Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Пермский национальный исследовательский политехнический университет»
Электротехнический факультет
Кафедра «Информационные технологии и автоматизированные системы»

ЛАБОРАТОРНАЯ РАБОТА №5

«*Наследование. Виртуальные функции. Полиморфизм*»

Выполнила:
студентка группы ИВТ-23-2б
Соловьева Екатерина
Александровна
Проверила:
доцент кафедры ИТАС
О. А. Полякова

2024 г.

Постановка задачи

1. Определить абстрактный класс.
2. Определить иерархию классов, в основе которой будет находиться абстрактный класс (см. лабораторную работу №4).
3. Определить класс Вектор, элементами которого будут указатели на объекты иерархии классов.
4. Перегрузить для класса Вектор операцию вывода объектов с помощью потоков.
5. В основной функции продемонстрировать перегруженные операции и полиморфизм Вектора.

11 Вариант:

Базовый класс:
ТРОЙКА_ЧИСЕЛ (TRIAD)
Первое_число (first) - int Второе_число (second) – int
Третье_число (third) - int

Определить методы изменения полей и увеличения полей на 1.
Создать производный класс TIME с полями часы, минуты и секунды. Переопределить методы увеличения полей на 1 и определить методы увеличения на n секунд и минут

Код программы на C++

Odject.h

```
#pragma once
#include <iostream>
using namespace std;
class Object {
public:
	Object() {}
public:
	~Object() {}
	virtual void Show() = 0;
};
```

Triad.h

```
#pragma once
#include "Object.h"
#include <iostream>
using namespace std;
class Triad :
	public Object {
public:
	Triad(void);
public:
	virtual ~Triad(void);
	void Show() {}
	Triad(int, int, int);
	Triad(const Triad&);
	int getFirst() { return first; }
	int getSecond() { return second; }
	int getThird() { return third; }
	void setFirst(int);
	void setSecond(int);
	void setThird(int);
	Triad& operator =(const Triad&);
	friend istream& operator >>(istream& in, Triad& T);
	friend ostream& operator <<(ostream& out, const Triad& T);
protected:
	int first, second, third;
};
```

Time.h

```
#pragma once
#include <iostream>
#include "Triad.h"
using namespace std;
class Time :
	public Triad {
public:
	Time(void);
public:
	~Time(void);
	Time(int, int, int);
	Time(const Time&);
	int getHour() { return hour; }
	int getMin() { return min; }
	int getSec() { return sec; }
	void setHour(int);
	void setMin(int);
	void setSec(int);
	Time& operator =(const Time&);
	friend istream& operator >>(istream& in, Time& time);
	friend ostream& operator <<(ostream& out, const Time& time);
protected:
	int hour, min, sec;
};
```

Vector.h

```
#pragma once
#include <iostream>
#include "Object.h"
using namespace std;
class Vector {
public:
	Vector(void);
	Vector(int);
public:
	~Vector(void);
	void Add(Object*);
	friend ostream& operator <<(ostream& out, const Vector&);
private:
	Object** ptr;
	int size, cur;
};
```

Time.cpp

```
#include <iostream>
#include "Time.h"
using namespace std;
Time::Time(void) :Triad() {
	hour = 0;
	min = 0;
	sec = 0;
}
Time::~Time(void) {}
Time::Time(int H, int M, int S) {
	hour = H;
	min = M;
```

```
        sec = S;
}
Time::Time(const Time& time) {
        hour = time.hour;
        sec = time.sec;
        min = time.min;
}
void Time::setHour(int H) {
        hour = H;
}
void Time::setMin(int M) {
        min = M;
}
void Time::setSec(int S) {
        sec = S;
}
Time& Time::operator =(const Time& time) {
        if (&time == this) {
                return *this;
        }
        first = time.first;
        second = time.second;
        third = time.third;
        hour = time.hour;
        min = time.min;
        sec = time.sec;
        return *this;
}
istream& operator >>(istream& in, Time& time) {
        cout << "\nFirst: " << time.first;
```

```
        cout << "\nSecond: " << time.second;
        cout << "\nThird: " << time.third;
        cout << "\nHour: " << time.hour;
        cout << "\nMinutes: " << time.min;
        cout << "\nSeconds: " << time.sec;
        return in;
}
ostream& operator <<(ostream& out, const Time& time) {
        out << "\nFirst: " << time.first;
        out << "\nSecond: " << time.second;
        out << "\nThird: " << time.third;
        out << "\nHour: " << time.hour;
        out << "\nMinutes: " << time.min;
        out << "\nSeconds: " << time.sec;
        return out;
}
```

