Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Уфимский государственный авиационный технический университет»

Факультет информатики и робототехники

Кафедра Вычислительной математики и кибернетики

**Лабораторная работа №5**

По дисциплине «Объектно-ориентированное программирование»

«Виртуальность»

Выполнил: студент группы ПРО-222

Вальшин Руслан Ильдусович

УФА 2020

1. **Цель работы:**

Определение и реализация тестовых классов, и написание программы, иллюстрирующей их использование.

1. **Задачи:**

* Определения
* перекрываемых методов (показать на примере случай, когда вызывается перекрываемый метод, а когда наследуемый)
* виртуальных методов, в т.ч. деструкторов (показать на примере, когда вызывается наследуемый виртуальный метод, а когда базовый)
* Реализация
* проверки на принадлежность некоторому классу
* безопасного приведения типов (dynamic\_cast)
* безопасного приведения типов (вручную)
* Передача объектов как параметров в функции

1. **Ход работы:**

Определение класса-предка Animal

class Animal

{

protected:

int age;

int weight;

string name;

public:

Animal()

{

age = 0;

weight = 0;

name = "";

printf("Animal()\n");

}

Animal(int age, int weight, string name = "")

{

this->age = age;

this->weight = weight;

this->name = name;

printf("Animal(int age, int weight, string name)\n");

}

Animal(const Animal& animal)

{

this->age = animal.age;

this->weight = animal.weight;

this->name = animal.name;

printf("Animal(const Animal& animal)\n");

}

int getAge()

{

return age;

}

int getWeight()

{

return weight;

}

string getName()

{

return name;

}

void sound()

{

printf("Animal хотел что-то сказать\n");

}

virtual void sound2()

{

printf("Animal попытался издать другой звук\n");

}

virtual string className()

{

return "Animal";

}

virtual bool isA(string className)

{

return className == "Animal";

}

virtual ~Animal()

{

printf("~Animal()");

}

};

Определение класса-потомка WildAnimal

class WildAnimal :public Animal

{

public:

WildAnimal()

{

printf("WildAnimal()\n");

}

WildAnimal(int age, int weight, string name = "")

:Animal(age, weight, name)

{

printf("WildAnimal(int age, int weight, string name = "")\n");

}

void sound()

{

printf("Raaaaarrrr\n");

}

void sound2()

{

printf("R-r-r-r-r-r\n");

}

void chase()

{

printf("Дикое животное начало погоню\n");

}

string className()

{

return "WildAnimal";

}

bool isA(string className)

{

return className == "WildAnimal";

}

~WildAnimal()

{

printf("~WildAnimal()");

}

};

Определение класса-предка Base

class Base

{

public:

Base()

{

printf("Base()\n");

}

Base(Base\* obj)

{

printf("Base(Base\* obj)\n");

}

Base(const Base& obj)

{

printf("Base(Base& obj)\n");

}

~Base()

{

printf("~Base()\n");

}

};

Определение класса-потомка Desc

class Desc :public Base

{

public:

Desc()

{

printf("Desc()\n");

}

Desc(Desc\* obj)

{

printf("Desc(Desc\* obj)\n");

}

Desc(const Desc& obj)

{

printf("Desc(Desc& obj)\n");

}

~Desc()

{

printf("~Desc()\n");

}

};

Определение func1(), func2(), func3()

void func1(Base obj)

{

printf("void func1(Base obj)\n");

}

void func2(Base\* obj)

{

printf("void func2(Base\* obj)\n");

}

void func3(Base& obj)

{

printf("void func3(Base& obj)\n");

}

Определение классов для доп. тестирования Test1 и Test2

class Test1

{

public:

Test1()

{

printf("Test1()\n");

}

void method1()

{

method2();

}

void method2()

{

printf("method2() из Test1\n");

}

void method3()

{

method4();

}

virtual void method4()

{

printf("method4() из Test1\n");

}

virtual ~Test1()

{

printf("~Test1()\n");

}

};

class Test2 : public Test1

{

public:

int\* n;

Test2()

{

n = new int[10];

printf("Test2() - Память выделена\n");

}

void method2()

{

printf("method2() из Test2\n");

}

void method4()

{

printf("method4() из Test2\n");

}

~Test2()

{

delete n;

printf("~Test2() - Память освобождена\n");

}

};

Начало тестирования

Создание объектов базового класса и потомка. Помещение потомка в указатель на предка.

