Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

ОТЧЕТ

к лабораторной работе №3

**«НАСЛЕДОВАНИЕ. ПРОСТОЕ НАСЛЕДОВАНИЕВ ЯЗЫКЕ С++»**

|  |  |
| --- | --- |
|  | |
|  |
|  |
|  |
|  |  | | |
|  | | |  | |  |
|  | | |  | |  |
| Студент гр. 324402 | | |  | | Цевелюк А. И. |
| Проверил | | |  | | Булыга Д. И. |

Минск 2024

**1 Цель работы**

Цель работы:изучить понятие базовых и производных классов, основные правила построения иерархии классов, особенности вызова конструкторов деструкторов при наследовании, механизмы разграничения доступа.

**2 Теоретические сведения по лабораторной работе**

Наследование — это механизм, который позволяет создавать новые классы на основе уже существующих, перенимая их свойства и методы.

Базовый класс — это класс, от которого происходит наследование. Производный класс — это класс, который наследует от базового класса.

Иерархия классов — это структура, которая отражает отношения наследования между классами. В иерархии классов базовый класс находится выше производного класса.

При наследовании конструкторы и деструкторы базового класса вызываются автоматически при создании и уничтожении объектов производного класса. При этом сначала вызывается конструктор базового класса, а затем конструктор производного класса. При уничтожении объекта сначала вызывается деструктор производного класса, а затем деструктор базового класса.

Разграничение доступа — это способ определения видимости и доступности членов класса для других классов. В C++ существуют три модификатора доступа: public, protected и private. Публичные члены класса доступны для всех классов, защищенные члены класса доступны для класса и его производных классов, а приватные члены класса доступны только для самого класса.

**3 Формулировка индивидуального задания**

Написать программу для автоматизации работы библиотеки. Создать классы «Книга», «Отдел», «Библиотека» с необходимым набором полей и методов. В классах реализовать следующие функции: добавление, удаление книг из отделов, выдача книг. Вывести данные о книгах (на экран и в файл), возврат по которым просрочен. Классы должны содержать методы получения и изменения значений всех полей. Все поля классов должны быть объявлены с атрибутами private или protected.

**4 Код решения индивидуального задания**

#include <fstream>

#include <iomanip>

#include <iostream>

#include <sstream>

#include <string>

#include <vector>

void convert\_string\_to\_time(const std::string& string, int& day, int& month, int& year)

{

tm tm;

std::istringstream stream(string);

stream >> std::get\_time(&tm, "%d.%m.%Y");

day = tm.tm\_mday;

month = tm.tm\_mon;

year = tm.tm\_year;

}

void clear\_file()

{

std::fstream file;

file.open("lab3\_logs.txt", std::ofstream::out | std::ofstream::trunc);

file.close();

}

bool is\_older(const std::string& first, const std::string& second)

{

int day1, day2, month1, month2, year1, year2;

convert\_string\_to\_time(first, day1, month1, year1);

convert\_string\_to\_time(second, day2, month2, year2);

if (year1 < year2)

{

return true;

}

if (year1 == year2)

{

if (month1 < month2)

{

return true;

}

if (month1 == month2)

{

if (day1 < day2)

{

return true;

}

}

}

return false;

}

class book

{

protected:

std::string name\_;

std::string author\_;

std::string date\_;

public:

book();

book(const std::string& name, const std::string& author, const std::string& date);

void set\_name();

void set\_author();

void set\_date();

void print\_book() const;

void print\_given\_book() const;

void set\_book();

std::string get\_name() const;

std::string get\_author() const;

std::string get\_date() const;

};

std::string book::get\_name() const

{

return name\_;

}

std::string book::get\_author() const

{

return author\_;

}

std::string book::get\_date() const

{

return date\_;

}

void book::set\_name()

{

std::string name;

std::cout << "Enter name of book: ";

std::cin >> name;

name\_ = name;

}

void book::set\_author()

{

std::string author;

std::cout << "Enter author of book: ";

std::cin >> author;

author\_ = author;

}

void book::set\_date()

{

std::string date;

std::cout << "Enter date to return of book: ";

std::cin >> date;

date\_ = date;

}

void book::print\_book() const

{

std::cout << "Name: " << name\_ << ", Author: " << author\_ << '\n';

}

void book::print\_given\_book() const

{

std::cout << "Name: " << name\_ << ", Author: " << author\_ << ", Date to return: " << date\_ << '\n';

}

book::book()

{

name\_ = "Unnamed";

author\_ = "Unknown";

date\_ = "01.01.2020";