Triad.cpp

```
#include "Triad.h"
#include <iostream>
Triad::Triad(void) {
        first = 0;
        second = 0;
        third = 0;
}
Triad::~Triad(void) {}
Triad::Triad(int F, int S, int T) {
        first = F;
        second = S;
        third = T;
}
```

```
Triad::Triad(const Triad& triad) {
	first = triad.first;
	second = triad.second;
	third = triad.third;
}
void Triad::setFirst(int F) {
	first = F;
}
void Triad::setSecond(int S) {
	second = S;
}
void Triad::setThird(int T) {
	third = T;
}
Triad& Triad::operator =(const Triad& triad) {
	if (&triad == this) {
		return *this;
	}
	first = triad.first;
	second = triad.second;
	third = triad.third;
	return *this;
}
istream& operator >>(istream& in, Triad& T) {
	cout << "\nFirst: "; in >> T.first;
	cout << "\nSecond: "; in >> T.second;
	cout << "\nThird: "; in >> T.third;
	return in;
}
ostream& operator <<(ostream& out, const Triad& T) {
```

```
        out << "\nFIRST: " << T.first;
        out << "\nSECOND: " << T.second;
        out << "\nTHIRD: " << T.third;
        return out;
}
```

Vector.cpp

```
#include <iostream>
#include "Vector.h"
using namespace std;
Vector::Vector() {
        ptr = 0;
        size = 0;
        cur = 0;
}
Vector::~Vector() {
        if (ptr != 0) {
                delete[] ptr;
        }
        ptr = 0;
}
Vector::Vector(int n) {
        ptr = new Object * [n];
        cur = 0;
        size = n;
}
void Vector::Add(Object* p) {
        if (cur < size) {
                ptr[cur] = p;
                cur++;
        }
```

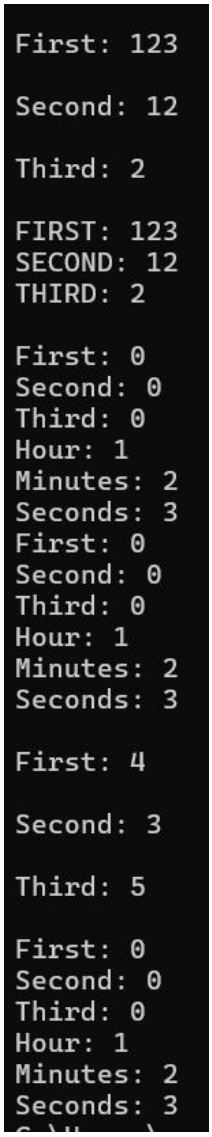
```
}
ostream& operator <<(ostream& out, const Vector& v) {
	if (v.size == 0) {
		out << "EMPTY" << endl;
	}
	Object** p = v.ptr;
	for (int i = 0; i < v.cur; i++) {
		(*p)->Show();
		p++;
	}
	return out;
}
```