Animal\* animal1 = new Animal(5, 4, "Кот");

WildAnimal\* wildAnimal1 = new WildAnimal(8, 160, "Тигр");

Animal\* animal2 = new WildAnimal(10, 190, "Лев");

Вызов перекрываемых методов. В первом случае метод не перекрыт, вызывается метод базового класса. Во втором случае метод перекрыт, вызывается метод наследника.

animal1->sound();

wildAnimal1->sound();

Виртуальные методы. sound2 в предке виртуальный. При вызове метода наследником из под указателя на предка, вызывается метод предка, т.е. метод остаётся переопределённым.

animal1->sound2();

wildAnimal1->sound2();

animal2->sound2();

Проверка на принадлежность классу. Метод className перекрывается в наследнике.

if (animal2->className() == "WildAnimal")

printf("WildAnimal\n");

if (animal2->isA("WildAnimal"))

printf("WildAnimal\n");

Безопасное приведение типов. dynamic\_cast в случае успеха приведения animal2 к типу WildAnimal\* вернёт указатель на объект этого типа, в противном случае вернёт 0.

WildAnimal\* temp = dynamic\_cast<WildAnimal\*>(animal2);

if (temp)

temp->chase();

Безопасное приведение типов (вручную). Метод isA проверяет animal2 на соответствие типу WildAnimal через строковые константы.

if (animal2->isA("WildAnimal"))

((WildAnimal\*)animal2)->chase();

Передача объектов как параметров в функции. При передаче указателя в функцию func1, которая принимает объект в качестве параметра, создастся новый объект класса Base. При этом будет вызван конструктор Base(Base\* obj), где в качестве obj передастся base. Base(Base\* obj) является конструктором копирования, и если его реализовать неправильно, то func1 будет работать с объектом по свойствам отличным от base. func2 принимает указатель. В этом случае создастся копия указателя, а не копия объекта. В func3 будет передаваться прямая ссылка на объект, никаких копий создаваться не будет.

Base\* base = new Base();

Desc\* desc = new Desc();

printf("\n");

func1(base);

func2(base);

func3(\*base);

printf("\n");

func1(desc);

func2(desc);

func3(\*desc);

Виртуальный деструктор. Деструктор Test1 – виртуальный, поэтому при удалении test вызовется сначала деструктор Test2, а потом деструктор Test1. Если бы деструктор не был виртуальным, то деструктор Test2 не был бы вызван, а значит произойдёт утечка памяти.

Test1\* test = new Test2();

delete test;

Вызов переопределённого метода. method1 вызывает method2 в классе-предке. method2 переопределён в потомке. При вызове из потомка method1 будет вызван method2 предка.

Test2\* test\_ = new Test2();

test\_->method1();

test\_->method2();

Вызов переопределённого виртуального метода. method3 вызывает method4 в классе-предке. method4 виртуальный в предке и переопределён в потомке. При вызове из потомка method3 будет вызван method4 потомка.

test\_->method3();

test\_->method4();

1. **Выводы:**

* Если метод переопределить в потомке, и вызвать этот метод из-под указателя предка на потомка, вызовется метод предка.
* Если виртуальный метод переопределить в потомке, и вызвать этот метод из-под указателя предка на потомка, вызовется метод потомка.
* Для безопасного приведения типов можно использовать dynamic\_cast или написать собственные виртуальные методы для переопределения.
* Передавать объекты следует по ссылкам или указателям, чтобы избежать излишнего и возможно неправильного копирования объекта.
* Виртуальный деструктор необходим для правильного удаления объектов, хранящихся в указателе на предка. Если не сделать деструктор виртуальным в предке, то деструктор потомка не будет вызван, что приведёт к утечке памяти.
* При вызове переопределённого не виртуального метода другим методом в классе предке из-под указателя предка на потомка, будет вызван метод предка. Если метода виртуальный, то будет вызван метод потомка.

