 * (a + b);  
}
```

Класс Triangle

```
class Triangle: public Shape
{
public:
    Triangle(double, double, double);
    virtual ~Triangle();
    virtual void Print() const;
    virtual double Area() const;
    virtual double Perimetr() const;
private:
    double a,b,c;
};
```

Определение функций класса Triangle

```
Triangle::Triangle(double _a, double _b, double _c):a(_a), b(_b), c(_c) {  
    std::cout << "Triangle constructor" << std::endl;  
}
```

```
Triangle::~Triangle(){  
    std::cout << "Triangle destructor" << std::endl;  
}
```

```
void Triangle::Print() const {  
    std::cout << "*****Triangle*****" << std::endl;  
    std::cout << "a = " << a << std::endl;  
    std::cout << "b = " << b << std::endl;  
    std::cout << "c = " << c << std::endl;  
    std::cout << "Perimetr = " << Perimetr() << std::endl;  
    std::cout << "Area = " << Area() << std::endl;  
    std::cout << "*****" << std::endl; }  

```

```
double Triangle::Area() const {  
    double per = Perimetr() / 2;  
    return sqrt(per * (per - a) * (per - b) * (per - c));  
}
```

```
double Triangle::Perimetr() const {  
    return a + b + c;  
}
```


Функция main

```
int main()
{
    Shape sh;
    const int SIZE = 3;
    Shape* p[SIZE];
    p[0] = new Circle(5.2);
    p[1] = new Triangle(2.3,3.4,4.1);
    p[2] = new Rectangle(4.5, 5.4);
    for(int i =0 ; i < SIZE; i++)
    {
        p[i]->Print();
        delete p[i];
    }
    return 0;
}
```

Ошибка!

Нельзя создать объект абстрактного класса!

Хорошо!

Указатель на абстрактный класс создать можно

Вызов виртуальной функции

Для такого уничтожения нужен виртуальный деструктор

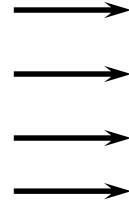
Таблица виртуальных функций

Объект класса Circle

vtbl*
radius

vtbl:

1	*
2	*
3	*
4	*



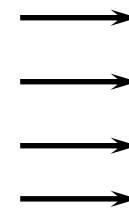
Circle::Print()
Circle::Area()
Circle::Perimetr()
Circle::~~Circle()

Объект класса Triangle

vtbl*
a
b
b

vtbl:

1	*
2	*
3	*
4	*



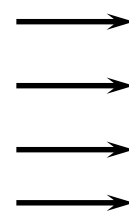
Triangle::Print()
Triangle::Area()
Triangle::Perimetr()
Triangle::~~Circle()

Объект класса Rectangle

vtbl*
a
b

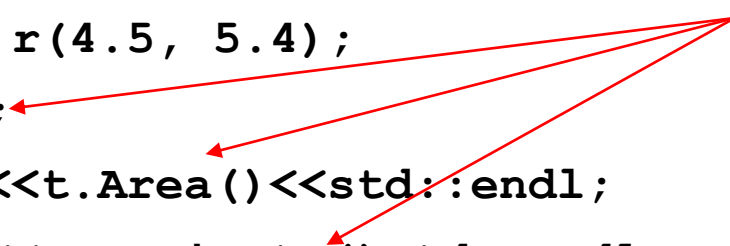
vtbl:

1	*
2	*
3	*
4	*



Rectangle::Print()
Rectangle::Area()
Rectangle::Perimetr()
Rectangle::~~Circle()

```
Circle c(5.2);  
Triangle t(2.3,3.4,4.1);  
Rectangle r(4.5, 5.4);  
c.Print();  
std::cout<<t.Area()<<std::endl;  
std::cout<<r.Perimetr()<<std::endl;
```

Three red arrows originate from a text box on the right and point to the function calls in the code: 'c.Print()', 't.Area()', and 'r.Perimetr()'.

- Arrow 1: Points from the text box to 'c.Print()'.
- Arrow 2: Points from the text box to 't.Area()'.
- Arrow 3: Points from the text box to 'r.Perimetr()'.

При вызове функций
через объект
виртуальность не
работает

```
class Shape
{
    virtual void Print()const = 0;
    virtual double Area()const = 0;
    virtual double Perimetr()const = 0;
public:
    Shape();
    virtual ~Shape();
    void PrintInfo()const
    {
        Print();
    }
};
```

Виртуальные
функции могут
быть закрытыми

`this->Print()`

В производном классе
переопределяем
виртуальные функции

