

Отчет по л/р 1.

предмет «Языки и методы программирования»

Алешин Иван Сергеевич | ПИ2-1 | 26.04.2021

# 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.

#include <iostream>

#include <string>

class Harbinger

{

private:

std::string h\_name;

int h\_age;

public:

Harbinger(std::string name = "", int age = 0)

: h\_name(name), h\_age(age)

{

}

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

int getAge() const { return 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.

class Harbinger

{

private:

std::string h\_name;

int h\_age;

string CatchPhrase;

public:

Harbinger(string name = "", int age = 0, string phrase = "")

{

h\_name = name;

h\_age = age;

CatchPhrase = phrase;

}

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

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

void setName(string name) {

h\_name = name;

}

void setAge(int age) {

h\_age = age;

}

string getCatchPhrase() const {

return CatchPhrase;

}

void setCatchPhrase(string CatchPhrase) {

this->CatchPhrase = CatchPhrase;

}

};

## 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 {

private:

string ph\_name, ph\_call, strN;

int ph\_age;

string MagnumOpus[3];

public:

Prophet(string name = "", int age = 0, string call = "") {

ph\_name = name;

ph\_age = age;

ph\_call = call;

} /\* Добавляем конструктор для инициализации\*/

void setNAme(string name) {

if (name.length() > 24) {

cout << "Ошибка";

cin >> strN;

setNAme(strN);

}

else if (name.length() > ph\_call.length()) {

ph\_name = ph\_call;

ph\_call = name;

}

else ph\_name = name;

}/\* Сеттер для ввода строк и их сравнивания. Сразу проверяется длина строки, если больше заданной длины, то просит ввести новую строку.\*/

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

void setAGe(int age) {

ph\_age = age;

}

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

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

void setMagnum(int n, string str) {

if (str.length() > 24) {

cout << "Ошибка" << endl;

}

else { MagnumOpus[n] = str; }

if (str.length() > ph\_call.length()) {

dte = str;

}

pelse { ph\_name = str; }

} /\* сеттера для массива, аналогичен сеттера для ph\_name

string getMagnum(int i) {

return MagnumOpus[i];

}

void sort () {

string a ="";

for (int i = 0; i < 2; i++) {

if (MagnumOpus[i] > MagnumOpus[i+1]) {

a = MagnumOpus[i];

MagnumOpus[i] = MagnumOpus[i+1];

MagnumOpus[i+1] = a;

}

}

}

}; /\* сортировка массива

## 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 {

private:

string gd\_name;

int gd\_year;

public:

DeusEx(string name = "Иисус", int year=(rand()%10000)) {

gd\_name = name;

gd\_year = year;

}

void forecast() {

sort (); /\* функция из предыдущего класса для сортировки \*/

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

cout << "Первое пророчество : " << getMagnum(0) << endl;

cout << "Второе пророчество: " << getMagnum(1) << endl;

cout << "Главное пророчество: " << getMagnum(2) << endl;

cout << "Так говорил " << getNAme() << " в возрасте " << getAGe() << endl;

cout << "В год " << gd\_year << " пришёл " << gd\_name << ", которого предрекали " << getName() << " и " << getNAme() << 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.

#include <iostream>

#include <string>

using namespace std;

class Harbinger

{

private:

  std::string h\_name;

  int h\_age;

  string CatchPhrase;

public:

  Harbinger(string name = "", int age = 0, string phrase = "")

  {

    h\_name = name;

    h\_age = age;

    CatchPhrase = phrase;

  }

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

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

  void setName(string name) {

    h\_name = name;

  }

  void setAge(int age) {

    h\_age = age;

  }

  string getCatchPhrase() const {

    return CatchPhrase;

  }

  void setCatchPhrase(string CatchPhrase) {

    this->CatchPhrase = CatchPhrase;

  }

};

class Prophet : public Harbinger {

private:

  string ph\_name, ph\_call, strN;

  int ph\_age;

  string MagnumOpus[3];

public:

  Prophet(string name = "", int age = 0, string call = "") {

    ph\_name = name;

    ph\_age = age;

    ph\_call = call;

  }

  void setNAme(string name) {

    if (name.length() > 24) {

      cout << "Ошибка";

      cin >> strN;

      setNAme(strN);

    }

    else if (name.length() > ph\_call.length()) {

      ph\_name = ph\_call;

      ph\_call = name;

    }

    else ph\_name = name;

  }

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

  void setAGe(int age) {

    ph\_age = age;

  }

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

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

  void setMagmum(int n, string str) {

    if (str.length() > 24) {

      cout << "Ошибка" << endl;

    }

    else { MagnumOpus[n] = str; }

    if (str.length() > ph\_call.length()) {

      ph\_call = str;

    }

    else { ph\_name = str; }

  }

  string getMagnum(int i) {

    return MagnumOpus[i];

  }

  void sort () {

    string a ="";

    for (int i = 0; i < 2; i++) {

      if (MagnumOpus[i] > MagnumOpus[i+1]) {

        a = MagnumOpus[i];

        MagnumOpus[i] = MagnumOpus[i+1];

        MagnumOpus[i+1] = a;

      }

    }

  }

};

class DeusEx : public Prophet {

private:

  string gd\_name;

  int gd\_year;

public:

  DeusEx(string name = "Иисус", int year=(rand()%10000)) {

    gd\_name = name;

    gd\_year = year;

  }

  void forecast() {

    sort ();

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

    cout << "Первое пророчество : " << getMagnum(0) << endl;

    cout << "Второе пророчество: " << getMagnum(1) << endl;

    cout << "Главное пророчество: " << getMagnum(2) << endl;

    cout << "Так говорил " << getName() << " в возрасте " << getAGe() << endl;

    cout << "В год " << gd\_year << " пришёл " << gd\_name << ", которого предрекали " << getName();

  }

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

  int getGd\_year() const { return gd\_year; }

void bonus () {cout << getName () << endl;}

};

class Believer : private DeusEx {

private:

  string bv\_name;

  int bv\_age;

public:

  Believer(string name, int age) {

    bv\_name = name;

    bv\_age = age;

  }

  void setBv\_name(string name) {

    bv\_name = name;

  }

  void setBv\_age(int age) {

    bv\_age = age;

  }

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

  int getBv\_age() const { return bv\_age; }

  void output() {

cout << " и " << bv\_name << endl;

    cout << "Верующий " << bv\_name << " молится " << getGd\_name()<< " именем " ;

  }

void output2 () { cout << "С " << getGd\_year() << " и по сей день";}

/\* Для вызова gd\_name и gd\_year создаём геттеры в родительском классе.\*/

};

int main() {

setlocale(LC\_ALL, "Russian");

DeusEx first = DeusEx();

first.setName("John");

first.setAge(32);

first.setCatchPhrase("You know!");

first.setMagmum(0, "All people will know!");/\* Заполнение массива \*/

first.setMagmum(1, "We will become better!");

first.setMagmum(2, "The world will change");

first.setAGe(28);/\* Добавляет возраст в переменную ph\_age \*/

first.forecast();

Believer F = Believer("Anna", 22);

F.output();

first.bonus();

F.output2 ();

return 0;

}