

Тема 11) Полиморфизм: статично и

динамично свързване. Виртуални функции.

Ключови думи - override, final, абстрактен клас.

Виртуални гадинки

Полиморфизъм

- ~~1 тип~~ много функционалност

I статично свързване (compile time polymorphism)

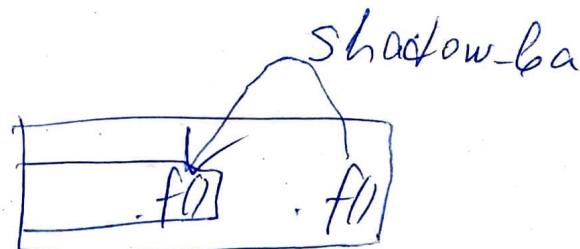
- избора на функция се прави в компилацията (предварително)
- определение на това на * или ref *)

1) function overloading

Пример: operators

$3 + 7$

"ABC" + "DCE"



Пример: одесен

class Base

f() { ... }

class Der:Base

f() { ... }

Der f;

d. f(); // Der::f()

Base b = &d;
b.f(); // Base::f()

(1)

II динамично свързване (Runtime Polymorphism)

- избора на обект се прави по време на изпълнение на програмата (Runtime)
- през програмни обекти

Def виртуална функция

- Една функция, която е дефинирана в базовия клас и преопределена (Override) в класа наследник.

Пример:

Base
Virtual f()

Der: Base
f()

Der d;
d.f(); //Der::f()
Base * b = &d;
b.f(); //Der::f()

Override

- предизвикване на виртуална функция
- уведомление за developer (носащ гримки)

Диаграма:



A Пример:

Base
{
 virt f();
 Base() {f();}
 ~Base() {f();}
 g() {f();}
}

Der: Base
{
 f() override
}

{
Der d; //Base::f()
d.g(); //Der::f()
Base * ptr = &d;
ptr->g(); //Der::f()
3 //Base::f()

final

- може да се override на virtual ф-ка
- не може да се override-ва наследеното и не.

Пример: struct A

 virtual f()

struct B: A

 f() override

struct C: B

 f() final

struct D

 f() ~~override~~

Compile time

- за касове

struct D final {{}}

D - не може да

бъде наследяван

Абстрактен клас

- клас с няколко чисто (pure) virtual ф-ки
- от него не може да се прибегне към инициализация
- определян за наследяване

Чисто виртуална функция (pure virtual)

- наследника е задължен да го преопределва
- в противен случай в този към ще се предположи B-
абстрактен клас

- може да има/няма имплементация

Пример: class Base

 virtual void f()=0;

Base::f() { ... }

class Der: Base

 void f() override;

Виртуална таблици

- таблици съдържат:

- табла на класа

- указатели към virt таблиците

- преди изпълнение на virt таблица се запълват vtable кодовете за съвпадка

Пример

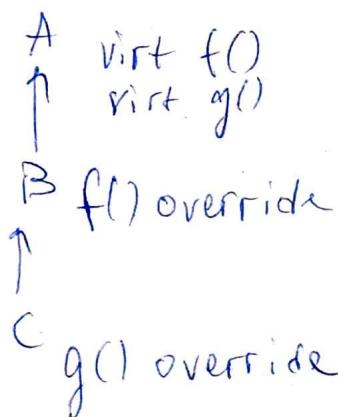


table A

A::f()	A::g()
--------	--------

table B

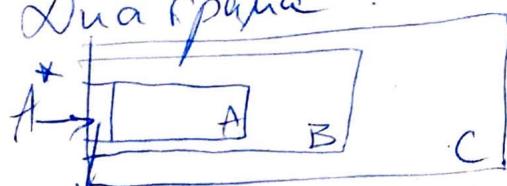
B::f()	B::g()
--------	--------

table C

B::f()	C::g()
--------	--------

коинтентънте
изпълненията на vTables

Два спосба:



vpointer-съм към vtable на конкретния object

vTable C

C::f()	C::g()
--------	--------

ptr → f();

1. дефинираме ptr

2. дефинираме ptr на vpointer

3. напирам функция - трябва да съществува f(B* this), но ptr е от тип A*

4. касъраме ptr като B*

5. изпълняваме бъдещата функция

```
#pragma once
```

```
#include <iostream>
```

```
class Shape
```

```
{ protected:
```

```
    struct point
```

```
{
```

```
        point() = default;
```

```
        point(int x, int y) : x(x), y(y) {}
```

```
        int x = 0;
```

```
        int y = 0;
```

```
        double getDist(const point& other) const
```

```
{    int dx = x - other.x;
```

```
    int dy = y - other.y;
```

```
    return sqrt(dx*dx + dy*dy);
```

```
}
```

```
};
```

```
const point& getPointAtIndex(size_t index) const;
```

```
private:
```

```
    point* points = nullptr;
```

```
    size_t pointsCount = 0;
```

```
    void copyFrom(const Shape& other);
```

```
    void moveFrom(Shape& other);
```

```
    void free();
```

Shape.h

public:

Shape (size_t pointsCount);

Shape (const Shape& other);

Shape (Shape&& other) noexcept;

Shape& operator=(const Shape& other);

Shape& operator=(Shape&& other) noexcept;

virtual ~Shape();

void setPoint(size_t pointIndex, int x, int y);

virtual double getArea() const = 0;

virtual double getPer() const;

virtual bool isPointIn(int x, int y) const = 0;

