Министерство образование Республики Беларусь

Учреждение образования «Полоцкий государственный университет»

Информационно-технологический факультет

Кафедра технологий программирования

Лабораторная работа №2 по теме «Наследования и виртуальные функции»

По дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Выполнил: Крутилин С.Д.

Группа 19-ИТ-1.

Проверил: Васильева Д. М.

Новополоцк

2021 г.

**Вариант 1**

**Цель работы**: Получить практические навыки создания иерархии классов и использования статических компонентов класса.

**Задание 1. Постановка задачи**

Для получения практических навыков создания иерархии классов и использования статических компонент, предложено реализовать демонстрационную программу, в которой создаются объекты различных классов и помещаются в список, после чего список просматривается. Необходимо выполнить следующие задания:

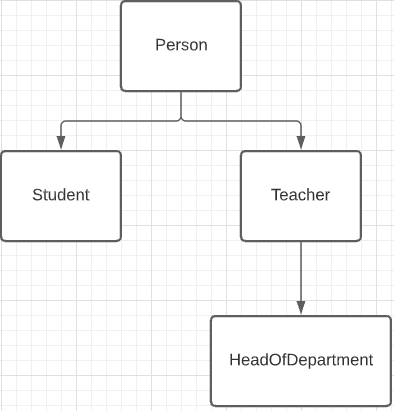
1. Определить иерархию классов (в соответствии с вариантом).
2. Определить в классе статическую компоненту - указатель на начало связанного списка объектов и статическую функцию для просмотра списка.
3. Реализовать классы.
4. Написать демонстрационную программу, в которой создаются объекты различных классов и помещаются в список, после чего список просматривается.
5. Сделать соответствующие методы не виртуальными и посмотреть, что будет.
6. Реализовать вариант, когда объект добавляется в список при создании, т.е. в конструкторе

**Задание 2. Определить иерархию классов**

Для моего варианта предложены следующие классы:

1. Студент
2. Преподаватель
3. Персона
4. Завкафедрой

Логично составить наследование, в котором персона является абстрактным классом, от которого наследуется студент и преподаватель, а от преподавателя наследуется завкафедрой, так как имеет ряд дополнительных возможностей.



**Задание 3. Определение пользовательских классов с комментариями.**

Определение абстрактного класса Person:

1. class Person
2. {
3. private:
4. // указатель на начало списка
5. static Person\* begin;
6. protected:
7. // поля класса
8. string firstName;
9. string lastName;
10. int age;
11. // указатель на предыдущий элемент
12. Person\* prev = NULL;
13. public:
14. // статический метод для отображения списка
15. // созданных объектов
16. static void print();
17. // все необходимые конструкторы
18. // по умолчанию
19. Person();
20. // копирования
21. Person(const Person& person);
22. // с параметрами
23. Person(string firstName, string lastName, int age);
24. // виртуальный деструктор
25. // нужен для корректного удаления объектов
26. virtual ~Person();
27. // абстрактный метод
28. // который обязует классов наследников
29. // реализовать самостоятельно
30. virtual void show() = 0;
31. // метод добавления в список
32. void add();
33. };

Определение класса Student:

1. class Student :
2. // наследуемся от абстрактного Person
3. virtual public Person
4. {
5. protected:
6. // поля класса
7. int averageScore;
8. int yearOfStudy;
9. public:
10. // конструкторы
11. Student();
12. Student(const Student& student);
13. Student(string firstName, string lastName, int age, int averageScore, int yearOfStudy);
14. // Деструктор
15. ~Student();
16. // Метод вывода информации о объекте
17. virtual void show();
18. };

Определение класса Teacher:

1. class Teacher :
2. // Наследуемся от Person
3. public Person
4. {
5. protected:
6. // свои поля класса
7. int experience;
8. bool hasVacation;
9. public:
10. // конструкторы
11. Teacher();
12. Teacher(const Teacher& teacher);
13. Teacher(string firstName, string lastName, int age, int experience, bool hasVacation);
14. // Деструктор
15. ~Teacher();
16. // метод вывода информации о объекте
17. virtual void show();
18. };

Определение класса HeadOfPerson:

1. class HeadOfDepartment :
2. // наследуемся от Teacher
3. public Teacher
4. {
5. protected:
6. // свои поля данных
7. int numberOfSubordinates;
8. public:
9. // конструкторы
10. HeadOfDepartment();
11. HeadOfDepartment(const HeadOfDepartment& headOfDeapartment);
12. HeadOfDepartment(string firstName, string lastName, int age, int experinece, bool hasVacation, int numberOfSubordinates);
13. // Деструктор
14. ~HeadOfDepartment();
15. virtual void show();
16. };

**Задание 4. Реализация конструкторов с параметрами и деструкторов.**

Конструктор Person:

1. // передаем данные и копируем их в поля класса
2. Person::Person(string firstName, string lastName, int age)
3. {
4. this->firstName = firstName;
5. this->lastName = lastName;
6. this->age = age;
7. this->add();
8. }

Конструктор Student:

