

Отчет по л/р 1.

предмет «Языки и методы программирования»

Король Екатерина Юрьевна| ПИ 2-2 |03.07.2021

# Предмет лабораторной работы

Создание программы, реализующей последовательный механизм наследования на основании пошагового расширения классов при помощи механизмов наследования в языке с++.

## Начальный Класс

В данной лабораторной работе у вас имеется начальный класс, описание которого задано первой часть вашего задания будет создать код реализующий этот класс и проверить его работоспособность на примере конструктора по умолчанию, когда вы убедитесь в работоспособности исходного класса, добавьте к публичным методам класса два метода-сеттера, пожалуйста. Эти методы должны задавать значения для h\_name и h\_age.

После этого добавьте в класс Harbinger ещё один private-элемент типа std::string, назовите эту строку CatchPhrase. Напишите геттер и сеттер для неё, и добавьте новый конструктор с тремя аргументами, для каждого из которых будет значение по умолчанию.

Для отчета вставьте получившийся код ниже этой надписи, но до следующего раздела.

#include <iostream>

#include <string>

using namespace std;

class Harbinger

{

private:

string h\_name;

string CatchPhrase;

int h\_age;

public:

Harbinger(string name = "", int age = 0) : h\_name(name), h\_age(age) // конструктор Harbinger

{

}

Harbinger(string name, int age, string CatchPhrase1)

{

h\_name = name;

h\_age = age;

CatchPhrase = CatchPhrase1;

}

void setName(string name) { h\_name = name; } // сеттеры Harbinger

void setAge(int age) { h\_age = age; }

void setCatchPhrase(string CatchPhrase1) { CatchPhrase = CatchPhrase1; }

string getCatchPhrase() const { return CatchPhrase; } // геттеры Harbinger

string getName() const { return h\_name; }

int getAge() const { return h\_age; }

};

## Первое наследование

На втором этапе от класса Habringer необходимо унаследовать класс Prophet. Очень важно правильно выбрать модификатор наследования таким образом, чтобы все члены- переменные и методы класса предка не изменили своего модификатора доступа, будучи унаследованными в дочернем классе. (обратите внимание на то, как разные типы модификаторов наследования влияют на модификаторы доступа). В классе Prophet, в свою очередь, нужно создать массив из 3 строк, назвать его MagnumOpus , и написать геттер и сеттер для этого массива. Также у класса Phophet должны быть собственные члены-переменные (с геттерами и сеттерами) ph\_name(std::string), ph\_age(int) и ph\_call(std::string). В ph\_call всегда должна храниться самая длинная строка из массива MagnumOpus, и, как только какая-то строка из массива MagnumOpus после переопределения становится самой длинной, в ph\_call тут же должна записываться эта строка. Длину всех строк ограничить 25 символами, при вводе в сеттере проверять чтобы задаваемая строка не была слишком длинной. Проверьте при помощи создания объекта класса Phophet. (Код с комментариями вставить ниже для отчета):

class Prophet : public Harbinger // наследование с модификатором доступа public

{

string ph\_name;

string ph\_call;

int ph\_age;

string\* MagnumOpus;

public:

Prophet(string name = "", int age = 0, string call = "") : ph\_name(name), ph\_age(age), ph\_call(call) // Prophet

{

MagnumOpus = new string[3];

}

void max() // поиск самой длинной строки в MagnumOpus для записи в ph\_call

{

int g;

for (size\_t i = 0; i < 2; i++)

{

g = MagnumOpus[i].length();

if (g <= MagnumOpus[i + 1].length())

{

ph\_call = MagnumOpus[i + 1];

}

}

}

void setMagnumOpus(string Magnum[]) // сеттер MagnumOpus

{

for (size\_t i = 0; i < 3; i++)

{

if (Magnum[i].length() <= 25) // проверка длины вводимой строки

{

MagnumOpus[i] = Magnum[i];

}

}

max();

}

const string\* getMagnumOpus() { return MagnumOpus; } // геттер MagnumOpus

void setpName(string name) { ph\_name = name; }

void setpAge(int age) { ph\_age = age; }

void setpCall(string call) { ph\_call = call; }

string getpName() const { return ph\_name; }

int getpAge() const { return ph\_age; }

string getpCall() const { return ph\_call; }

};

