Указатели и полиморфизм

#include <iostream>

using namespace std;

class Creature {

public:

Creature() {

m\_name = "Creature";

};

string get\_name() {

return m\_name;

}

protected:

string m\_name;

};

class Dragon : public Creature {

public:

Dragon() {

m\_name = "Dragon";

};

};

Добавим существ

#include <iostream>

using namespace std;

class Creature {

public:

Creature() {

m\_name = "Creature";

};

string get\_name() {

return m\_name;

}

protected:

string m\_name;

};

class Dragon : public Creature {

public:

Dragon() {

m\_name = "Dragon";

};

};

class Goblin : public Creature {

public:

Goblin() {

m\_name = "Goblin";

};

};

class Ork : public Creature {

public:

Ork() {

m\_name = "Ork";

};

};

int main() {

Creature\*\* creatures = new Creature \* [3];

creatures[0] = new Dragon();

creatures[1] = new Goblin();

creatures[2] = new Ork();

for (size\_t i = 0; i < 3; ++i) {

cout << creatures[i]->get\_name();

}

for (size\_t i = 0; i < 3; ++i) {

delete creatures[i];

}

delete creatures;

return 0;

}

Виртуальные методы

#include <iostream>

using namespace std;

class Creature {

public:

string get\_name() {

return "Creature";

}

};

class Dragon : public Creature {

public:

string get\_name() {

return "Dragon";

}

};

int main() {

Dragon dragon;

Creature\* dragon\_ptr = &dragon;

// Вызывается Creature::get\_name()

cout << dragon\_ptr->get\_name();

return 0;

}

Виртуальный деструктор

#include <iostream>

using namespace std;

class Creature {

public:

Creature() {

cout << "Creature CONSTRUCTOR\n";

}

~Creature() {

cout << "Creature DESTRUCTOR\n";

}

};

class Dragon : public Creature {

public:

Dragon() {

cout << "Dragon CONSTRUCTOR\n";

}

~Dragon() {

cout << "Dragon DESTRUCTOR\n";

}

};

int main() {

Creature\* creature = new Dragon();

delete creature;

return 0;

}

Корректно:

#include <iostream>

using namespace std;

class Creature {

public:

Creature() {

cout << "Creature CONSTRUCTOR\n";

}

virtual ~Creature() {

cout << "Creature DESTRUCTOR\n";

}

};

class Dragon : public Creature {

public:

Dragon() {

cout << "Dragon CONSTRUCTOR\n";

}

~Dragon() override {

cout << "Dragon DESTRUCTOR\n";

}

};

Чисто виртуальные ф-ии и абстрактные классы

#include <iostream>

using namespace std;

class BinaryOperation {

public:

BinaryOperation() = default;

virtual bool calculate() = 0;

public:

int m\_left;

int m\_rigth;

};

class IsEqual : public BinaryOperation {

public:

IsEqual(int left, int rigth) {

m\_left = left;

m\_rigth = rigth;

}

bool calculate() override {

return m\_left == m\_rigth;

}

};

class IsMore : public BinaryOperation {

public:

IsMore(int left, int rigth) {

m\_left = left;

m\_rigth = rigth;

}

bool calculate() override {

return m\_left > m\_rigth;

}

};

2 - РАБОТА С ПРОИЗВОДНЫМИ КЛАССАМИ ПРИ ПОМОЩИ УКАЗАТЕЛЯ

#include <iostream>

using namespace std;

class X //задаем класс х

{

protected: //модификатор доступа - i доступна только этому классу

int i;

public: //модификатор доступа

void seti(int c) { i = c; }

virtual void print() { cout << endl << "class X : " << i; }

};

class Y : public X // наследование, задали класс у

{

public:

void print() { cout << endl << "class Y : " << i; } // переопределение базовой функции

};

int main()

{

X x;

X\* px = &x; // Указатель на базовый класс

Y y;

x.seti(10);

y.seti(15);

px->print(); // класс X: 10

px = &y;

px->print(); // класс Y: 15

cin.get();

return 0;

}

3 - КЛЮЧЕВОЕ СЛОВО VIRTUAL

#include < iostream>

using namespace std;

class Base //задаем класс базовый

{

public:

virtual void f() //задаем виртуальную ф-ию

{

cout << "base\n";

}

};

class Derived : public Base //задаем унаследованный класс

{

void f() override // 'override' здесь опционально

{

cout << "derived \n";

}

};

int main()

{

Base b;

Derived d;

// Вызов virtual-функции по ссылке:

Base& br = b; // тип br это Base&

Base& dr = d; // тип dr также Base&

br.f(); // напечатается "base"

dr.f(); // напечатается "derived"

// Вызов virtual-функции через указатель:

Base\* bp = &b; // тип bp это Base\*

Base\* dp = &d; // тип dp также Base\*

bp->f(); // напечатается "base"

dp->f(); // напечатается "derived"

// Вызов невиртуальной функции:

br.Base::f(); // напечатается "base"

dr.Base::f(); // напечатается "base"

}

4 - ВИРТУАЛЬНЫЙ ДЕСТРУКТОР

#include <iostream>

using namespace std;

// Вспомогательный класс

class Object

{

public:

Object() { cout << "Object::ctor()" << endl; }

~Object() { cout << "Object::dtor()" << endl; }

};

// Базовый класс

class Base

{

public:

Base() { cout << "Base::ctor()" << endl; }

virtual ~Base() { cout << "Base::dtor()" << endl; }

virtual void print() = 0;

};

// Производный класс

class Derived : public Base

{

public:

Derived() { cout << "Derived::ctor()" << endl; }

~Derived() { cout << "Derived::dtor()" << endl; }

void print() {}

Object obj;

};

int main()

{

Base\* p = new Derived;

delete p;

return 0;

}

6 - АБСТРАКТНЫЕ КЛАССЫ

class Figure

{

public:

virtual double getSquare() = 0;

virtual double getPerimeter() = 0;

virtual void showFigureType() = 0;

};

///

#include <iostream>

using namespace std;

class Figure //абстрактный класс фигуры

{

public:

//виртуальные ф-ии

virtual double getSquare() = 0;

virtual double getPerimeter() = 0;

virtual void showFigureType() = 0;

};

//наследованный класс прямоугольник

class Rectangle : public Figure

{

private:

//поле класса

double width;

double height;

public:

Rectangle(double w, double h) : width(w), height(h)

{

}

double getSquare() override //виртуальная ф-ия переопределяет поведение др.в.ф-ии

{

return width \* height;

}

double getPerimeter() override //виртуальная ф-ия переопределяет поведение др.в.ф-ии

{

return width \* 2 + height \* 2;

}

void showFigureType()

{

cout << "Rectangle" << endl;

}

};

class Circle : public Figure //наследованный класс круг

{

private:

double radius;

public:

Circle(double r) : radius(r)

{

}

double getSquare() override //виртуальная ф-ия переопределяет поведение др.в.ф-ии

{

return radius \* radius \* 3.14;

}

double getPerimeter() override //виртуальная ф-ия переопределяет поведение др.в.ф-ии

{

return 2 \* 3.14 \* radius;

}

void showFigureType()

{

cout << "Circle" << endl;

}

};

int main()

{

Rectangle rect(30, 50);

Circle circle(30);

cout << "Rectangle square: " << rect.getSquare() << endl;

cout << "Circle square: " << circle.getSquare() << endl;

return 0;

}