ФГБОУ ВО “Чувашский государственный университет им. И. Н. Ульянова”
Факультет: ИВТ
Кафедра: Вычислительной техники
Предмет: Объектно-ориентированное программирование

Лабораторная работа №7

Исследование. Полиморфизм

Выполнил: студент группы ИВТ-41-20
Галкин Дмитрий
Проверил: доцент Обломов И.А.

Чебоксары 2022 г.

Теория

Класс – абстрактный тип данных, определяемый пользователем и предоставляющий собой модель реального мира в виде данных и функций для работы с ними.

Спецификаторы доступа:

- *private* – запрещает обращение к полям класса из вне
- *public (обязательный спецификатор)* – объявляет поля как общедоступные

Специфические особенности:

- Могут иметь любой тип, кроме типа объявляемого класса, но могут быть указателями или ссылками на данный класс;
- Могут быть объявлены с модификатором const, при этом инициализируются с помощью конструкторов специального вида
- Могут быть объявлены с модификатором static, но никак не auto, register или extern.

Индивидуальное задание

1. Объявить базовый и производный классы, моделирующие слово (базовый класс) и строку (производный класс). В базовом классе предусмотреть поле для хранения одного слова произвольной длины (можно использовать тип string), а в производном, кроме наследованного слова, необходимо объявить поле, содержащее количество слов в строке. Для классов определить методы, обеспечивающие ввод слов и строк и вывод их на экран. Проанализировать случаи приватного, защищенного и обобществленного наследования.

```
class Base {
    protected:
        std::string word;
public:
    static std::string stat_word;
    void show() {
        std::cout << "Word: " << word << std::endl;
    }
};
class Derived_public: public Base {
    int count_word_in_line;

public:
    Derived_public(std::string word): Base(word) {
        std::stringstream stream(word);
        std::string oneWord;
        count_word_in_line = 0;

        while (stream >> oneWord) {++count_word_in_line;}
    }

    void show() {
        std::cout << "\t Derived (public)" << std::endl;
        Base::show();
        std::cout << "Count words: " << count_word_in_line << std::endl << std::endl;
    }
};
class Derived_protected: protected Base {
    int count_word_in_line;

public:
    Derived_protected(std::string word): Base(word) {
        std::stringstream stream(word);
```

```
        std::string oneWord;
        count_word_in_line = 0;

        while (stream >> oneWord) {++count_word_in_line;}
    }

    void show() {
        std::cout << "\t Derived (protected)" << std::endl;
        std::cout << "Word: " << word << std::endl; // Если поле в области protected,
или если поле private
                                        // Base::show()
        std::cout << "Count words: " << count_word_in_line << std::endl << std::endl;
    }
};

class Derived_private: private Base {
    int count_word_in_line;

public:
    Derived_private(std::string word): Base(word) {
        std::stringstream stream(word);
        std::string oneWord;
        count_word_in_line = 0;

        while (stream >> oneWord) {++count_word_in_line;}
    }

    void show() {
        std::cout << "\t Derived (private)" << std::endl;
        std::cout << "Word (private): " << word << std::endl; // только если поле в
области protected
        Base::show();
        std::cout << "Count words: " << count_word_in_line << std::endl << std::endl;
    }
};

{
    example_show(1);

    // Public
    Derived_public derived_public("Hello StarBacks!");
    derived_public.show();

    // Protected
    Derived_protected derived_protected("Hello StarBacks!");
    derived_protected.show();

    // Private
    Derived_private derived_private("Hello StarBacks!");
    derived_private.show();
}
```

2. В конструкторах базового и произвольного классов обеспечить вывод в стандартный поток сообщения, идентифицирующего принадлежность объекта тому или иному классу. Проанализировать последовательность активизации конструкторов при объявлении объектов производного класса. Дополнить классы деструкторами, оценить порядок их активации.

```
Base(std::string word) {
    std::cout << "Start construct Base" << std::endl;
    this->word = word;
}
```

```
void show() {
    std::cout << "Word: " << word << std::endl;
}

~Base() {
    std::cout << "End Base" << std::endl;
}

class Derived: Base {

public:

    int count_word_in_line;

    Derived(std::string word): Base(word) {
        std::cout << "Start construct Derived" << std::endl;
        std::stringstream stream(word);
        std::string oneWord;
        count_word_in_line = 0;

        while (stream >> oneWord) {++count_word_in_line;}
    }

    ~Derived() {
        std::cout << "End Derived" << std::endl;
    }
};
```

3. Дополните объявление класса слово статическим полем, приведите пример его использования объектами и методами произвольного класса строка. Самостоятельно разобрать и провести пример наследования константных и ссылочных полей.

```
std::string getSurName() const;
std::string getName() const;
std::string getPatronymic() const;
std::string getFullName() const;
int getAge() const;
std::string isSex() const;
std::string getStatus() const;
Finance getFinance() const;
```

