Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра экономической информатики

ОТЧЁТ

по лабораторной работе

**ПРИНЦИП ПОЛИМОРФИЗМА.**

**ВИРТУАЛЬНЫЕ ФУНКЦИИ. АБСТРАКТНЫЕ**

**КЛАССЫ**

|  |  |
| --- | --- |
| Выполнил: | Студент группы 224403  Одинец И.И. |
| Проверил: | Ассистент кафедры ЭИ  Сизоненко Н.А. |

Минск 2023

**Цель работы:** изучить принципы и получить практические навыки

при использовании множественного наследования; рассмотреть случаи, когда

необходимо использовать множественное и виртуальное наследование.

**Теоретические сведения**

В языке С++ имеется возможность наследовать производный класс

одновременно от двух и более базовых классов. Общая форма такого

наследования имеет следующий вид:

class BaseClass1 {};

class BaseClass2 {};

class DerivedClass : public BaseClass1, public BaseClass2 {

//содержимое класса DerivedClass

};

При множественном наследовании имена базовых классов разделяются

запятыми, и перед каждым именем базового класса указывается свой

спецификатор наследования (public, private, protected).

Иерархическая структура, в которой производный класс

одновременно наследуется от нескольких базовых классов, называется

множественным наследованием. В этом случае производный класс, имея

собственные компоненты, имеет доступ к protected- и public-компонентам

базовых классов. Конструкторы базовых классов при создании объекта

производного класса вызываются в том порядке, в котором они указаны в

списке при объявлении производного класса

Виртуальное наследование базового класса гарантирует, что в любом

производном классе будет присутствовать только одна его копия.

Применение оператора "::" (разрешения контекста, разрешения области

видимости) позволяет программе "ручным способом" выбрать версию класса.

Но если необходима только одна копия базового класса, то это достигается с

помощью виртуального наследования базовых классов

Если два (или больше) класса выведены из общего базового класса, то

можно предотвратить включение нескольких копий базового класса в

объекте дочерних классов, выведенного из этих классов, что реализуется

путем объявления наследовании базового класса виртуальным. Для этого

достаточно предварить имя наследуемого базового класса ключевым словом

virtual.

Разница между обычным базовым и виртуальным классами становится

очевидной только тогда, когда этот базовый класс наследуется более одного

раза. Если наследование базового класса объявляется виртуальным, то только

один его экземпляр будет включен в объект наследующего класса. В

противном случае в этом объекте будет присутствовать несколько его копий.

ЧТЕНИЕ ГРАФОВ НАСЛЕДОВАНИЯ

Связи между классами могут изображаются графически и с помощью

языка моделирования UML, что облегчает их понимание и повышает

презентативность. Например, диаграмма классов демонстрирует общую

структуру иерархии классов системы, их коопераций, атрибутов (полей),

методов, интерфейсов и взаимосвязей (отношений) между ними. Диаграмма

классов широко используется не только для документирования и

визуализации, но также для конструирования посредством прямого или

обратного проектирования.

Стрелки на этом рисунке направлены вверх к базовому классу.

Согласно стилевой графике C++ стрелка наследования (обобщения,

генерализации, generalization) должна указывать на базовый класс.

НАСЛЕДОВАНИЕ ИНТЕРФЕЙСА

Одно из применений множественного наследования, не вызывающее

никаких возражений, связано с наследованием интерфейса. В языке C++ все

наследование является наследованием реализации, поскольку все аспекты

базового класса, интерфейс и реализация становятся частью производного класса. Унаследовать только часть класса (скажем, интерфейс) невозможно.

Защищенное и закрытое наследование позволяет ограничить доступ к

элементам, унаследованным от базового класса, со стороны клиентов объекта

производного класса, но на самом производном классе это не сказывается; он

все равно содержит все данные базового класса и может обращаться ко всем

незакрытым элементам базового класса.

С другой стороны, наследование интерфейса только добавляет

объявления функций в интерфейс производного класса. Такая возможность

не поддерживается в языке C++ напрямую. Стандартная методика имитации

наследования интерфейса основана на наследовании от интерфейсного

класса, то есть класса, содержащего только объявления методов (но не их

определения). Все объявления интерфейсного класса, кроме деструктора,

должны быть чисто виртуальными функциями.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Для чего используется множественное наследование? Чем оно

отличается от простого наследования?

Множественное наследование позволяет классу перенимать функциональность у множества других классов. Также одно из применений множественного наследования, не вызывающее

никаких возражений, связано с наследованием интерфейса.

2. Каков порядок вызова конструкторов и деструкторов при

множественном наследовании?

Сначала срабатывают конструкторы базовых классов в порядке их задания;

После конструкторы членов, являющихся объектами класса;

Затем конструктор порожденного класса.

3. Каков порядок вызова конструкторов и деструкторов при

множественном виртуальном наследовании?

Конструкторы виртуальных базовых классов выполняются до конструкторов не виртуальных базовых классов, независимо от того, как эти классы заданы в списке порождения;

Если класс имеет несколько виртуальных базовых классов, то конструкторы этих классов вызываются в порядке объявления виртуальных базовых классов в списке порождения;

Деструкторы виртуальных базовых классов выполняются после деструкторов не виртуальных базовых классов.

4. Какие проблемы возможны при множественном наследовании и как

они разрешаются при использовании виртуального наследования?

Проблема двойного включения при ромбовидном наследовании решается с помощью виртуального наследования. Также Применение оператора "::" (разрешения контекста, разрешения области видимости) позволяет программе "ручным способом" выбрать версию класса.

5. Какие языки не поддерживают множественного наследования

классов? Почему?

Такие языки как PHP, C#, Java не поддерживают множественного наследования классов, но поддерживают множественное наследование интерфейсов.