}

book::book(const std::string& name, const std::string& author, const std::string& date)

{

name\_ = name;

author\_ = author;

date\_ = date;

}

void book::set\_book()

{

int choice = 0;

std::cout << "Enter your choice\n1 - name only\n2 - author only\n3 - name and author\nany else - default: ";

std::cin >> choice;

switch (choice)

{

case 1:

set\_name();

break;

case 2:

set\_author();

break;

case 3:

set\_name();

set\_author();

break;

default:

break;

}

}

class section : protected book

{

protected:

std::vector<book> books\_section\_;

std::string section\_name\_;

int size\_;

public:

section();

section(const std::string& name, int size);

explicit section(const std::string& name);

explicit section(int size);

void set\_section\_name();

void set\_section\_size();

void print\_section() const;

void print\_given\_section() const;

void set\_section();

void print\_outdated\_books() const;

std::string get\_section\_name() const;

int& get\_section\_size();

std::vector<book>& get\_section\_vector();

};

std::string section::get\_section\_name() const

{

return section\_name\_;

}

int& section::get\_section\_size()

{

return size\_;

}

std::vector<book>& section::get\_section\_vector()

{

return books\_section\_;

}

void section::print\_section() const

{

std::cout << "Section " << section\_name\_ << ":\n";

for (const book& book : books\_section\_)

book.print\_book();

}

void section::print\_given\_section() const

{

std::cout << "Section " << section\_name\_ << ":\n";

for (const book& book : books\_section\_)

book.print\_given\_book();

}

section::section()

{

section\_name\_ = "Unnamed section";

size\_ = 5;

}

section::section(const int size)

{

section\_name\_ = "Unnamed section";

size\_ = size;

}

section::section(const std::string& name)

{

section\_name\_ = name;

size\_ = 5;

}

section::section(const std::string& name, const int size)

{

section\_name\_ = name;

size\_ = size;

}

void section::set\_section\_name()

{

std::string name;

std::cout << "Enter name of section: ";

std::cin >> name;

section\_name\_ = name;

}

void section::set\_section\_size()

{

int size;

std::cout << "Enter size of section: ";

std::cin >> size;

size\_ = size;

}

void section::set\_section()

{

int choice = 0;

std::cout << "Enter your choice\n1 - size only\n2 - name only\n3 - size and name\nany else - default: ";

std::cin >> choice;

switch (choice)

{

case 1:

set\_section\_size();

break;

case 2:

set\_section\_name();

break;

case 3:

set\_section\_name();

set\_section\_size();

break;

default:

break;

}

for (int i = 0; i < size\_; i++)

{

book book\_to\_push;

book\_to\_push.set\_book();

books\_section\_.push\_back(book\_to\_push);

}

}

void section::print\_outdated\_books() const

{

std::string now\_date;

clear\_file();

std::fstream file;

file.open("lab3\_logs.txt", std::ios::app);

std::cout << "Enter today's date: ";

std::cin >> now\_date;

bool found = false;

std::cout << "Outdated books:\n";

for (const book& book: books\_section\_)

{

if (is\_older(book.get\_date(), now\_date))

{

found = true;

book.print\_given\_book();

file << "Name: " << book.get\_name() << ", Author: " << book.get\_author() << ", Date to return: " << book.get\_date() << '\n';

}

}

if (!found)

std::cout << "There's no outdated given books!";

file.close();

}

class library : protected section

{

private:

std::vector<section> books\_library\_;

int library\_size\_;

public:

library();

explicit library(int size);

void set\_library();

void print\_library() const;

void add\_book();

void delete\_book();

void give\_book(section& given\_books);

};

library::library()

{

library\_size\_ = 3;

}

library::library(const int size)

{

library\_size\_ = size;

}

void library::set\_library()

{

int size = 0;

std::cout << "Enter size of your library (0 = default): ";

std::cin >> size;

if (size != 0)

library\_size\_ = size;

else

library\_size\_ = 3;

for (int i = 0; i < library\_size\_; i++)

{

section to\_push\_section;

to\_push\_section.set\_section();

books\_library\_.push\_back(to\_push\_section);

}

}

void library::print\_library() const

{

for (const section& section : books\_library\_)

section.print\_section();

}

void library::add\_book()

{

std::string section\_name;

bool found = false;

std::cout << "Enter section name in which to add: ";

std::cin >> section\_name;

for (section& section : books\_library\_)

{

if (section.get\_section\_name() == section\_name)

