Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Тема: «Простое наследование. Принцип подстановки»

Выполнил работу

Студент группы РИС-22-2Б

Третьяков Н. А.

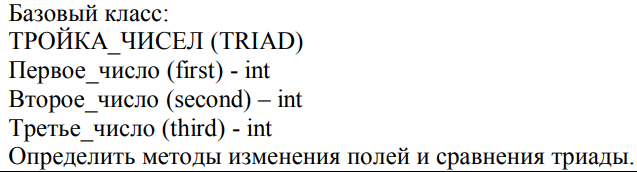
Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

Постановка задачи.





1. Определить пользовательский класс.

2. Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.

3. Определить в классе деструктор.

4. Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).

5. Перегрузить операцию присваивания.

6. Перегрузить операции ввода и вывода объектов с помощью потоков.

7. Определить производный класс.

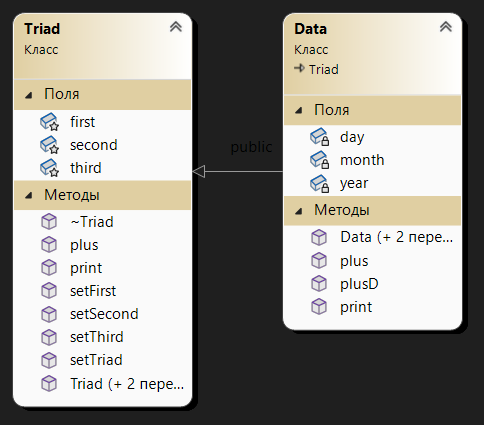
8. Написать программу, в которой продемонстрировать создание объектов и работу

всех перегруженных операций.

9. Реализовать функции, получающие и возвращающие объект базового класса.

Продемонстрировать принцип подстановки.

UML диаграмма.



Код программы.

Data.h

#pragma once

#include <iostream>

#include "Triad.h"

using namespace std;

class Data : public Triad

{

int year;

int month;

int day;

public:

Data();

Data(int y, int m, int d);

Data(const Data&);

void plus();

void plusD(int d);

void print();

};

Data.cpp

#include <iostream>

#include "Data.h"

#include "Triad.h"

using namespace std;

Data::Data()

{

year = 0;

month = 0;

day = 0;

}

Data::Data(int y, int m, int d)

{

if (d > 30)

{

m = d / 30;

day = d % 30;

}

else

{

day = d;

}

if (m > 12)

{

y = m / 12;

month = m % 12;

}

else

{

month = m;

}

year = y;

}

Data::Data(const Data& d)

{

day = d.day;

month = d.month;

year = d.year;

}

void Data::print()

{

cout << year << ". " << month << ". " << day << endl;

}

void Data::plus()

{

year++;

month++;

day++;

}

void Data::plusD(int d)

{

if (day + d > 30)

{

month = (day + d) / 30;

day = (day + d) % 30;

if (month > 12)

{

year += month / 12;

month = month % 12;

}

}

else

{

day += d;

}

}

Triad.h

#pragma once

#include <iostream>

using namespace std;

class Triad

{

protected:

int first;

int second;

int third;

public:

virtual void print();

Triad();

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

Triad(const Triad&);

void setFirst(int f);

void setSecond(int s);

void setThird(int t);

void setTriad(int f, int s, int t);

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

void plus();

~Triad() {};

};

Triad.cpp

#include <iostream>

#include "Triad.h"

using namespace std;

Triad::Triad()

{

first = 0;

second = 0;

third = 0;

}

Triad::Triad(int f, int s, int t)

{

first = f;

second = s;

third = t;

}

Triad::Triad(const Triad& t)

{

first = t.first;

second = t.second;

third = t.third;

}

void Triad::print()

{

cout << "1) " << first << endl;

cout << "2) " << second << endl;

cout << "3) " << third << endl;

}

std::istream& operator >> (std::istream& in, Triad& t)

{

in >> t.first;

in >> t.second;

in >> t.third;

return in;

}

void Triad::plus()

{

first++;

second++;

third++;

}

void Triad::setFirst(int f)

{

first = f;

}

void Triad::setSecond(int s)

{

second = s;

}

void Triad::setThird(int t)

{

third = t;

}

void Triad::setTriad(int f, int s, int t)

{

first = f;

second = s;

third = t;

}

Main.cpp

#include <iostream>

#include "Triad.h"

#include "Data.h"

using namespace std;

int main()

{

setlocale(0, "RUS");

Data d;

Triad t;

bool f = 1;

int n, x, y, z;

while (f)