1. Student::Student(const Student& student)
2. // часть переданных данных передаем
3. // конструктору родителя
4. : Person(student.firstName, student.lastName, student.age)
5. {
6. this->averageScore = student.averageScore;
7. this->yearOfStudy = student.yearOfStudy;
8. }

Конструктор Teacher:

1. Teacher::Teacher(const Teacher& teacher)
2. // часть переданных данных передаем
3. // конструктору родителя
4. : Person(teacher.firstName, teacher.lastName, teacher.age)
5. {
6. this->experience = teacher.experience;
7. this->hasVacation = teacher.hasVacation;
8. }

Конструктор HeadOfPerson:

1. HeadOfDepartment::HeadOfDepartment(string firstName, string lastName, int age, int experinece, bool hasVacation, int numberOfSubordinates)
2. // часть переданных данных передаем
3. // конструктору родителя
4. : Teacher(firstName, lastName, age, experience, hasVacation)
5. {
6. this->numberOfSubordinates = numberOfSubordinates;
7. }

**Задание 5. Реализация методов для добавления объектов в список.**

Основная идея данного списка в том, что если у нас список пустой, то просто переопределяем указатель на начало списка на переданный объект. Если список уже существует, то делаем так, чтобы текущий объект указывал на предыдущий, а ссылка на начало списка указывала на текущий объект.

1. void Person::add()
2. {
3. if (Person::begin == NULL) {
4. Person::begin = this;
5. }
6. else {
7. this->prev = Person::begin;
8. Person::begin = this;
9. }
10. }

**Задание 6. Реализация методов для просмотра списка.**

Основная идея в том, что абстрактный класс Person не содержит реализации метода show, поэтому метод должен быть реализован в классах потомках.

Метод для просмотра информации о объекте класса Student:

1. void Student::show()
2. {
3. cout << "----------------Student------------------" << endl;
4. cout << "First name: " << this->firstName << endl;
5. cout << "Last name: " << this->lastName << endl;
6. cout << "Age: " << this->age << endl;
7. cout << "Average score: " << this->averageScore << endl;
8. cout << "Year of study: " << this->yearOfStudy << endl;
9. cout << "-----------------------------------------" << endl;
10. }

Метод для просмотра информации о объекте класса Teacher:

1. void Teacher::show()
2. {
3. cout << "--------------Teacher--------------------" << endl;
4. cout << "First name: " << this->firstName << endl;
5. cout << "Last name: " << this->lastName << endl;
6. cout << "Age: " << this->age << endl;
7. cout << "Experience: " << this->experience << endl;
8. cout << "Has vacation: " << this->hasVacation << endl;
9. cout << "-----------------------------------------" << endl;
10. }

Метод для просмотра информации о объекте класса HeadOfPerson:

1. void HeadOfDepartment::show()
2. {
3. cout << "------------Head-of-department-----------" << endl;
4. cout << "First name: " << this->firstName << endl;
5. cout << "Last name: " << this->lastName << endl;
6. cout << "Age: " << this->age << endl;
7. cout << "Experience: " << this->experience << endl;
8. cout << "Has vacation: " << this->hasVacation << endl;
9. cout << "Number of subordinates" << this->numberOfSubordinates << endl;
10. cout << "-----------------------------------------" << endl;
11. }

И статический метод в абстрактном классе для просмотра списка:

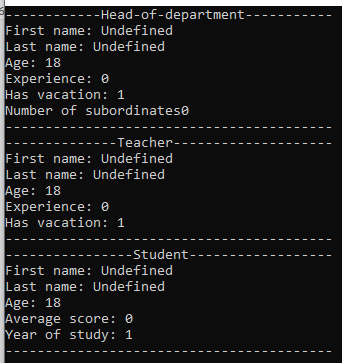
1. void Person::print()
2. {
3. Person\* temporaryPersonp = Person::begin;
5. // просто перебираем список
6. // и вызываем виртуальный метод show
7. while (temporaryPersonp != NULL) {
8. temporaryPersonp->show();
9. temporaryPersonp = temporaryPersonp->prev;
10. }
11. }

**Задание 7. Демонстрационная программа**

Демонстрационная программа выполняет подключения всех необходимых классов, а затем создает три объекта от каждого из класса кроме абстрактного Person. Затем выполняется вызов статического метода для выведения информации о каждом из объектов.

1. #include <iostream>
2. #include "Person.h"
3. #include "Student.h"
4. #include "Teacher.h"
5. #include "HeadOfDepartment.h"
6. int main()
7. {
8. Student\* first = new Student();
9. Teacher\* second = new Teacher();
10. HeadOfDepartment\* third = new HeadOfDepartment();
11. Person::print();
12. }

Так как все объекты созданы по умолчанию, то получаем следующий результат:



**Задание 8. О виртуальных методах.**

**Виртуальная функция** в языке С++ — это особый тип функции, которая, при её вызове, выполняет «наиболее» дочерний метод, который существует между родительским и дочерними классами. Это свойство еще известно, как **полиморфизм**. Дочерний метод вызывается тогда, когда совпадает сигнатура (имя, типы параметров и является ли метод константным) и тип возврата дочернего метода с сигнатурой и типом возврата метода родительского класса. Такие методы называются переопределениями (или «переопределенными методами»).