```
class Circle : public Shape {  
    virtual void Print() const;  
    virtual double Area() const;  
    virtual double Perimetr() const;  
public:  
    Circle(double) ;  
    virtual ~Circle() ;  
private:  
    double radius;  
};
```

```
int main()
{
    const int n = 3;
    Shape* p[n];
    p[0] = new Circle(5.2);
    p[1] = new Triangle(2.3, 3.4, 4.1);
    p[2] = new Rectangle(4.5, 5.4);
    for (int i = 0; i < n; i++)
    {
        p[i]->PrintInfo();
        delete p[i];
    }
    return 0;
}
```

Вызов неvirtуальной
функции приводит к
полиморфному
поведению программы

Возможные ошибки в переопределении виртуальных функций

```
class Shape
{
public:
    virtual void Print()const =0;
};

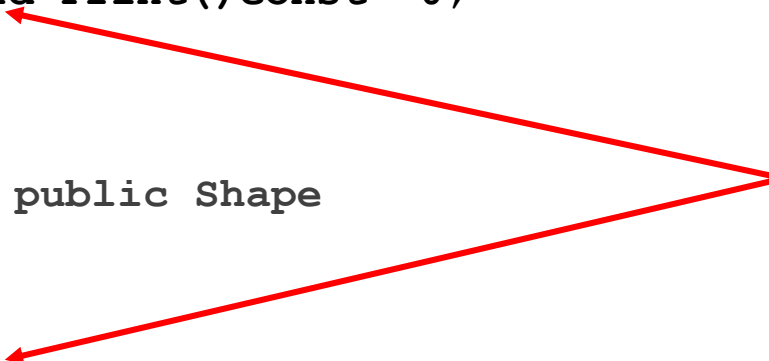
class Circle : public Shape
{
public:
    virtual void Print();
};
```

Плохо!!!
const участвует в
переопределении
виртуальных
функций

Возможные ошибки в переопределении виртуальных функций

```
class Shape
{
public:
    virtual void Print() const =0;
};

class Circle : public Shape
{
public:
    virtual bool Print() const;
};
```



Плохо!!!
Тип
возвращаемого
значения
участвует в
переопределении
виртуальных
функций

Возможные ошибки в переопределении виртуальных функций

```
class Shape
{
public:
    void Print() const;
};

class Circle : public Shape
{
public:
    virtual void Print() const;
};
```

Плохо!!!
Функция в базовом
классе
невиртуальная

Устранение ошибок при работе с виртуальными функциями

Виртуальные функции рекомендуется переопределять со словом **override**

Для запрета дальнейшего переопределения используют слово **final**

```
class Shape
{
public:
    void Print() const;
    virtual double Area() const = 0;
    virtual double Perimetr() const = 0;
};
```

```
class Circle: public Shape{
public:
    virtual void Print() const override;
    virtual double Area() override;
    virtual int Perimetr() const override;
private:
    double radius;
};
```

Ошибка на этапе
компиляции

Виртуальные функции в конструкторах и деструкторах

```
class A {  
public:  
    A() { f(); }  
    virtual ~A() { f(); }  
    virtual void f()const{  
        std::cout << "A::f()"  
                    << std::endl;  
    }  
};
```

```
class B:public A {  
public:  
    B() { f(); }  
    virtual ~B(){ f(); }  
    virtual void f()const override {  
        std::cout << "B::f()"  
                    << std::endl;  
    }  
};
```

Вывод программы:

```
int main() {  
    A a;  
    B b;  
    A *p = new B;  
    p->f();  
    delete p;}  
}
```

```
A::f()  
A::f()  
B::f()  
A::f()  
B::f()  
B::f()  
B::f()  
A::f()  
B::f()  
A::f()  
A::f()  
A::f()
```

Деструктор объекта b

Деструктор объекта a

«Виртуальные конструкторы»

```
class A {  
public:  
    virtual A* new_A() { return new A; }  
    virtual A* clone() { return new A(*this); }  
};
```

```
class B: public A {  
public:  
    virtual B* new_A() override { return new B; }  
    virtual B* clone() override { return new B(*this); }  
};
```

```
int main(){  
    A* a[4];  
    a[0] = new A();  
    a[1] = new B();  
    for (int i = 2; i < 4; i++)  
        a[i] = a[i - 2]->clone();  
    ...  
}
```

Конец