**Министерство образования и науки Российской Федерации**

**Пермское федеральное государственное бюджетное образовательное учреждение высшего образования «Пермский Национальный Исследовательский Политехнический Университет»**

**Электротехнический факультет**

**Кафедра «Информационные технологии и автоматизированные системы»**

**ОТЧЁТ**

По лабораторной работе №18.5 на тему

«Полиморфизм подтипов»

Вариант №11

Выполнил студент группы РИС-20-1б

Шумилов Лев Сергеевич

Проверил доцент кафедры ИТАС

Полякова О.А.

Пермь 2021

**Цель работы** –научиться работать с одним из самых используемых механизмов в объектно-ориентированном программировании – с полиморфизмом подтипов, базируемом на принципах наследования.

**Постановка задачи**

Задача – реализовать на языке программирования C++ программу, демонстрирующую работу полиморфизма подтипов за счёт работы наследования и переопределения класссов.

**Исходные данные для варианта №11:**

* Класс-родитель – тройка чисел;
* Класс-ребёнок – время (состоит из трёх чисел);
* Класс-контейнер: вектор, который содержит в себе элементы класса-родителя;

**Анализ задачи**

Для решения задачи были использованы следующие средства:

1. Язык программирования C++ (Microsoft Visual C++)
2. Текстовый редактор Microsoft Visual Studio Code
3. Система контроля версий Git

Объявление класса-родителя «Тройка чисел» в заголовочном файле:

#pragma once

#include <iostream>

using namespace std;

class Triad

{

protected:

int \_first;

int \_second;

int \_third;

public:

int First() { return \_first; }

int Second() { return \_second; }

int Third() { return \_third; }

virtual void SetTriad(int first, int second, int third);

virtual void SetFirst(int first);

virtual void SetSecond(int second);

virtual void SetThird(int third);

virtual void IncFirst();

virtual void IncSecond();

virtual void IncThird();

Triad& operator=(Triad& triad);

friend istream& operator>>(istream& input, Triad& triad);

friend ostream& operator<<(ostream& output, Triad& triad);

Triad();

Triad(int first, int second, int third);

Triad(Triad& copied);

};

Производный от него класс «Время»:

#pragma once

#include "Triad.h"

#include <iostream>

using namespace std;

class Time : public Triad

{

public:

int Hour() { return \_first; }

int Minute() { return \_second; }

int Second() { return \_third; }

void SetFirst(int first);

void SetSecond(int second);

void SetThird(int third);

void SetTriad(int first, int second, int third);

void IncFirst();

void IncSecond();

void IncThird();

friend istream& operator>>(istream& input, Time& time);

friend ostream& operator<<(ostream& output, Time& time);

Time();

Time(int first, int second, int third);

Time(Time& copied);

};

Класс-ребёнок наследует от «родителя» все его методы и конструктор с private-полями; Такие решение позволяет создавать множество похожих объектов без необходимости переписывать много кода;

Различия в реализации модификаторов полей классов:

Тройка чисел:

void Triad::SetTriad(int first, int second, int third)

{

\_first = first;

\_second = second;

\_third = third;

}

void Triad::SetFirst(int first)

{

\_first = first;

}

void Triad::SetSecond(int second)

{

\_second = second;

}

void Triad::SetThird(int third)

{

\_third = third;

}

Время (закладываются дополнительные проверки на временные промежутки):

void Time::SetFirst(int first)

{

\_first = first % 24;

}

void Time::SetSecond(int second)

{

if (second >= 0)

{

if (second >= 60)

{

SetFirst(\_first + second / 60);

\_second = second % 60;

}

else

\_second = second;

}

else

cout << "Error: invalid value = " << second << "!\n";

}

void Time::SetThird(int third)

{

if (third >= 0)

{

if (third >= 60)

{

SetSecond(\_second + third / 60);

\_third = third % 60;

}

else

\_third = third;