}

name

Shape . CPP

#include "Shape.h"

```
Shape::Shape (size_t pointsCount) : pointsCount (pointsCount)
{
    points = new point [pointsCount];
}

void Shape::copyFrom (const Shape& other)
{
    points = new point [other.pointsCount];
    for (int i = 0; i < other.pointsCount; i++)
        points [i] = other.points [i];
    pointsCount = other.pointsCount;
}

void Shape::moveFrom (Shape& other)
{
    points = other.points;
    pointsCount = other.pointsCount;
    other.points = nullptr;
    other.pointsCount = 0;
}

void Shape::free ()
{
    delete [] points;
    points = nullptr;
    pointsCount = 0;
}
```

```
Shape::Shape(const Shape& other)
```

```
{ copyFrom(other); }
```

```
}
```

```
Shape::Shape(Shape& &other) noexcept
```

```
{ moveFrom(std::move(other)); }
```

```
}
```

```
Shape & Shape::operator=(const Shape& other)
```

```
{ if(this != &other)
```

```
{ free(); }
```

```
copyFrom(other);
```

```
return *this;
```

```
}
```

```
Shape & Shape::operator=(Shape& &other) noexcept
```

```
{ if(this != &other)
```

```
{ free(); }
```

```
moveFrom(std::move(other));
```

```
return *this;
```

```
}
```

```
Shape::~Shape()
```

```
{ free(); }
```

```
}
```

Shape.cpp

```
const Shape::point & Shape::getPointAtIndex(size_t index) const {  
    if(index >= pointsCount)  
        throw std::out_of_range("Invalid index!");  
    return points[index];  
}  
  
void Shape::setPoint(size_t pointIndex, int x, int y)  
{  
    if(pointIndex >= pointsCount)  
        throw std::out_of_range("Invalid index!");  
    points[pointIndex] = point(x,y);  
}  
  
double Shape::getPer() const  
{  
    assert(points >= 3);  
  
    double per = 0;  
    for(int i = 0; i < pointsCount - 1; i++)  
    {  
        per += points[i].getDist(points[i + 1]);  
    }  
    per += points[size - 1].getDist(points[0]);  
    return per;  
}
```

```
#pragma once
```

Triangle.h

```
#include "Shape.h"
```

```
class Triangle : public Shape
```

```
{
```

```
public:
```

```
Triangle(int x1, int y1, int x2, int y2, int x3, int y3)
```

```
double getArea() const override;
```

```
bool isPointIn(int x, int y) const override;
```

```
} ;
```

```
Triangle::Triangle(int x1, int y1, int x2, int y2, int x3, int y3) : Shape(3)
```

```
{ set SetPoint(0, x1, y1);
```

```
SetPoint(1, x2, y2);
```

```
SetPoint(2, x3, y3);
```

```
}
```

```
double Triangle::getArea() const
```

```
{ const Shape::point& p1 = getPointAtIndex(0);
```

```
const Shape::point& p2 = getPointAtIndex(1);
```

```
const Shape::point& p3 = getPointAtIndex(2);
```

```
return abs(p1.x * p2.y + p2.x * p3.y + p3.x * p1.y - p3.x * p2.y -
```

```
p2.x * p1.y - p1.x * p3.y) / 2.0;
```

```
}
```

Code Triangle::isPointIn (int x, int y) const

{

Shape::point p(x, y);

Triangle t1 (getPointAtIndex(0).x, getPointAtIndex(0).y,
getPointAtIndex(1).x, getPoint(1).y, x, y);

Tr t2 (getPoint(1).x, getPoint(1).y, getPoint(2).x, getPoint(2).y,
x, y);

Tr t3 (getPoint(2).x, getPoint(2).y, getPoint(0).x, getPoint(0).y,
x, y);

return abs(t1.getArea() + t2.getArea() + t3.getArea() -
getArea()) <= 0.000001;

Circle.h

```
#pragma once
```

```
#include "Shape.h"
```

```
class Circle : public Shape
```

```
{
```

```
public:
```

```
Circle(int, int, int);
```

```
double getArea() const override;
```

```
double getParam() const override;
```

```
bool isPointIn(const Point &) const override;
```

```
private:
```

~~(Circle) = default;~~

```
int _radius = 0;
```

```
};
```

```
Circle::Circle(int x, int y, int radius) : Shape(1), -> radius  
(radius)
```

```
{ Shape::setPoint(0, x, y); }
```

```
double Circle::getArea() const
```

```
{ return ShapeConsts::Pi * radius * radius;
```

```
double Circle::getParam() const
```

```
{ return 2 * ShapeConsts::Pi * radius;
```

```
bool Circle::isPointIn(int x, int y) const
{
    Shape::point p(x, y);
    return p.getDist(getPointAtIndex(0)) <= radius;
}
```

```
#pragma once
```

Rectangle.h

```
#include "Shape.h"
```

```
class Rectangle: public Shape
```

```
{
```

```
public:
```

```
    Rectangle (int x1, int y1, int x3, int y3);
```

```
    double getArea() const override;
```

```
    bool isPointIn (int x, int y) const override;
```

```
};
```

```
Rectangle::Rectangle (int x1, int y1, int x3, int y3): Shape(4)
```

```
{
```

```
    setPoint(0, x1, y1);
```

```
    setPoint(1, x1, y3);
```

```
    setPoint(2, x3, y3);
```

```
    setPoint(3, x3, y1);
```

```
}
```

```
double Rect::getArea() const
```

```
{
```

```
    const Shape::point & p0 = getPointAtIndex(0);
```

```
    const Shape::point & p1 = getPointAtIndex(1);
```

```
    const Shape::point & p3 = getPointAtIndex(3);
```

```
    return p0.getDist(p1)*p2.getDist(p3);
```

```
}
```

Code Rect::isPointIn(int x, int y) const

{

return $x \geq$ getPointAtIndex(0).x & $x \leq$

}