Приложение А. Ссылка на репозиторий и скриншот истории коммитов

<https://github.com/Ruslan376M/OOP_Lab5>

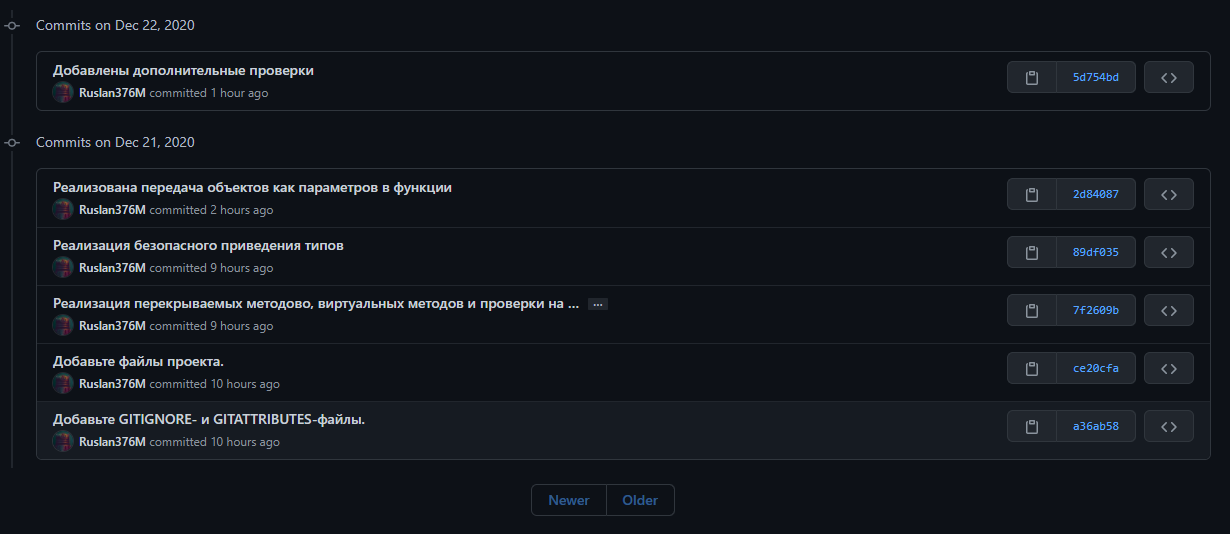


Рисунок 1 - Скриншот истории коммитов

Приложение B. Пример работы программы.

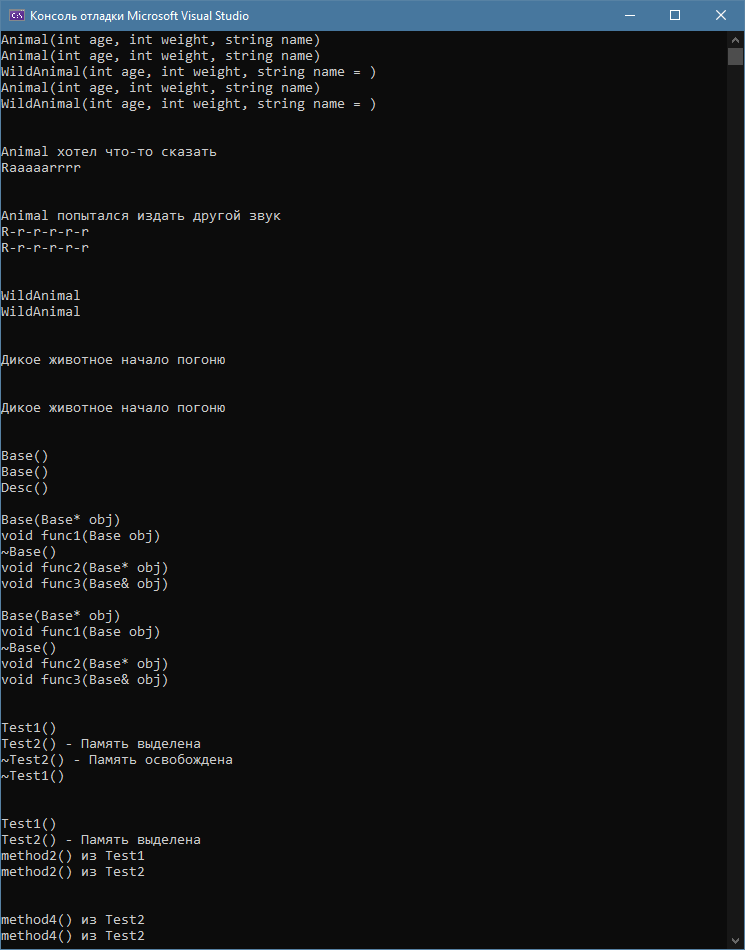


Рисунок 2 - Пример работы программы

Приложение C. Исходный код программы.

