Министерство науки и высшего образования Российской Федерации Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет Кафедра «Информационные технологии и автоматизированные системы» направление подготовки: 09.03.04 - «Программная инженерия»

**«Лабораторная работа №4**

**"Наследование. Виртуальные функции. Полиморфизм."**

Выполнил студент гр. РИС-24-3б Носков Егор Михайлович

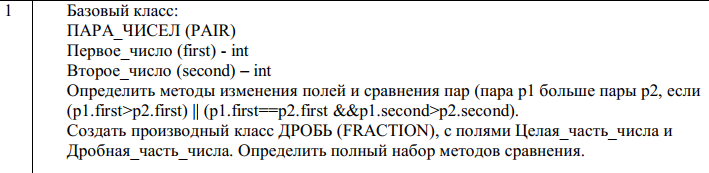
Проверил: Доцент кафедры ИТАС Ольга Андреевна Полякова

|  |  |
| --- | --- |
|  | |
| (оценка) (подпись) | |
|  |
| (дата) |

Г. Пермь, 2024

**Постановка задачи**

**Вариант 1:**

****

**Программа на языке C++**

Object.h:

#pragma once

class Object

{

public:

Object(void) {};

~Object(void) {};

virtual void show()=0;

};

Object.cpp:

#include "Object.h"

Pair.h:

#pragma once

#include "Object.h"

#include <iostream>

using namespace std;

class Pair :

public Object

{

protected:

int first;

int second;

public:

Pair(void);

void show();

Pair(int, int);

Pair(const Pair&);

virtual ~Pair(void);

int get\_first() { return first; }

int get\_second() { return second; }

void set\_first(int);

void set\_second(int);

bool operator>(const Pair&);

bool operator==(const Pair&);

friend istream& operator>>(istream& in, Pair& A);

friend std::ostream& operator<<(ostream& out, const Pair& A);

void printinfo(const Pair&);

};

Pair.cpp:

#include "Pair.h"

Pair::Pair(void)

{

first = 0;

second = 0;

}

Pair::~Pair(void) {}

Pair::Pair(int A, int B) { first = A; second = B; }

Pair::Pair(const Pair& pair) { this->first = pair.first; this->second = pair.second; }

void Pair::set\_first(int A) { first = A; }

void Pair::set\_second(int A) { second = A; }

bool Pair::operator>(const Pair& A) { if ((this->first > A.first) || ((this->first == A.first) && this->second > A.second)) return 1; else return 0; }

bool Pair::operator==(const Pair& A) { if ((this->first == A.first) && (this->second == A.second)) return 1; else return 0; }

istream& operator>>(istream& in, Pair& A) {

cout << "Введите первое число: "; in >> A.first;

cout << "Введите второе число: "; in >> A.second;

return in;

}

ostream& operator<<(ostream& out, const Pair& A) {

out << "Пара чисел:" << endl;

out << "Первое число: " << A.first << endl;

out << "Второе число: " << A.second << endl;

return out;

}

void Pair::printinfo(const Pair& A) { cout << A << endl; }

void Pair::show() {

cout << "Первое число: " << this->first<<endl;

cout << "Второе число: " << this->second<<endl;

}

Fraction.h:

#pragma once

#include "Pair.h"

class Fraction :

public Pair

{

protected:

int integerpart;

int fraction;

public:

Fraction(void);

~Fraction(void);

void show();

Fraction(int, int, int, int);

Fraction(const Fraction&);

int get\_integerpart() { return integerpart; }

int get\_fraction() { return fraction; }

void set\_integerpart(int);

void set\_fraction(int);

bool operator>(const Fraction&);

bool operator==(const Fraction&);

friend istream& operator>>(istream& in, Fraction& A);

friend std::ostream& operator<<(ostream& out, const Fraction& A);

void printinfo(const Fraction&);

};

Fraction.cpp:

#include "Fraction.h"

Fraction::Fraction(void) : Pair() { integerpart = 0; fraction = 0; }

Fraction::~Fraction(void) {}

Fraction::Fraction(int A, int B, int C, int D) : Pair(A, B) { integerpart = C; fraction = D; }