**Индивидуальное задание**

Вариант 21:

.

Построить иерархию классов согласно схеме наследования,

приведенной на рисунке ниже по предметной области «Интернет магазин».

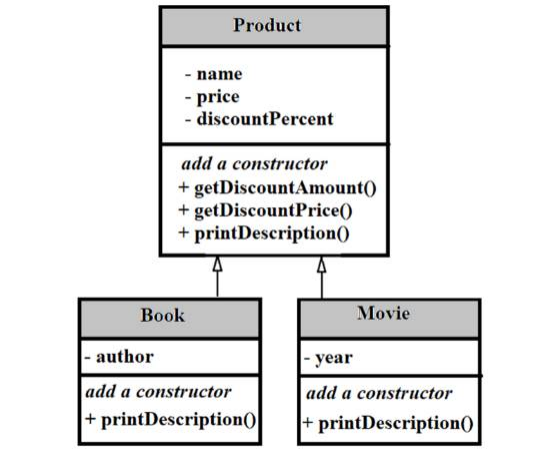
Каждый класс должен содержать необходимые конструкторы и методы

работы с полями классов. Функция main() должна иллюстрировать работу с

массивами объектов всех созданных классов. При необходимости

самостоятельно добавить классы для реализации множественного

наследования.



**Листинг**

class Product

{

protected:

std::string name;

float price;

int discount\_percent;

public:

Product(std::string s, float f, int i):name(s), price{f}, discount\_percent(i){}

Product() :name(""), price{0}, discount\_percent(0) {}

virtual void print\_info() {

std::cout << "Product name: " << name << std::endl;

std::cout << "Product price: " << price << std::endl;

std::cout << "Product discount\_percent: " << discount\_percent << std::endl;

for (int i = 0; i < 15; i++)

std::cout << '-';

std::cout << std::endl;

};

std::string get\_name() { return name; }

float get\_price() { return price; }

int get\_discount\_percent() { return discount\_percent; }

void set\_name(std::string s) { name=s; }

void set\_price(float f) { price=f; }

void set\_discount\_percent(int p) { discount\_percent=p; }

};

class Book :

public Product

{

protected:

std::string author;

public:

Book(std::string s):author(s){}

Book(std::string author,std::string s, float f, int i) :author(author) {

this->set\_name(s);

this->set\_price(f);

this->set\_discount\_percent(i);

}

Book():author(""){}

virtual void print\_info() override {

std::cout << "Book:" << author << std::endl;

Product::print\_info();

};

std::string get\_author() { return author; }

void set\_author(std::string s) { author =s; }

};

class Movie :

public virtual Product

{

protected:

int year;

public:

Movie(int i):year(i){}

Movie(int year, std::string s, float f, int i) :year(year) {

this->set\_name(s);

this->set\_price(f);

this->set\_discount\_percent(i);

}

Movie():year(0){}

virtual void print\_info() override {

std::cout << "Movie:" << std::endl;

std::cout << "Movie year: " << year << std::endl;

Product::print\_info();

};

int get\_year() { return year; }

void set\_year(int p) { year = p; }

};

class Game :

public virtual Product

{

protected:

std::string genre;

public:

Game(std::string genre):genre(genre){}

Game(std::string genre, std::string s, float f, int i) :genre(genre) {

this->set\_name(s);

this->set\_price(f);

this->set\_discount\_percent(i);

}

Game():genre(""){}

virtual void print\_info() override

{

std::cout << "Game:" << std::endl;

std::cout << "Game genre: " << genre << std::endl;

Product::print\_info();

};

std::string get\_genre() { return genre; }

void set\_genre(std::string s) { genre = s; }

};

class Interactive\_movie :

public Movie, public Game

{

protected:

int number\_of\_story\_ending;

public:

Interactive\_movie(int i):number\_of\_story\_ending(i){}

Interactive\_movie(int N\_o\_s\_ending, std::string s, float f, int i, std::string genre, int year) :number\_of\_story\_ending(N\_o\_s\_ending) {

this->set\_genre(genre);

this->set\_year(year);

this->set\_name(s);

this->set\_price(f);

this->set\_discount\_percent(i);

}

Interactive\_movie() :number\_of\_story\_ending(0) {}

virtual void print\_info() override

{

std::cout << "Interactive film:" << std::endl;

std::cout << "Number of story ending: " << number\_of\_story\_ending << std::endl;

std::cout << "Movie year: " << year << std::endl;

std::cout << "Game genre: " << genre << std::endl;

Product::print\_info();

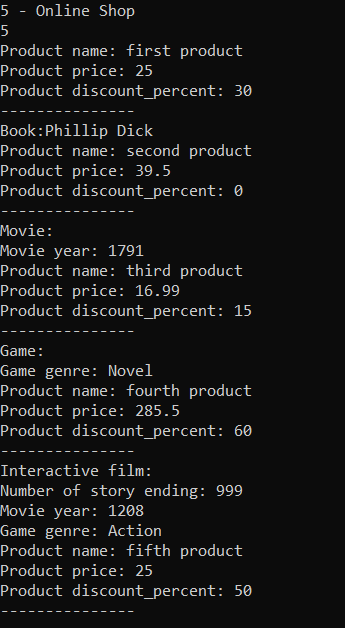
};

int get\_number\_of\_story\_ending() { return number\_of\_story\_ending; }

void set\_number\_of\_story\_ending(int p) { number\_of\_story\_ending = p; }

};

На рисунке 1 представлено создание обьектов классов



**Рисунок 1 – Создание объектов класса**

**Вывод:** в ходе работы были изучены принципы и получены практические навыки при использовании множественного наследования; рассмотрены случаи, когда необходимо использовать множественное и виртуальное наследование.