{

found = true;

book to\_push\_book;

to\_push\_book.set\_book();

section.get\_section\_vector().push\_back(to\_push\_book);

section.get\_section\_size()++;

break;

}

}

if (!found)

std::cout << "There's no section with such name!\n";

}

void library::delete\_book()

{

std::string section\_name, book\_name;

bool found = false;

std::cout << "Enter section name from which to delete: ";

std::cin >> section\_name;

std::cout << "Enter book to delete name: ";

std::cin >> book\_name;

for (section& section : books\_library\_)

{

if (section.get\_section\_name() == section\_name)

{

found = true;

for (int i = 0; i < section.get\_section\_size();)

{

if (section.get\_section\_vector()[i].get\_name() == book\_name)

{

section.get\_section\_vector().erase(section.get\_section\_vector().begin() + i);

section.get\_section\_size()--;

}

else

i++;

}

break;

}

}

if (!found)

std::cout << "There's no book with such name, or there's no section with such name!\n";

}

void library::give\_book(section& given\_books)

{

std::string section\_name, book\_name;

bool found = false;

std::cout << "Enter section name from which to give: ";

std::cin >> section\_name;

std::cout << "Enter book to give name: ";

std::cin >> book\_name;

book book\_to\_give;

for (section& section : books\_library\_)

{

if (section.get\_section\_name() == section\_name)

{

found = true;

for (int i = 0; i < section.get\_section\_size();)

{

if (section.get\_section\_vector()[i].get\_name() == book\_name)

{

book\_to\_give = section.get\_section\_vector()[i];

section.get\_section\_vector().erase(section.get\_section\_vector().begin() + i);

section.get\_section\_size()--;

break;

}

i++;

}

break;

}

}

if (!found)

{

std::cout << "There's no book with such name, or there's no section with such name!\n";

return;

}

book\_to\_give.set\_date();

given\_books.get\_section\_vector().push\_back(book\_to\_give);

given\_books.get\_section\_size()++;

}

void clear()

{

system("pause");

system("cls");

}

int main()

{

library library;

section given\_books("Given books", 0);

library.set\_library();

clear();

int choice = 0;

do

{

library.print\_library();

std::cout << "Enter your choice:\n1 - add book to library\n2 - delete book from library\n3 - give book from library\n4 - print info about given books\n5 - print info about outdated given books\n-1 - exit: ";

std::cin >> choice;

switch (choice)

{

case -1:

break;

case 1:

library.add\_book();

clear();

break;

case 2:

library.delete\_book();

clear();

break;

case 3:

library.give\_book(given\_books);

clear();

break;

case 4:

given\_books.print\_given\_section();

clear();

break;

case 5:

given\_books.print\_outdated\_books();

clear();

break;

default:

std::cout << "Invalid input!";

clear();