Fraction::Fraction(const Fraction& A) { this->first = A.first; this->second = A.second; this->integerpart = A.integerpart; this->fraction = A.fraction; }

void Fraction::set\_integerpart(int A) { this->integerpart = A; }

void Fraction::set\_fraction(int A) { this->fraction = A; }

bool Fraction::operator>(const Fraction& A) { if ((this->integerpart > A.integerpart) || ((this->integerpart == A.integerpart) && this->fraction > A.fraction)) return 1; else return 0; }

bool Fraction::operator==(const Fraction& A) { if ((this->integerpart == A.integerpart) && (this->fraction == A.fraction)) return 1; else return 0; }

istream& operator>>(istream& in, Fraction& A) {

cout << "Введите первое число: "; in >> A.first;

cout << "Введите второе число: "; in >> A.second;

cout << "Введите целую часть дроби: "; in >> A.integerpart;

cout << "Введите дробную часть: "; in >> A.fraction;

return in;

}

ostream& operator<<(ostream& out, const Fraction& A) {

out << "Пара чисел:" << endl;

out << "Первое число: " << A.first << endl;

out << "Второе число: " << A.second << endl;

out << "Дробь: " << A.integerpart << "." << A.fraction << endl;

return out;

}

void Fraction::printinfo(const Fraction& A) { cout << A << endl; }

void Fraction::show() {

cout << "Пара чисел:" << endl;

cout << "Первое число: " << this->first << endl;

cout << "Второе число: " << this->second << endl;

cout << "Дробь: " << this->integerpart << "." << this->fraction << endl;

}

Vector.h:

#pragma once

#include "Object.h"

#include <iostream>

using namespace std;

class Vector

{

public:

Vector(void);

Vector(int);

~Vector(void);

void Add(Object\*);

friend std::ostream& operator<<(ostream& out, const Vector&);

private:

Object\*\* beg;

int size;

int cur;

};

Vector.cpp:

#include "Vector.h"

Vector::Vector(void) { beg = 0; size = 0; cur = 0; }

Vector::~Vector(void) {if (beg != 0) { delete[] beg; beg = 0; }}

Vector::Vector(int A) { beg = new Object \* [A]; cur = 0; size = A; }

void Vector::Add(Object\* A) {

if (cur < size) {

beg[cur] = A;

cur++;

}

}

ostream& operator<<(ostream& out, const Vector& A) {

if (A.size == 0) out << "Empty" << endl;

Object\*\* p = A.beg;

for (int i = 0; i < A.cur; i++) {

(\*p)->show();

p++;

}

return out;

}

Pair\_main.cpp:

#include "Object.h"

#include "Pair.h"

#include "Fraction.h"

#include "Vector.h"

#include <iostream>

using namespace std;

int main() {

setlocale(LC\_ALL, "RU");

Vector v(5);

Pair a;

cin >> a;

Fraction b;

cin >> b;

Object\* p = &a;

v.Add(p);

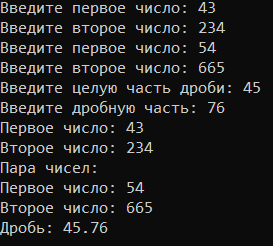
p = &b;

v.Add(p);

