Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ
И РАДИОЭЛЕКТРОНИКИ (ТУСУР)
Кафедра компьютерных систем в управлении и проектировании (КСУП)

НАСЛЕДОВАНИЕ И ПОЛИМОРФИЗМ

Отчёт по лабораторной работе №5 по дисциплине
«Объектно-ориентированное программирование»

Студент группы 588-1
_____ / Чан Хыу Тхай
«_» _____2022 г.

Руководитель старший научный
сотрудник, доцент каф. КСУП
_____ / Горяинов А.Е.
«___» _____ 2022 г.

Томск 2022

1. Цель работы

Целью этой лабораторной работы является изучение наследования и полиморфизма, а также получение практических навыков решения связанных с ними задач объектно-ориентированного программирования.

2. Теоретические основы

Наследование — один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с инкапсуляцией, полиморфизмом и абстракцией), позволяющий описать новый класс на основе уже существующего (родительского), при этом свойства и функциональность родительского класса заимствуются новым классом.

Другими словами, класс-наследник реализует спецификацию уже существующего класса (базовый класс). Это позволяет обращаться с объектами класса-наследника точно так же, как с объектами базового класса.

Простое наследование:

Класс, от которого произошло наследование, называется базовым или родительским (англ. base class). Классы, которые произошли от базового, называются потомками, наследниками или производными классами (англ. derived class).

В некоторых языках используются абстрактные классы. Абстрактный класс — это класс, содержащий хотя бы один абстрактный метод, он описан в программе, имеет поля, методы и не может использоваться для непосредственного создания объекта. То есть от абстрактного класса можно только наследовать. Объекты создаются только на основе производных классов, наследованных от абстрактного. Например, абстрактным классом может быть базовый класс «сотрудник вуза», от которого наследуются классы «аспирант», «профессор» и т. д. Так как производные классы имеют общие поля и функции (например, поле «год рождения»), то эти члены класса могут быть описаны в

базовом классе. В программе создаются объекты на основе классов «аспирант», «профессор», но нет смысла создавать объект на основе класса «сотрудник вуза».

Множественное наследование

При множественном наследовании у класса может быть более одного предка. В этом случае класс наследует методы всех предков. Достоинства такого подхода в большей гибкости. Множественное наследование реализовано в C++. Из других языков, предоставляющих эту возможность, можно отметить Python и Эйфель. Множественное наследование поддерживается в языке UML.

Множественное наследование — потенциальный источник ошибок, которые могут возникнуть из-за наличия одинаковых имен методов в предках. В языках, которые позиционируются как наследники C++ (Java, C# и др.), от множественного наследования было решено отказаться в пользу интерфейсов. Практически всегда можно обойтись без использования данного механизма. Однако, если такая необходимость все-таки возникла, то, для разрешения конфликтов использования наследованных методов с одинаковыми именами, возможно, например, применить операцию расширения видимости — «::» — для вызова конкретного метода конкретного родителя.

Полиморфи́зм — возможность объектов с одинаковой спецификацией иметь различную реализацию.

Язык программирования поддерживает полиморфизм, если классы с одинаковой спецификацией могут иметь различную реализацию — например, реализация класса может быть изменена в процессе наследования[1].

Кратко смысл полиморфизма можно выразить фразой: «Один интерфейс, множество реализаций».

Полиморфизм — один из четырёх важнейших механизмов объектно-ориентированного программирования (наряду с абстракцией, инкапсуляцией и наследованием).

Полиморфизм позволяет писать более абстрактные программы и повысить коэффициент повторного использования кода. Общие свойства объектов объединяются в систему, которую могут называть по-разному — интерфейс, класс. Общность имеет внешнее и внутреннее выражение:

- внешняя общность проявляется как одинаковый набор методов с одинаковыми именами и сигнатурами (именем методов и типами аргументов и их количеством);
- внутренняя общность — одинаковая функциональность методов. Её можно описать интуитивно или выразить в виде строгих законов, правил, которым должны подчиняться методы.

Формы полиморфизма

Используя Параметрический полиморфизм можно создавать универсальные базовые типы. В случае параметрического полиморфизма, функция реализуется для всех типов одинаково и таким образом функция реализована для произвольного типа. В параметрическом полиморфизме рассматриваются параметрические методы и типы.