#include <iostream>

#include <locale.h>

#include <string>

using namespace std;

class Animal

{

protected:

int age;

int weight;

string name;

public:

Animal()

{

age = 0;

weight = 0;

name = "";

printf("Animal()\n");

}

Animal(int age, int weight, string name = "")

{

this->age = age;

this->weight = weight;

this->name = name;

printf("Animal(int age, int weight, string name)\n");

}

Animal(const Animal& animal)

{

this->age = animal.age;

this->weight = animal.weight;

this->name = animal.name;

printf("Animal(const Animal& animal)\n");

}

int getAge()

{

return age;

}

int getWeight()

{

return weight;

}

string getName()

{

return name;

}

void sound()

{

printf("Animal хотел что-то сказать\n");

}

virtual void sound2()

{

printf("Animal попытался издать другой звук\n");

}

virtual string className()

{

return "Animal";

}

virtual bool isA(string className)

{

return className == "Animal";

}

virtual ~Animal()

{

printf("~Animal()");

}

};

class WildAnimal :public Animal

{

public:

WildAnimal()

{

printf("WildAnimal()\n");

}

WildAnimal(int age, int weight, string name = "")

:Animal(age, weight, name)

{

printf("WildAnimal(int age, int weight, string name = "")\n");

}

void sound()

{

printf("Raaaaarrrr\n");

}

void sound2()

{

printf("R-r-r-r-r-r\n");

}

void chase()

{

printf("Дикое животное начало погоню\n");

}

string className()

{

return "WildAnimal";

}

bool isA(string className)

{

return className == "WildAnimal";

}

~WildAnimal()

{

printf("~WildAnimal()");

}

};

class Base

{

public:

Base()

{

printf("Base()\n");

}

Base(Base\* obj)

{

printf("Base(Base\* obj)\n");

}

Base(const Base& obj)

{

printf("Base(Base& obj)\n");

}

~Base()

{

printf("~Base()\n");

}

};

class Desc :public Base

{

public:

Desc()

{

printf("Desc()\n");

}

Desc(Desc\* obj)

{

printf("Desc(Desc\* obj)\n");

}

Desc(const Desc& obj)

{

printf("Desc(Desc& obj)\n");

}

~Desc()

{

printf("~Desc()\n");

}

};

void func1(Base obj)

{

printf("void func1(Base obj)\n");

}

void func2(Base\* obj)

{

printf("void func2(Base\* obj)\n");

}

void func3(Base& obj)

{

printf("void func3(Base& obj)\n");

}

class Test1

{

public:

Test1()

{

printf("Test1()\n");

}

void method1()

{

method2();

}

void method2()

{

printf("method2() из Test1\n");

}

void method3()

{

method4();

}

virtual void method4()

{

printf("method4() из Test1\n");

}

virtual ~Test1()

{

printf("~Test1()\n");

}

};

class Test2 : public Test1

{

public:

int\* n;

Test2()

{

n = new int[10];

printf("Test2() - Память выделена\n");

}

void method2()

{

printf("method2() из Test2\n");

}

void method4()

{

printf("method4() из Test2\n");

}

~Test2()

{

delete n;

printf("~Test2() - Память освобождена\n");

}

};

int main()

{

setlocale(LC\_ALL, "Russian");

Animal\* animal1 = new Animal(5, 4, "Кот");

WildAnimal\* wildAnimal1 = new WildAnimal(8, 160, "Тигр");

Animal\* animal2 = new WildAnimal(10, 190, "Лев");

printf("\n\n");

// Перекрываемые методы

animal1->sound();

wildAnimal1->sound();

printf("\n\n");

// Виртуальные методы

animal1->sound2();

wildAnimal1->sound2();

animal2->sound2();

printf("\n\n");

// Проверка на принадлежность классу

if (animal2->className() == "WildAnimal")

printf("WildAnimal\n");

if (animal2->isA("WildAnimal"))

printf("WildAnimal\n");

printf("\n\n");

// Безопасное приведение типов (dynamic\_cast)

WildAnimal\* temp = dynamic\_cast<WildAnimal\*>(animal2);

if (temp)

temp->chase();

printf("\n\n");

// Безопасное приведение типов (вручную)

if (animal2->isA("WildAnimal"))

((WildAnimal\*)animal2)->chase();

printf("\n\n");

// Передача объектов как параметров в функции

Base\* base = new Base();

Desc\* desc = new Desc();

printf("\n");

func1(base);

func2(base);

func3(\*base);

printf("\n");

func1(desc);

func2(desc);

func3(\*desc);

printf("\n\n");

// Виртуальный деструктор

Test1\* test = new Test2();

delete test;

printf("\n\n");

// Вызов переопределённого метода

Test2\* test\_ = new Test2();

test\_->method1();

test\_->method2();

printf("\n\n");

// Вызов переопределённого виртуального метода

test\_->method3();

test\_->method4();

printf("\n\n");

}