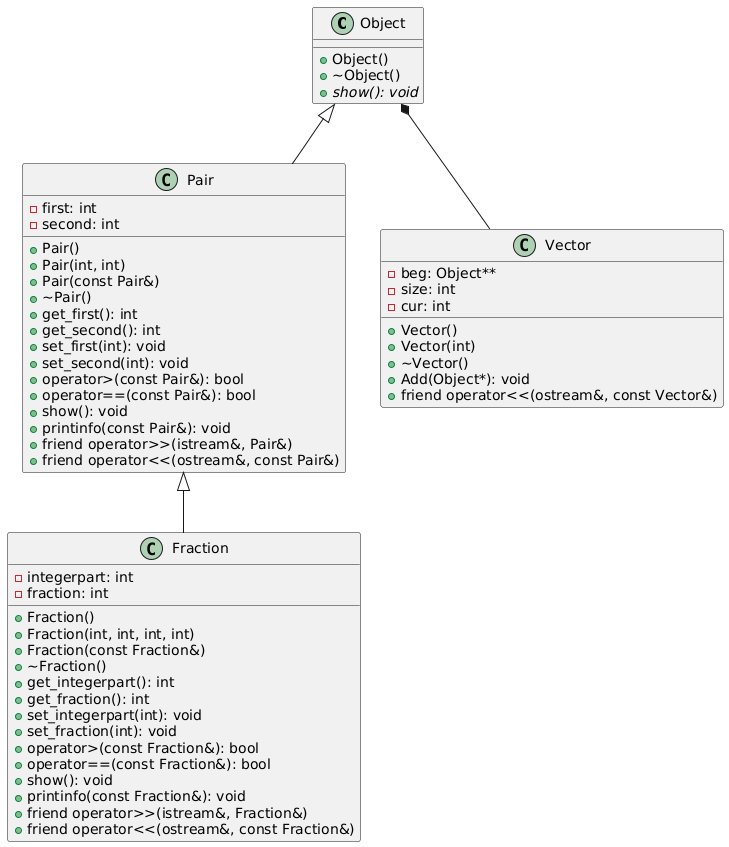
cout << v;

}

Вывод программы:



**UML-диаграмма классов**

****

**Контрольные вопросы**

1. **Какой метод называется чисто виртуальным? Чем он отличается от виртуального метода?**
   * **Чисто виртуальный метод** — это метод, который объявляется в базовом классе с синтаксисом = 0 (например, virtual void show() = 0;). Он не имеет реализации в базовом классе и должен быть переопределён в производных классах.
   * **Отличие от виртуального метода**: Виртуальный метод может иметь реализацию в базовом классе и позволяет производным классам переопределять его, а чисто виртуальный метод не имеет реализации и делает класс абстрактным.
2. **Какой класс называется абстрактным?**  
   Абстрактный класс — это класс, содержащий хотя бы один чисто виртуальный метод. Создавать объекты такого класса нельзя, он предназначен только для наследования.
3. **Для чего предназначены абстрактные классы?**
   * Для определения общего интерфейса для производных классов.
   * Для организации полиморфизма.
   * Для запрета создания объектов базового класса, если он представляет абстрактную концепцию.
4. **Что такое полиморфные функции?**  
   Полиморфные функции — это функции, которые могут работать с объектами разных классов, связанных наследованием, через указатели или ссылки на базовый класс. Они вызывают соответствующие методы в зависимости от типа объекта во время выполнения (позднее связывание).
5. **Чем полиморфизм отличается от принципа подстановки?**
   * **Полиморфизм** — это возможность использовать методы производных классов через интерфейс базового класса.
   * **Принцип подстановки (Liskov Substitution Principle)** — частный случай полиморфизма, требующий, чтобы объекты производных классов могли заменять объекты базового класса без изменения корректности программы.
6. **Примеры иерархий с использованием абстрактных классов:**
   * **Геометрические фигуры**:

class Shape {

public:

virtual double area() = 0; *// Чисто виртуальный метод*

};

class Circle : public Shape { ... };

class Rectangle : public Shape { ... };

* + **Животные**:

class Animal {

public:

virtual void makeSound() = 0;

};

class Dog : public Animal { ... };

class Cat : public Animal { ... };

1. **Примеры полиморфных функций:**
   * Функция, принимающая указатель на базовый класс:

void printArea(Shape\* shape) {

cout << shape->area(); *// Вызовется area() для Circle или Rectangle*

}

* + Виртуальный метод show() в примере из задания:

Object\* obj = new Fraction();

obj->show(); *// Вызовется Fraction::show()*

1. **В каких случаях используется механизм позднего связывания?**
   * При работе с полиморфными функциями и виртуальными методами.
   * Когда необходимо вызывать методы производных классов через указатель/ссылку на базовый класс.
   * В шаблонах проектирования (например, Стратегия, Фабрика).

**Ссылка на репозиторий с отчетом Github**

****