{

cout << "Changing the Triad object" << endl;

cout << "1) Change the field first" << endl;

cout << "2) Change the second field" << endl;

cout << "3) Change the third field " << endl;

cout << "4) Change all three fields" << endl;

cout << "5) Don't change anything" << endl;

cout << "Select action" << endl;

cin >> n;

switch (n)

{

case 1:

{

cout << "Enter the number to replace the data in the selected field with" << endl;

cin >> x;

t.setFirst(x);

break;

}

case 2:

{

cout << "Enter the number to replace the data in the selected field with" << endl;

cin >> x;

t.setSecond(x);

break;

}

case 3:

{

cout << "Enter the number to replace the data in the selected field with" << endl;

cin >> x;

t.setThird(x);

break;

}

case 4:

{

cout << "Enter the number to replace the data with in the field first" << endl;

cin >> x;

cout << "Enter the number to replace the data with in the field second" << endl;

cin >> y;

cout << "Enter the number to replace the data with in the field third" << endl;

cin >> z;

t.setTriad(x, y, z);

break;

}

case 5:

{

f = 0;

break;

}

}

}

t.print();

d.plus();

d.print();

cout << "How many days do I need to add" << endl;

cin >> x;

d.plusD(x);

d.print();

}

Контрольные вопросы

1. Для чего используется механизм наследования?

- Для создания новых классов, на основе уже существующих.

2. Каким образом наследуются компоненты класса, описанные со спецификатором public?

Компоненты класса, описанные со спецификатором public, становятся доступными в базовом классе. Если тип наследования public, то становятся доступными для обращения из экземпляра производного класса. Если тип наследования protected, то доступны только внутри базового класса.

3. Каким образом наследуются компоненты класса, описанные со спецификатором private?

Компоненты, описанные со спецификатором private, не доступны в производном классе.

4. Каким образом наследуются компоненты класса, описанные со спецификатором protected?

-Компоненты, описанные со спецификатором protected, при типе наследования public/protected остаются protected – доступны в производном классе и в классах-наследниках производного. При типе наследования private, становятся private – Доступны только внутри производного класса.

5. Каким образом описывается производный класс?

Class <имя> : <тип наследования> <имя базового>,…

6. Наследуются ли конструкторы?

-Конструкторы не наследуются, однако могут быть вызваны для передачи параметров.

7. Наследуются ли деструкторы?

- Деструкторы не наследуются.

8. В каком порядке конструируются объекты производных классов?

-Сначала Компоненты базового класса, потом базовый класс, потом компоненты производного класса, затем производный. компоненты конструируются в том порядке, в котором они записаны.

9. В каком порядке уничтожаются объекты производных классов?

В обратном порядке создания. Сначала производный класс, затем компоненты производного (в обратном порядке их записи), затем базовый класс, затем компоненты базового класса(в обратном порядке их записи).

10. Что представляют собой виртуальные функции и механизм позднего связывания?

Виртуальные функции – это функции которые могут быть переопределены в производном. Используя виртуальную функцию, мы можем вызывать функции производного класса, используя указатель базового класса.

Когда в классе объявляется виртуальный метод, то создается специальная таблица для этого класса. Эта таблица представляет собой массив указателей на виртуальные функции. Во время компиляции нет информации о том, какой метод будет вызван. Вызов метода выполняется во время выполнения программы.

11. Могут ли быть виртуальными конструкторы? Деструкторы?

Конструкторы виртуальными быть не могут, в отличие от деструкторов.

12. Наследуется ли спецификатор virtual?

Виртуальность наследуется.

13. Какое отношение устанавливает между классами открытое наследование?

- Открытое наследование является наследованием типа. Т. е. производный класс является частным случаем базового класса (если не происходит переопределение методов класса – для реализации принципа подстановки). Или производный класс может замещать методы базового класса (в том случае, если компонентные функции базового класса описаны со спецификатором virtual – для реализации полиморфизма).

14. Какое отношение устанавливает между классами закрытое наследование?

Производный имеет доступ ко всему функционалу, за исключением private компонентов, класса-предка, но ни наследники, ни экземпляры класса не имеют доступ к этому функционалу. Это нужно для того, чтобы средствами базового класса реализовать новый класс.

15. В чем заключается принцип подстановки?

- Если класс D является частным случаем класса B, то везде, где был использован класс B, можно использовать класс D. Принцип подстановки возможен, если не были переопределены методы базового класса.

16. Имеется иерархия классов:

class Student

{

int age;

public:

string name;

...

};

class Employee : public Student

{

protected:

string post;

...

};

class Teacher : public Employee

{

protected: int stage;

...

};

Teacher x;

Какие компонентные данные будет иметь объект х?

- Через объект x можно обратиться только к полю name. Но компонентные функции и друзья класса Teacher имеют доступ к protected полям string post и int stage;