}

else

cout << "Error: invalid value = " << third << "!\n";

}

void Time::SetTriad(int first, int second, int third)

{

SetFirst(first);

SetSecond(second);

SetThird(third);

}

Код класса-родителя, передаваемый по наследству «ребёнку»:

void Triad::IncFirst()

{

\_first++;

}

void Triad::IncSecond()

{

\_second++;

}

void Triad::IncThird()

{

\_third++;

}

Triad& Triad::operator=(Triad& triad)

{

\_first = triad.\_first;

\_second = triad.\_second;

\_third = triad.\_third;

return \*this;

}

std::istream& operator>>(std::istream& input, Triad& triad)

{

int

first,

second,

third;

std::cout << "Enter the first number: ";

input >> first;

std::cout << "Enter the second number: ";

input >> second;

std::cout << "Enter the third number: ";

input >> third;

triad.SetTriad(first, second, third);

return input;

}

std::ostream& operator<<(std::ostream& output, Triad& triad)

{

output << "First = " << triad.First() << std::endl;

output << "Second = " << triad.Second() << std::endl;

output << "Third = " << triad.Third() << std::endl << std::endl;

return output;

}

Triad::Triad()

{

\_first = 0;

\_second = 0;

\_third = 0;

}

Triad::Triad(int first, int second, int third)

{

SetTriad(first, second, third);

}

Triad::Triad(Triad& copied)

{

\_first = copied.\_first;

\_second = copied.\_second;

\_third = copied.\_third;

}

Класс-контейнер:

#pragma once

#include "Triad.h"

class Vector

{

private:

Triad\*\* \_begin;

int \_length;

int \_currentPosition;

public:

Vector();

Vector(int length);

Vector(const Vector& copied);

~Vector();

void AddTriad(Triad\* triad);

void Print();

};

Реализация:

#include "Vector.h"

#include <iostream>

using namespace std;

Vector::Vector()

{

\_begin = nullptr;

\_length = 0;

\_currentPosition = 0;

}

Vector::Vector(int length)

{

\_length = length;

\_begin = new Triad\* [\_length];

\_currentPosition = 0;

}

Vector::Vector(const Vector& copied)

{

\_length = copied.\_length;

\_currentPosition = copied.\_currentPosition;

if (\_length != 0)

{

\_begin = new Triad \* [\_length];

for (int i = 0; i < \_length; i++)

\_begin[i] = copied.\_begin[i];

}

}

Vector::~Vector()

{

if (\_length != 0)

delete[] \_begin;

\_begin = nullptr;

}

void Vector::AddTriad(Triad\* triad)

{

if (\_currentPosition < \_length)

{

\_begin[\_currentPosition] = triad;

\_currentPosition++;

}

}

void Vector::Print()

{

for (int i = 0; i < \_currentPosition; i++)

cout << \*(\_begin[i]);

cout << endl;

}

Основная программа, которая демонстрирует полиморфизм в действии:

#include <iostream>

#include "Triad.h"

#include "Time.h"

#include "Vector.h"

using namespace std;

int main()

{

Vector vtr(10);

Triad tr(253, 539, 5369);

Time tm(23, 59, 59);

Time tm2(11, 5, 12);

vtr.AddTriad(&tr);

vtr.AddTriad(&tm);

vtr.AddTriad(&tm2);

vtr.Print();

return 0;

}

**Полный исходный код программы на языке программирования C++:**

Triad.h

#pragma once

#include <iostream>

using namespace std;

class Triad

{

protected:

int \_first;

int \_second;

int \_third;

public:

int First() { return \_first; }

int Second() { return \_second; }

int Third() { return \_third; }

virtual void SetTriad(int first, int second, int third);

virtual void SetFirst(int first);

virtual void SetSecond(int second);

virtual void SetThird(int third);

virtual void IncFirst();

virtual void IncSecond();

virtual void IncThird();