}

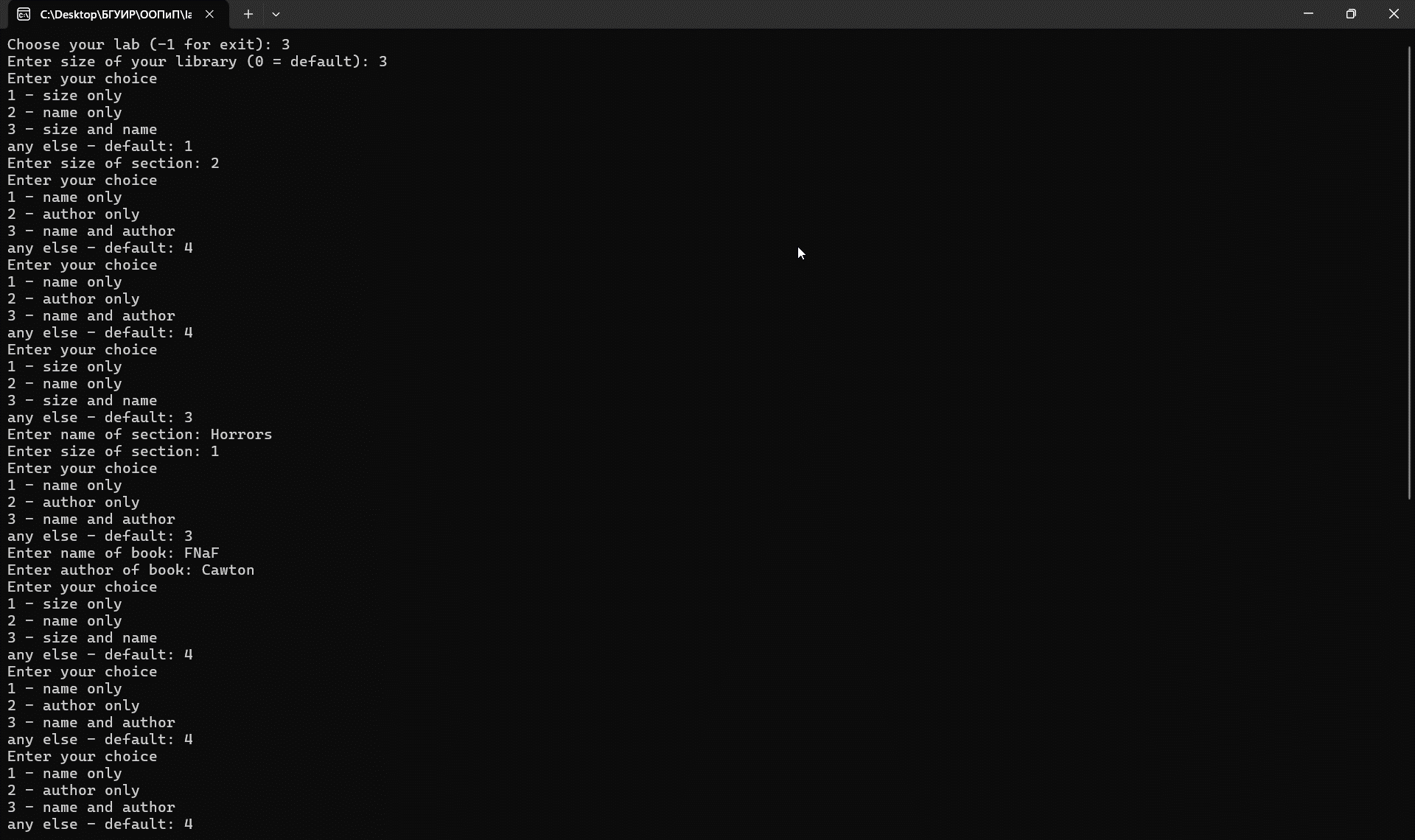
} while (choice != -1);

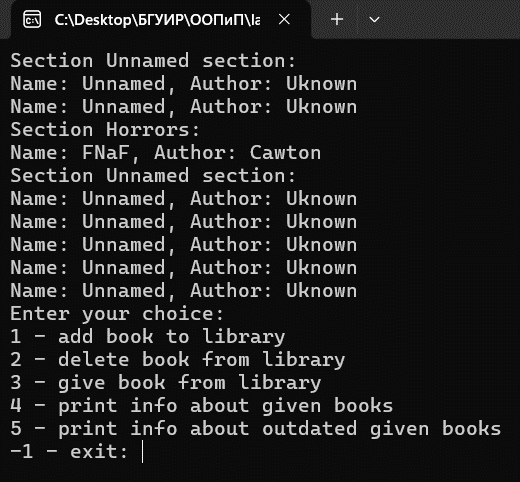
std::cout << "Have a nice day!\n";

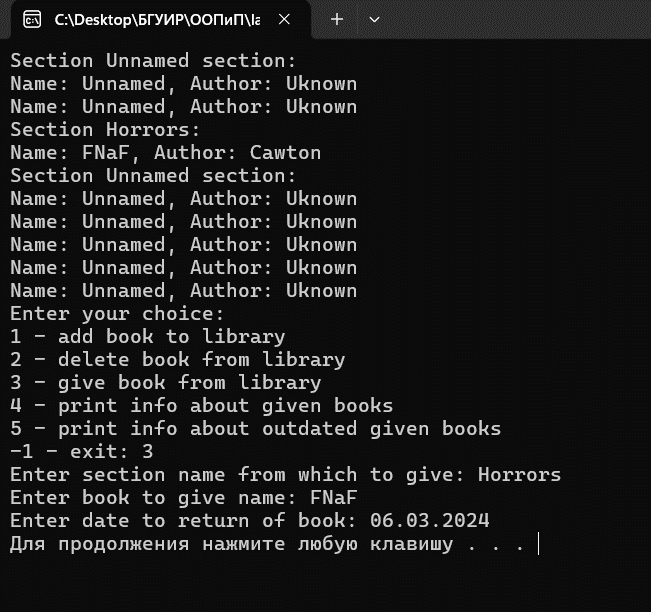
return 0;