Параметрические метод

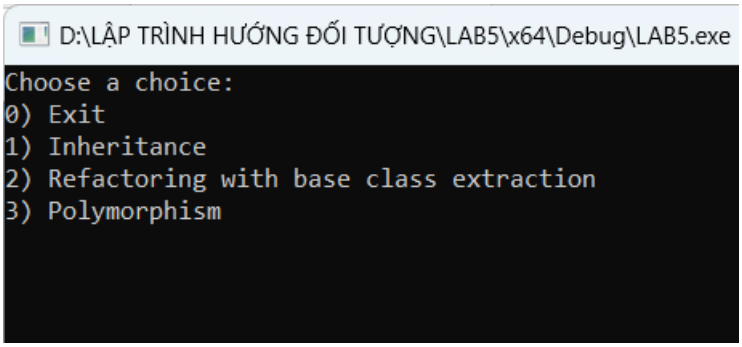
Если полиморфизм включения влияет на наше восприятие объекта, то параметрический полиморфизм влияет на используемые методы, так как можно создавать методы родственных классов, откладывая объявление типов до времени выполнения. Для во избежание написания отдельного метода каждого типа применяется параметрический полиморфизм, при этом тип параметров будет являться таким же параметром, как и операнды...

Параметрические типы.

Вместо того, чтобы писать класс для каждого конкретного типа следует создать типы, которые будут реализованы во время выполнения программы то есть мы создаем параметрический тип.

3. Ход работы

Ниже представлен интерфейс лабораторной работы:



В первой задаче нужно создать класс Person со строковыми полями имени, фамилии, отчества. Создайте соответствующие сеттеры, геттеры и конструктор класса (рис. 1).

```
class PersonNode
{
private:
    string name;
    string surname;
    string middlename;
public:
    string  getName();
    string  getSurname();
    string  getMiddleName();
    void  setName();
    void  setSurname();
    void  setMiddleName();
    PersonNode();
    PersonNode(string name, string surename, string middlename);
};
```

Рис. 1 – Создать класс Person

В второй задаче нужно создать дочерний от класса Person класс Student с дополнительными полями номера зачетной книжки и года поступления. Создайте соответствующие сеттеры, геттеры и конструктор класса. Конструктор дочер-него класса должен быть основан на наследовании от конструктора базового класса (рис. 2).

```
class StudentNode : public PersonNode
{
private:
    string number;
    int year;
public:
    string getNumber();
    int getYear();
    void setYear();
    void setNumber();
    void setYear(int i);
    void setNumber(string s);
    StudentNode();
    bool checkYear();
    StudentNode(string name, string surename, string patronymic, string number, int year);
};
```

Рис. 2 – Создать класс Student

В второй задаче нужно создать дочерний от класса Person класс Teacher с дополнительными полем должности. Создайте со-ответствующие сеттеры, геттеры и конструктор класса. Конструктор дочернего класса должен быть основан на наследовании от конструктора базового класса (рис. 3).

```
class TeacherNode : public PersonNode
{
private:
    string position;
public:
    string getPosition();
    void setPosition();
    void setPosition(string s);
    TeacherNode();
    TeacherNode(string name, string surename, string patronymic, string position);
};
```

Рис. 3 – Создать класс Teacher

Следующая задача — нужно создать функцию ShowName(Person* person), принимающую указатель на базовый класс. Функция при-нимает на вход объект типа Person (или объект производного класса), и выводит на экран фамилию, имя и отчество. Пример вывода (рис. 4).

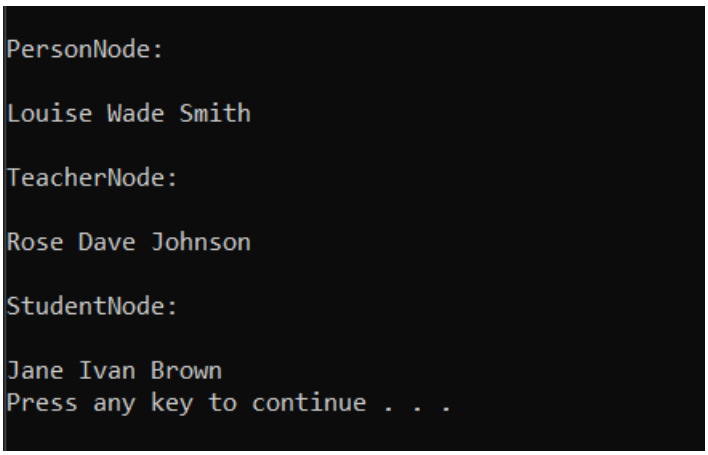


Рис. 4 – Пример вывода

Рефакторинг с выделением базового класса:

Работа разработчика неразрывно связана с умением читать чужой код, находить в нём ошибки проектирования или ошибки работы, и исправлять их.

В следующей задаче необходимо заменить использование ранее существовавших функций в main() вызовом новой универсальной функции Login(). Также необходимо объединить массив пользователей программы в единый массив всех пользователей через массив указателей на базовый класс (рис. 6) и Результатом работы функции main() показан на рисунке 5.