Triad& operator=(Triad& triad);

friend istream& operator>>(istream& input, Triad& triad);

friend ostream& operator<<(ostream& output, Triad& triad);

Triad();

Triad(int first, int second, int third);

Triad(Triad& copied);

};

Triad.cpp

#include <iostream>

#include <string>

#include "../Header files/Triad.h"

void Triad::SetTriad(int first, int second, int third)

{

\_first = first;

\_second = second;

\_third = third;

}

void Triad::SetFirst(int first)

{

\_first = first;

}

void Triad::SetSecond(int second)

{

\_second = second;

}

void Triad::SetThird(int third)

{

\_third = third;

}

void Triad::IncFirst()

{

\_first++;

}

void Triad::IncSecond()

{

\_second++;

}

void Triad::IncThird()

{

\_third++;

}

Triad& Triad::operator=(Triad& triad)

{

\_first = triad.\_first;

\_second = triad.\_second;

\_third = triad.\_third;

return \*this;

}

std::istream& operator>>(std::istream& input, Triad& triad)

{

int

first,

second,

third;

std::cout << "Enter the first number: ";

input >> first;

std::cout << "Enter the second number: ";

input >> second;

std::cout << "Enter the third number: ";

input >> third;

triad.SetTriad(first, second, third);

return input;

}

std::ostream& operator<<(std::ostream& output, Triad& triad)

{

output << "First = " << triad.First() << std::endl;

output << "Second = " << triad.Second() << std::endl;

output << "Third = " << triad.Third() << std::endl << std::endl;

return output;

}

Triad::Triad()

{

\_first = 0;

\_second = 0;

\_third = 0;

}

Triad::Triad(int first, int second, int third)

{

SetTriad(first, second, third);

}

Triad::Triad(Triad& copied)

{

\_first = copied.\_first;

\_second = copied.\_second;

\_third = copied.\_third;

}

Time.h

#pragma once

#include "Triad.h"

#include <iostream>

using namespace std;

class Time : public Triad

{

public:

int Hour() { return \_first; }

int Minute() { return \_second; }

int Second() { return \_third; }

void SetFirst(int first);

void SetSecond(int second);

void SetThird(int third);

void SetTriad(int first, int second, int third);

void IncFirst();

void IncSecond();

void IncThird();

friend istream& operator>>(istream& input, Time& time);

friend ostream& operator<<(ostream& output, Time& time);

Time();

Time(int first, int second, int third);

Time(Time& copied);

};

Time.cpp

#include "../Header files/Time.h"

#include <iostream>

void Time::SetFirst(int first)

{

\_first = first % 24;

}

void Time::SetSecond(int second)

{

if (second >= 0)

{

if (second >= 60)

{

SetFirst(\_first + second / 60);

\_second = second % 60;

}

else

\_second = second;

}

else

cout << "Error: invalid value = " << second << "!\n";

}

void Time::SetThird(int third)

{

if (third >= 0)

{

if (third >= 60)

{

SetSecond(\_second + third / 60);

\_third = third % 60;

}

else

\_third = third;

}

else

cout << "Error: invalid value = " << third << "!\n";

}

void Time::SetTriad(int first, int second, int third)

{

SetFirst(first);

SetSecond(second);

SetThird(third);

}

void Time::IncFirst()

{

SetFirst(\_first + 1);

}

void Time::IncSecond()

{

SetSecond(\_second + 1);

}

void Time::IncThird()

{

SetThird(\_third + 1);

}

Time::Time()

{

\_first = 0;

\_second = 0;

\_third = 0;

}

Time::Time(int first, int second, int third)

{

SetTriad(first, second, third);

}

Time::Time(Time& copied)

{

\_first = copied.\_first;

\_second = copied.\_second;

\_third = copied.\_third;

}

istream& operator>>(istream& input, Time& time)

{

int

first,

second,

third;

std::cout << "Enter hours: ";

input >> first;

std::cout << "Enter minutes: ";

input >> second;

std::cout << "Enter seconds: ";

input >> third;

time.SetTriad(first, second, third);

return input;