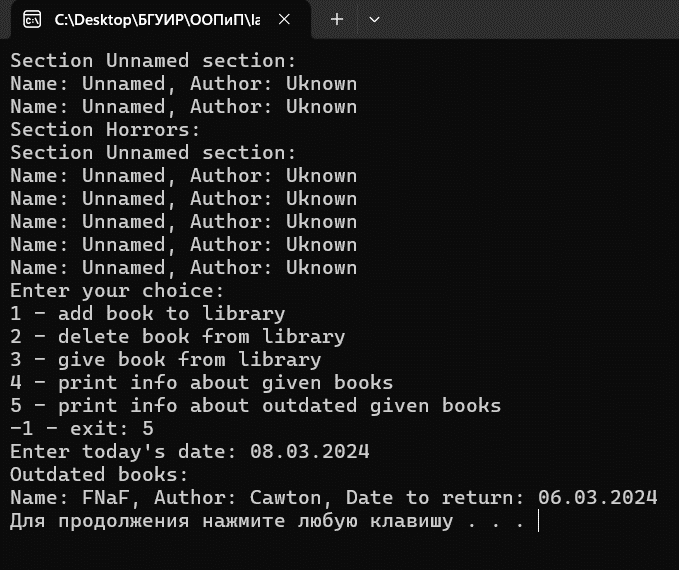
}

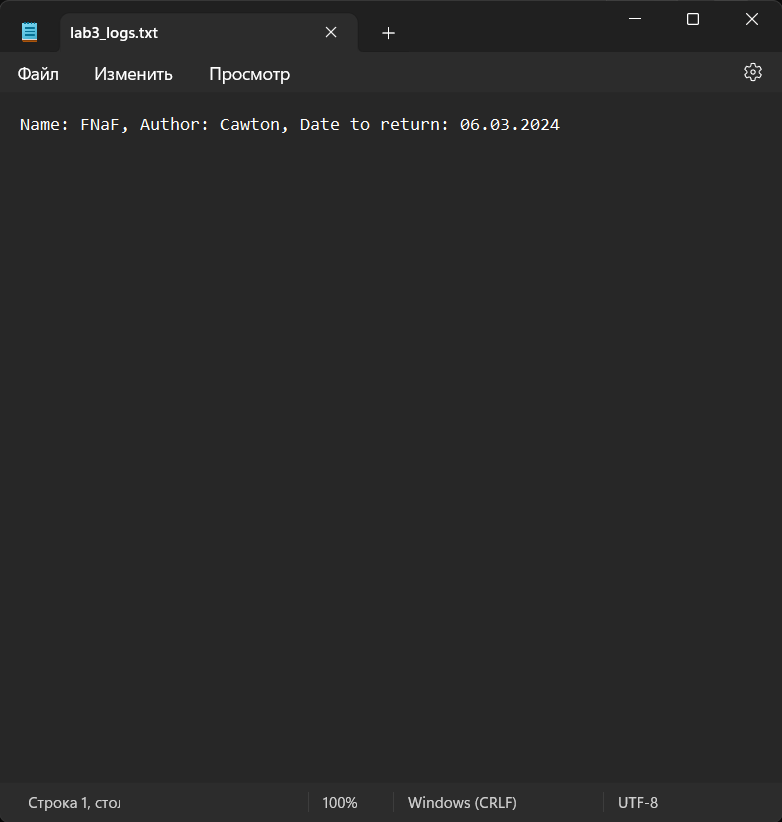
**5 Скриншоты выполнения индивидуального задания**











**6 Ответы на контрольные вопросы**

1. Какие модификаторы доступа есть в языке С++ к содержимому класса (к полям и методам)? Опишите каждый из них (наименование, предназначение).

Модификаторы доступа в языке C++ к содержимому класса (к полям и методам) — это ключевые слова, которые определяют, какие члены класса видны и доступны для других классов или функций. Существуют три модификатора доступа: public, protected и private.

public означает, что члены класса, объявленные с этим модификатором, доступны для всех классов и функций, в том числе для производных классов и внешних функций.

protected означает, что члены класса, объявленные с этим модификатором, доступны только для класса самого и его производных классов, но не для внешних функций1.

private означает, что члены класса, объявленные с этим модификатором, доступны только для класса самого, но не для его производных классов и внешних функций.

1. Какова цель использования наследования при проектировании и разработке объектно-ориентированных программ?