4. В классе слово определить дружественную функцию, выводящую отдельное слово в стандартный поток. Перезагрузите аналогичную функцию в классе строка. Приведите примеры использования дружественных функций.

```
friend std::ostream &operator <<(std::ostream &, const Base &);

std::ostream &operator <<(std::ostream &out, const Base &b) {
    out << "Func friend Base: " << b.word << std::endl;
    return out;
}

std::ostream &operator <<(std::ostream &out, const Derived &d) {
    out << "Func friend Derived: " << d.count_word_in_line << std::endl;
    return out;
}

{
    cout << endl;
    example_show(4);
    Base base("Hello my");
    Derived derived("Hello my friends");
```

```
    cout << base;
    cout << derived;
}
```

5. Создать иерархию классов, показать пример наследования переменной this объектами производных классов.

```
class One {
    int one_1, one_2;

public:
    One(int o_1, int o_2):one_1(o_1), one_2(o_2) {};
    int Get() {
        std::cout << "address is object: " << this << ' ';
        return one_1;
    }
};

class Two_1 : public One {
    long two_1;
    public:
        Two_1(int o_1, int o_2): One(o_1, o_2){};
};

class Two_2 : public One {
    char two_2;
    public:
        Two_2(int o_1, int o_2): One(o_1, o_2){};
};

class All: public Two_1, public Two_2 {
    public:
        All(int o_1, int o_2, int o_3, int o_4): Two_1(o_1, o_2), Two_2(o_3, o_4){};
};

{
    cout << endl;
    example show(5);
    All var(1, 2, 3, 4);

    cout << "meaning 1: " << var.Two_1::Get() << endl;
    cout << "meaning 1: " << var.Two_2::Get() << endl;
}
```

6. Разработать иерархию классов «человек, служащий, студент», в которой класс человек имеет поля имя, фамилия, возраст. Класс служащий дополняет его полем специальность, а класс студент – полями группа и средний балл. Предусмотреть полиморфные методы, позволяющие получить информацию о субъекте в зависимости от его типа.

```
class People {
    std::string name;
    std::string surname;
    int age;

    public:

    People(const std::string &name, const std::string &surname, int age) : name(name),
surname(surname), age(age) {}

    People() {}

    void Out() {
            std::cout << "People { \nname = " << name
```

```
                        << "\nsurname = " << surname
                        << "\nage = " << age << " }" << std::endl;
        }
};

class Employee : public People, AbstractPeople {
    std::string speciality;

    public:

    Employee(const std::string &name, const std::string &surname, int age, const
std::string &speciality) : People(name,

surname,

age),

speciality(

speciality) {}
    void Out()  {
            std::cout << "Employee { \nspeciality = " << speciality << " }" <<
std::endl;
        }

};

class Students : public People, AbstractPeople {
    std::string group;
    double average_score;

    public:

    Students(const std::string &name, const std::string &surname, int age, const
std::string &group,
             double averageScore) : People(name, surname, age), group(group),
average_score(averageScore) {}

    void Out()  {
            std::cout << "Students { \ngroup = " << group
                        << "\naverage_score = " << average_score << " }" << std::endl;
        }
};
```

7. В задании №6 класс человек определить как абстрактный. Привести примеры использования абстрактного класса.

```
class AbstractPeople {
    virtual void Out() = 0;
};
```

8. В иерархии «человек, служащий, студент» преобразовать указатель на класс человек в указатель на класс служащий и студент. Показать результат работы.

```
{
    cout << endl;
    example_show(8);
    People *ptr_people = new People("Dima", "Galkin", 19);
    Employee *ptr_employee = new Employee("Dima", "Galkin", 19, "Programmer");
    Students *ptr_students = new Students("Dima", "Galkin", 19, "ivt-41-21", 4.5);

    fun_employee(ptr_people);
    ptr_people = ptr_employee;
    fun_employee(ptr_people);
```

```
    fun_employee(ptr_employee);
    ptr_people = ptr_students;
    fun_students(ptr_people);

}
```

9. В соответствие с заданием №7 обеспечить понижающее преобразование ссылок класса человек в ссылку служащий, ссылки на класс служащий в ссылку на класс студент.

```
{
    cout << endl;
    example_show(9);

    People ptr_people("Dima", "Galkin", 19);
    Employee ptr_employee("Dima", "Galkin", 19, "Programmer");
    Students ptr_students("Dima", "Galkin", 19, "ivt-41-21", 4.5);

    Employee &employee_ref = ptr_employee;
    People &people_ref = employee_ref;
    dynamic_cast<Employee &>(people_ref).Out();

    Students &students_ref = ptr_students;
    Employee &employee_1ref = reinterpret_cast<Employee &>(students_ref);
    dynamic_cast<Students &>(employee_1ref).Out();
}
```

Вывод: Я практическим путем выяснил как использовать указатели на практики и познакомился с базовыми принципами наследования в ООП на языке C++.