}

ostream& operator<<(ostream& output, Time& time)

{

output << "Time = " << time.Hour() << ':' << time.Minute() << ':' << time.Second() << ' ';

return output;

}

Vector.h

#pragma once

#include "Triad.h"

class Vector

{

private:

Triad\*\* \_begin;

int \_length;

int \_currentPosition;

public:

Vector();

Vector(int length);

Vector(const Vector& copied);

~Vector();

void AddTriad(Triad\* triad);

void Print();

};

Vector.cpp

#include "Vector.h"

#include <iostream>

using namespace std;

Vector::Vector()

{

\_begin = nullptr;

\_length = 0;

\_currentPosition = 0;

}

Vector::Vector(int length)

{

\_length = length;

\_begin = new Triad\* [\_length];

\_currentPosition = 0;

}

Vector::Vector(const Vector& copied)

{

\_length = copied.\_length;

\_currentPosition = copied.\_currentPosition;

if (\_length != 0)

{

\_begin = new Triad \* [\_length];

for (int i = 0; i < \_length; i++)

\_begin[i] = copied.\_begin[i];

}

}

Vector::~Vector()

{

if (\_length != 0)

delete[] \_begin;

\_begin = nullptr;

}

void Vector::AddTriad(Triad\* triad)

{

if (\_currentPosition < \_length)

{

\_begin[\_currentPosition] = triad;

\_currentPosition++;

}

}

void Vector::Print()

{

for (int i = 0; i < \_currentPosition; i++)

cout << \*(\_begin[i]);

cout << endl;

}

main.cpp

#include <iostream>

#include "Triad.h"

#include "Time.h"

#include "Vector.h"

using namespace std;

int main()

{

Vector vtr(10);

Triad tr(253, 539, 5369);

Time tm(23, 59, 59);

Time tm2(11, 5, 12);

vtr.AddTriad(&tr);

vtr.AddTriad(&tm);

vtr.AddTriad(&tm2);

vtr.Print();

return 0;

}

**Ответы на вопросы:**

**Какой метод называется чисто виртуальным? Чем он отличается от виртуального метода?**

Чисто виртуальный метод содержит признак = 0 вместо тела, например:

virtual void f(int) = 0;

Чисто виртуальный метод должен переопределяться в производном классе (возможно, опять как чисто виртуальный).

**Какой класс называется абстрактным?**

Класс, содержащий хотя бы один чисто виртуальный метод.

**Для чего предназначены абстрактные классы?**

Абстрактные классы предназначены для представления общих понятий, которые предполагается конкретизировать в производных классах. Абстрактный класс может использоваться только в качестве базового для других классов — объекты абстрактного класса создавать нельзя, поскольку прямой или косвенный вызов чисто виртуального метода приводит к ошибке при выполнении.

**Что такое полиморфные функции?**

Функция, параметром которой является указатель на абстрактный класс. На место этого параметра при выполнении программы может передаваться указатель на объект любого производного класса. Это позволяет работать с объектом любого типа в пределах одной иерархии.

**Чем полиморфизм отличается от принципа подстановки?**

Принцип звучит так: везде, где может быть использован объект базового класса (при присваивании, при передаче параметров и возврате результата), вместо него разрешается использовать объект производного класса.

Полиморфизм- способность функции работать с переменными разных классов одной иерархии благодаря наличию в основании иерархии абстрактного класса.

**В каких случаях используется механизм позднего связывания?**

Для выполнения позднего связывания используются указатели на функции. Указатель на функцию — это тип указателя, который указывает на функцию вместо переменной. Функция, на которую указывает указатель, может быть вызвана через указатель и оператор вызова функции.

Он используется в случаях, когда заранее неизвестно, какой вариант полиморфной функции будет вызван.

**Скриншот выполненной программы:**