Цель использования наследования при проектировании и разработке объектно-ориентированных программ — это повышение степени абстракции, повторное использование кода, упрощение расширения и модификации программ, а также реализация полиморфизма. Наследование позволяет создавать новые классы на основе уже существующих, перенимая их свойства и методы, а также добавляя новые. Таким образом, наследование способствует иерархической организации классов, уменьшению дублирования кода, увеличению уровня обобщения и адаптации к изменяющимся требованиям.

1. Какой класс называется базовым и какой производным при наследовании? Перечислите другие возможные названия (имена) базового и производного классов.

Класс, от которого происходит наследование, называется базовым классом. Класс, который наследует от базового класса, называется производным классом. Другие возможные названия (имена) базового и производного классов — это: родительский и дочерний класс, суперкласс и подкласс, общий и специализированный класс.

1. В чем различие между public, private и protected наследованием классов?

Различие между public, private и protected наследованием классов заключается в том, как модификаторы доступа базового класса влияют на видимость и доступность членов производного класса. При public наследовании модификаторы доступа базового класса сохраняются в производном классе, то есть public члены базового класса становятся public членами производного класса, а protected члены базового класса становятся protected членами производного класса. При private наследовании все члены базового класса становятся private членами производного класса, независимо от их модификаторов доступа. При protected наследовании все public и protected члены базового класса становятся protected членами производного класса.

1. Какой порядок вызова и возврата управления из конструкторов при создании объектов дочернего класса?

Порядок вызова и возврата управления из конструкторов при создании объектов дочернего класса такой: сначала вызывается конструктор базового класса, затем конструктор производного класса, затем конструкторы всех членов-объектов производного класса. Порядок возврата управления из конструкторов обратный порядку вызова.

1. Какой порядок вызова и возврата управления из деструкторов при разрушении объектов дочернего класса?

Порядок вызова и возврата управления из деструкторов при разрушении объектов дочернего класса такой: сначала вызывается деструктор производного класса, затем деструкторы всех членов-объектов производного класса, затем деструктор базового класса. Порядок возврата управления из деструкторов обратный порядку вызова.

1. Опишите суть отношений агрегации и композиции между классами?

Суть отношений агрегации и композиции между классами — это способы представления отношений “часть-целое” между объектами. Агрегация — это отношение, при котором объекты, являющиеся частями другого объекта, могут существовать независимо от него и иметь свои собственные жизненные циклы. Композиция — это отношение, при котором объекты, являющиеся частями другого объекта, не могут существовать без него и зависят от его жизненного цикла.

1. Приведите пример наследования по произвольной предметной области с иерархией, содержащей 3 и более классов.

class Animal {

protected:

string name;

int age;

public:

Animal(string n, int a) {

name = n;

age = a;

}

string getName() {

return name;

}

int getAge() {

return age;

}

virtual void makeSound() {

cout << "Животное издает звук" << endl;

}

};

class Bird : public Animal {

protected:

int wingspan;

public:

Bird(string n, int a, int w) : Animal(n, a) {

wingspan = w;

}

int getWingspan() {

return wingspan;

}

void makeSound() override {

cout << "Птица издает звук" << endl;

}

};

class Dog : public Animal {

protected:

string breed;

public:

Dog(string n, int a, string b) : Animal(n, a) {

breed = b;

}

string getBreed() {

return breed;

}

void makeSound() override {

cout << "Собака издает звук" << endl;

}

};

class Parrot : public Bird {

private:

string color;

public:

Parrot(string n, int a, int w, string c) : Bird(n, a, w) {

color = c;

}

string getColor() {

return color;

}

void makeSound() override {

cout << "Попугай издает звук" << endl;

}

};

1. Могут ли быть унаследованы дочерним классом private поля и методы базового класса?

Private поля и методы базового класса не могут быть унаследованы дочерним классом в прямом смысле, то есть они не становятся членами дочернего класса и не доступны для него. Однако, они все еще существуют в объектах дочернего класса и могут быть использованы через public или protected методы базового класса.

1. Наследуются ли конструкторы и деструкторы?

Конструкторы и деструкторы не наследуются в том смысле, что они не становятся членами производного класса и не могут быть вызваны с его именем. Однако, при создании или уничтожении объекта производного класса, конструкторы и деструкторы базового класса вызываются автоматически, чтобы инициализировать или освободить ресурсы, связанные с базовым классом.

**7 Выводы по лабораторной работе**

Вывод: я изучил понятие базовых и производных классов, основные правила построения иерархии классов, особенности вызова конструкторов деструкторов при наследовании, механизмы разграничения доступа.