ПНИПУ

Федеральное государственное автономное образовательное учреждение высшего образования «Пермский национальный исследовательский политехнический университет»

Лабораторная работа №3.

Наследование. Виртуальные функции. Полиморфизм.

Выполнил студент группы РИС-23-3Б

Буковский Денис Владимирович

Проверила доцент кафедры ИТАС О.А. Полякова

2024

1. 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.

Базовый класс: ПАРА ЧИСЕЛ (PAIR)

Первое \_число (first) – int

Второе\_число (second) — int

Определить методы изменения полей и операцию сложения пар (a,b)+(c,d)=(a+b,c+d)

Создать производный класс ДЕНЕЖНАЯ CYMMA(MONEY), с полями Рубли и Копейки.

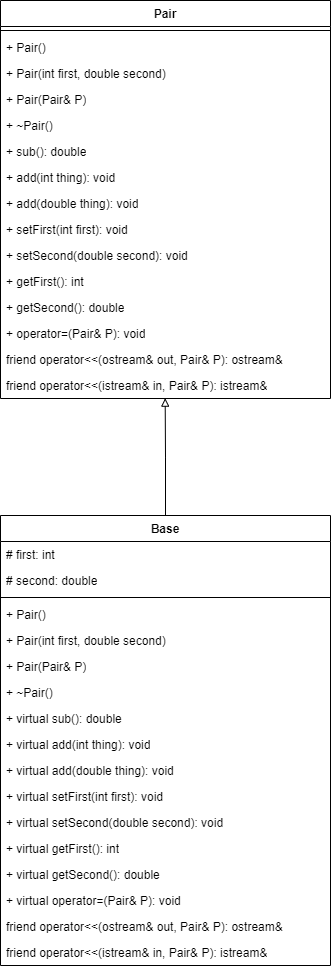
Переопределить операцию сложения и определить операции вычитания и деления денежных сумм.

1. Анализ задачи

Создадим класс Base. Определим в нём чисто виртуальные методы. Унаследуем класс Pair от класса Base и переопределим методы класса.

Протестируем работу программы в функции main().

1. Блок-схема



1. Код

Pair.h

#pragma once

#include <iostream>

class Base {

protected:

int first;

double second;

public:

virtual double sub() = 0;

virtual void add(int) = 0;

virtual void add(double) = 0;

virtual void setFirst(int) = 0;

virtual void setSecond(double) = 0;

virtual int getFirst() = 0;

virtual double getSecond() = 0;

};

class Pair: public Base {

public:

Pair();

Pair(int first, double second);

Pair(Pair& T);

~Pair();

double sub() override;

void add(int thing) override;

void add(double thing) override;

void setFirst(int first) override;

void setSecond(double second) override;

int getFirst() override;

double getSecond() override;

void operator= (Pair& T);

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

friend std::istream& operator>>(std::istream& in, Pair& p);

};

Pair.cpp

#include "Pair.h"

#include <iostream>

Pair::Pair() {

this->first = 0;

this->second = 0.0;

}

Pair::Pair(int first, double second) {

this->first = first;

this->second = second;

}

Pair::Pair(Pair& T) {

this->first = T.first;

this->second = T.second;

}

Pair::~Pair() {

std::cout << std::endl << "Pair " << this->first << " : " << this->second << " was deleted!";

}

double Pair::sub() {

return this->first - this->second;

}

void Pair::add(int thing) {

this->first += thing;

}

void Pair::add(double thing) {

this->second += thing;

}

void Pair::setFirst(int first) {

this->first = first;

}

void Pair::setSecond(double second) {

this->second = second;

}

int Pair::getFirst() {

return this->first;

}

double Pair::getSecond() {

return this->second;

}

void Pair::operator= (Pair& T) {

this->first = T.first;

this->second = T.second;

}

std::ostream& operator<<(std::ostream& out, const Pair& p) { // Функции перегрузки << и >> дружественные, поскольку они используют внешние типы

return(out << p.first << " : " << p.second);

}

std::istream& operator>>(std::istream& in, Pair& p) {

std::cout << "first: ";

in >> p.first;

std::cout << "second: ";

in >> p.second;

return in;

}

Main.cpp

#include <iostream>

#include "Pair.h"

using namespace std;

int main() {

Pair item1, item2 = item1, item3(123, 456.1);

cin >> item1;

cout << endl << item1 << ", " << item2 << ", " << item3 << endl << endl;

item3 = item1;

item3.add(123);

item3.add(456.1);

item3.setFirst(54);

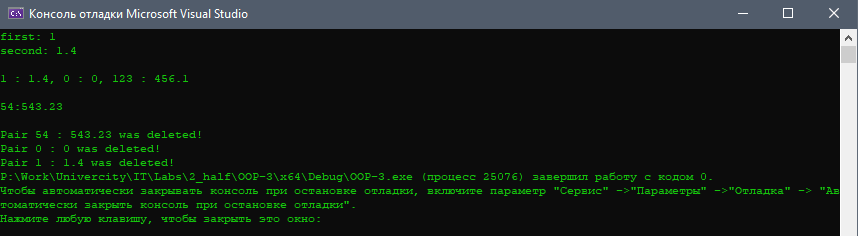
item3.setSecond(543.23);

cout << item3.getFirst() << ":" << item3.getSecond() << endl;

return 0;

}

1. Вывод программы



1. Ответы на контрольные вопросы
2. Чисто виртуальный метод – это виртуальный метод, который не имеет определения в текущем классе и должен быть переопределён в дочернем.
3. Абстрактный класс – это класс который используется для создания других классов.
4. Абстрактные классы нужны для поддержания принципа полиморфизма и взаимодействия между дочерними классами.
5. Полиморфные функции – это функции которые имеют общее название и свою реализацию для каждого дочернего класса.
6. Полиморфизм отличается от принципа подстановки тем, что в принципе подстановки абсолютно все методы дочерних классов должны быть описаны в классе-родителе.
7. Человек – Учитель/Уборщик/Продавец
8. И человек, и учитель могут говорить (как метод).
9. Механизм позднего связывания используется при создании общего интерфейса класса, поддержания связи с классом-родителем.