Main.cpp

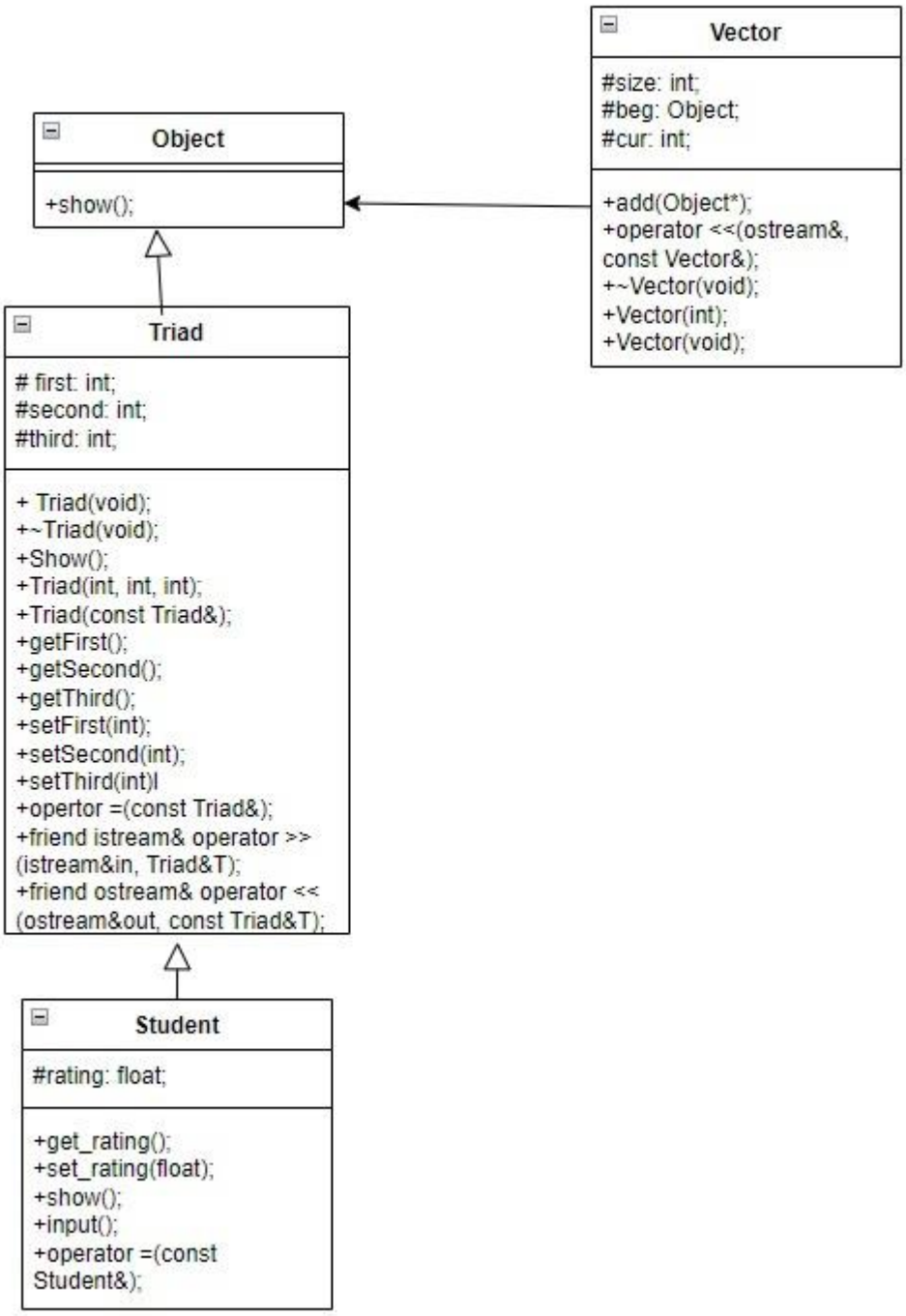
```
#include "Object.h"
#include <iostream>
#include "Time.h"
#include "Triad.h"
#include "Vector.h"
using namespace std;
int main() {
	Triad example;
	cin >> example;
	cout << example << endl;
	Object* p = &example;
	p->Show();
	Time example2;
	cin >> example2;
	cout << example2 << endl;
	p = &example2;
	p->Show();
```

```
    Vector v(5);
    Triad a;
    Time b;
    cin >> a >> b;
    p = &a;
    v.Add(p);
    p = &b;
    v.Add(p);
    cout << v;
    return 0;
}
```

Вывод:



UML-диаграмма



Анализ результатов

Программа сработала корректно и вывела необходимые результаты.