## Второе наследование

Слово Harbinger переводится как «Предвестник», слово Prophet это «Пророк». Во время второго наследования мы создадим класс DeusEx («Бог из…»). Класс DeusEx будет публично наследовать класс Prophet. У нового класса появятся элементы-переменные gd\_name(std::string), gd\_year(int). Эти два элемента единожды задаются при помощи конструктора как значения по умолчанию. Переменная gd\_year хранит в себе четырёхзначное число, отражающая какой-то год в диапазоне от 0000 до 9999. В класс DeusEx нужно добавить публичный метод, который бы последовательно строка за строкой выводил в консоль:

(из класса Harbinger):

«*h\_name* в возрасте *h\_age* говорил, что *CatchPhrase.*»

(две строки кроме ph\_call из класса Prophet)

«Первое пророчество: *MagnumOpus[1].*»

«Второе пророчество: *MagnumOpus[2].*»

(ph\_call из класса Prophet)

«Главное пророчество: *ph\_call.*»

(ph\_name и ph\_age из класса Prophet)

«Так говорил *ph\_name* в возрасте *ph\_age.*»

(gd\_name и gd\_year из класса DeusEx)

«В год gd\_year пришёл gd\_name, которого предрекали h\_name и ph\_name».

(Приведите получившийся код ниже для отчета, пожалуйста)

class DeusEx : public Prophet // наследование DeusEx-ом Prophet-а с модификатором доступа public

{

string gd\_name;

int gd\_year;

public:

DeusEx(string name = "Будда", int year = 1001) : gd\_name(name), gd\_year(year) // конструктор по умолчанию

{

}

void setgYear(int year) { gd\_year = year; } // сеттеры

void setgName(string name) { gd\_name = name; }

int getgYear() const { return gd\_year; } // геттеры

string getgName() const { return gd\_name; }

void print()

{

cout << getName() << " в возрасте " << getAge() << " говорил, что " << getCatchPhrase() << endl;

cout << "Первое пророчество: " << getMagnumOpus()[0] << endl;

cout << "Второе пророчество: " << getMagnumOpus()[1] << endl;

cout << "Главное пророчество: " << getpCall() << endl;

cout << "Так говорил " << getpName() << " в возрасте " << getpAge() << endl;

cout << "В год " << gd\_year << " пришел " << gd\_name << ", которого предракали " << getName() << " и " << getpName() << endl;

}

};

## Последнее наследование

После того как мы создали предвестника, пророка с заповедями и бога перейдем к верующему. Создайте класс Believer, унаследуйте его приватно от DeusEx, добавьте члены-переменные bv\_name(std::string), bv\_age(int). Геттер и сеттер для них обоих. Конструктор без значений по умолчания, но с определением значения переменных в теле конструктора через два аргумента. И один публичный метод с выводом «Верующий *bv\_name* молится *gd\_name* именем *ph\_name*. C *gd\_year* и по сей день.»

(Ниже приведите полный код со всеми классами и функцией main, которая позволит продемонстрировать функциональность созданных классов.)

И, пожалуйста, не забывайте про комментарии.

class Believer : private DeusEx // наследование Believer-ом DeusEx c модификатором private

{

string bv\_name;

int bv\_age;

public:

Believer(string name, int age) // конструктор

{

bv\_name = name;

bv\_age = age;

}

void printBeliever(DeusEx deusex, Prophet prophet)

{

cout << "Верующий " << getbName() << " молится " << deusex.getgName() << " именем " << prophet.getpName() << ". С " << deusex.getgYear() << " и по сей день." << endl;

}

void setbAge(int age) // сеттеры Believer

{ bv\_age = age; }

void setbName(string name) { bv\_name = name; }

int getbAge() const { return bv\_age; } // геттеры Believer

string getbName() const { return bv\_name; }

};

int main()

{

setlocale(LC\_ALL, "RUS");

Harbinger harbing("Илья", 19, " шишка "); // переменные классов

Prophet prophe("Печаль", 5);

DeusEx deus("Метакритик", 2029);

Believer vas("Кто-то", 7);

deus.setCatchPhrase(" не просил об этом"); // замена данных через наследование

deus.setAge(30);

deus.setName("Адам");

deus.setpName("Дженсен");

deus.setpAge(35);

deus.setpCall("пуля");

string Mass[3] = { "Square Enix", "закрыли", "DeusEx ов больше не будет" };

deus.setMagnumOpus(Mass); // заполнение OpusMagnum

deus.print();

vas.printBeliever(deus, prophe);

}