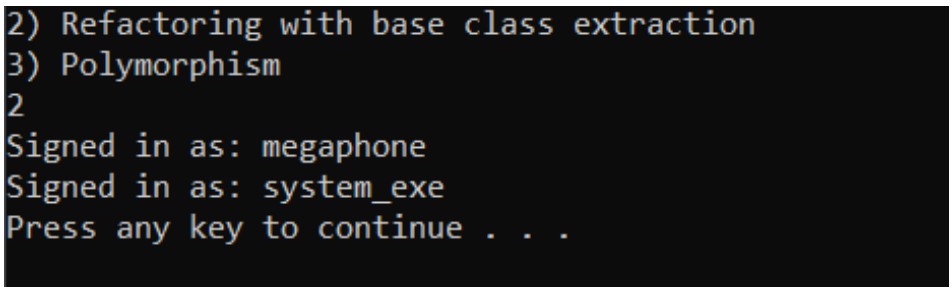


Рис. 5 – Результатом работы функции main()

```
User** users = new User * [4]
{
        new User(100000, "john1995", "1995john"),
        new User(100001, "ilon_mask", "X æ A-12"),
        new User(100002, "megaphone", "password"),
        new User(100003, "yogurt", "ksTPQzSu"),
};
PaidUser** paidUsers = new PaidUser * [4]
{
    new PaidUser(200000, "TheKnyazz", "JHPzPGFG"),
        new PaidUser(200001, "system_exe", "UgfkDGmU"),
        new PaidUser(200002, "RazorQ", "TBgRnbCP"),
        new PaidUser(200003, "schdub", "CetyQVID"),
};
string login = "megaphone";
string password = "password";
for (int i = 0; i < 4; i++)
{
    if (users[i]->Login(login, password))
    {
        cout << "Signed in as: " << users[i]->GetLogin() << endl;
    }
}
login = "system_exe";
password = "UgfkDGmU";
for (int i = 0; i < 4; i++)
{
    if (paidUsers[i]->Login(login, password))
    {
        cout << "Signed in as: " << paidUsers[i]->GetLogin() << endl;
    }
}
```

Рис. 6 – Фрагмент кода программы

Полиморфизм:

Следующая программа должна реализовать систему скидок в приложении для магазинов. Скидки действуют на товары определенной категории. Есть скидки процентные (скидка предоставляется в заданном размере от стоимости товара определенной категории), и есть скидки сертификатные (скидка предоставляется на за-данную в сертификате сумму, но не превышающую стоимость товара).

Необходимо создать функцию void ShowCheckWithDiscount() принимающую указатель на базовый класс скидки, а также массив товаров. Функция должна пе-

ребрать в цикле все товары массива, и рассчитать для них скидку через экземпляр discount.

(рис. 7).

Для каждого товара на экран выводится его название, старая стоимость и новая стоимость. После цикла программа вы-водит на экран общую стоимость всех товаров с учетом скидки. Пример вывода на экран (рис. 8).

```
void ShowCheckWithDiscount(DiscountBase* discount, Product* products, int productsCount)
{
    double allPrice = 0;
    for (int i = 0; i < productsCount; i++)
    {
        allPrice = allPrice + products[i].GetPrice();
    }

    for (int i = 0; i < productsCount; i++)
    {
        cout << "Old item price " << products[i].GetName() << ": " << products[i].GetPrice() << endl;
        products[i].SetPrice(discount->Calculate(&products[i]));
        cout << "Price when discount is applied: " << products[i].GetPrice() << endl;
    }

    double allPriceWithDiscount = 0;
    for (int i = 0; i < productsCount; i++)
    {
        allPriceWithDiscount = allPriceWithDiscount + products[i].GetPrice();
    }
    cout << "The price of the entire list of goods without discount: " << allPrice << endl;
    cout << "The price of the entire list of goods, taking into account the discount: " << allPriceWithDiscount << endl;
    cout << "The discount was: " << allPrice - allPriceWithDiscount << endl;
}
```

Рис. 7 – Функция void ShowCheckWithDiscount()

```
Old item price Chicken: 250
Price when discount is applied: 250
Old item price Cheese: 55
Price when discount is applied: 55
Old item price Apple: 300
Remaining amount of the certificate 200
Price when discount is applied: 0
Old item price Milk: 80
Price when discount is applied: 80
The price of the entire list of goods without discount: 685
The price of the entire list of goods, taking into account the discount: 385
The discount was: 300
Choose one of the proposed items!
Press any key to continue . . .
```

Рис. 8 – Отображаются результаты

4. Вывод

В ходе работы в лаборатории были выполнены все задачи, разработана природа «Наследование и полиморфизм» в объектно-ориентированном программировании и набор функций для работы с ней.