 МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ   
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ БІОМЕДИЧНОЇ ІНЖЕНЕРІЇ

КАФЕДРА БІОМЕДИЧНОЇ КІБЕРНЕТИКИ

**Компьютерный практикум №3**

з дисципліни «Об'єктно-орієнтоване програмування»

# на тему: «Успадкування. Поліморфізм»

Варіант №7

**Виконав:**

студент гр. БС-03

Затуловский Г. А.

**Перевірив:**

ас. каф. БМК Бабенко В.О.

Зараховано від \_\_\_.\_\_\_.\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

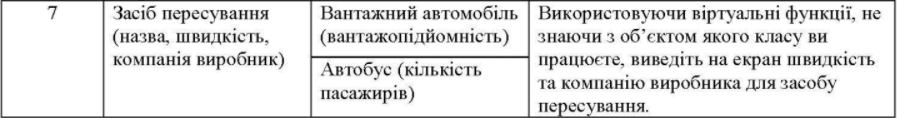
                      (підпис викладача)

Київ-2022

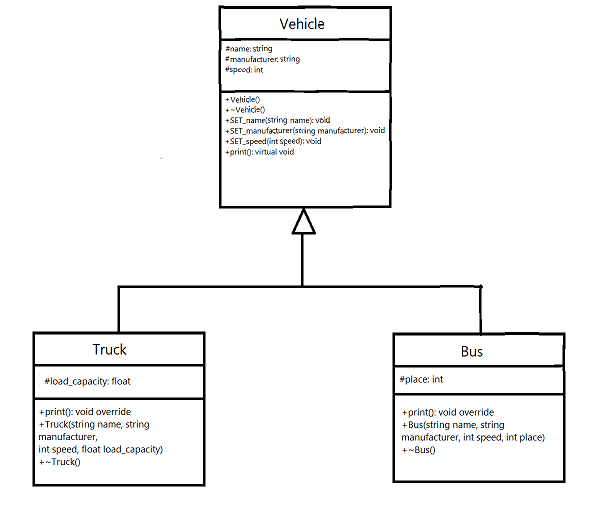
**Завдання:**

1. Вивчити теоретичні основи визначення класів, що використовують спадкування властивостей, та побудови ієрархії класів в мові С++.

2. Розробити діаграму класів в нотації UML та програмний застосунок з використанням успадкування, що реалізує ієрархію класів відповідно до свого варіанту (визначення класiн та їх реалізації мають бути розташовані у файлах . h та \*.cpp відповідно, для ініціалізації даних класів використовувати введення з клавіатури):



**UML діаграма:**

****

**Код програми:**

OOP\_N3.cpp:

#include <iostream>

#include "Vehicle.h"

#include "Bus.h"

#include "Truck.h"

#include <string>

using namespace std;

void CREATE\_VEHICLE()

{

string name = "";

string manufacturer = "";

int speed = 0;

int choice = 0;

cout << "Choice type of vehicle: \n"

<< "Bus - 1\n"

<< "Truck - 2" << endl << endl;

while (choice != 1 && choice != 2)

{

cin >> choice;

}

Vehicle veh;

cout << "Name: ";

cin >> name;

veh.SET\_name(name);

cout << "Manufacturer: ";

cin >> manufacturer;

veh.SET\_manufacturer(manufacturer);

cout << "Max speed: ";

cin >> speed;

veh.SET\_speed(speed);

veh.print();

if (choice == 1)

{

int place = 0;

cout << "Place: ";

cin >> place;

Bus veh(name, manufacturer, speed, place);

veh.print();

}

else

{

float load\_capacity = 0;

cout << "Load capacity (kg): ";

cin >> load\_capacity;

Truck veh(name, manufacturer, speed, load\_capacity);

veh.print();

}

cout << "Restart the program?\nYes - 1\nNo - 2\n";

cin >> choice;

if (choice == 1)

{

cout << endl << endl;

CREATE\_VEHICLE();

}

else

{

cout << "Closing the program" << endl;

}

}

int main()

{

CREATE\_VEHICLE();

return 0;

}

Vehicle.h:

#pragma once

#include <iostream>

#include <string>

using namespace std;

class Vehicle

{

protected:

string name;

string manufacturer;

int speed;

public:

Vehicle();

~Vehicle();

void SET\_name(string name);

void SET\_manufacturer(string manufacturer);

void SET\_speed(int speed);

virtual void print();

};

Truck.h:

#pragma once

#include <iostream>

#include <string>

#include "Vehicle.h"

using namespace std;

class Truck : public Vehicle

{

protected:

float load\_capacity;

public:

void print() override;

Truck(string name, string manufacturer, int speed, float load\_capacity);

~Truck();

};

Bus.h:

#pragma once

#include <iostream>

#include <string>

#include "Vehicle.h"

using namespace std;

class Bus : public Vehicle

{

protected:

int place;

public:

void print() override;

Bus(string name, string manufacturer, int speed, int place);

~Bus();

};

Vehicle.cpp:

#include "Vehicle.h"

#include <iostream>

#include <string>

using namespace std;

Vehicle::Vehicle()

{

name ="";

manufacturer ="";

speed = 0;

}

Vehicle::~Vehicle()

{

}

void Vehicle::SET\_name(string name)

{

this->name = name;

}

void Vehicle::SET\_manufacturer(string manufacturer)

{

this->manufacturer = manufacturer;

}

void Vehicle::SET\_speed(int speed)

{

this->speed = speed;

}

void Vehicle::print()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "VEHICLE INFO" <<

"\nName: "<< name <<

"\nManufacturer: " << manufacturer <<

"\nMax speed: "<<speed << endl << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

}

Truck.cpp:

#include "Truck.h"

void Truck::print()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "TRUCK INFO" <<

"\nName: " << name <<

"\nManufacturer: " << manufacturer <<

"\nMax speed: " << speed <<

"\nLoad capacity: " << load\_capacity << " kg "<<endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl << endl;;

}

Truck::Truck(string name, string manufacturer, int speed, float load\_capacity)

{

this->name= name;

this->manufacturer= manufacturer;

this->speed= speed;

this->load\_capacity = load\_capacity;

}

Truck::~Truck()

{

}

Bus.cpp:

#include "Bus.h"

void Bus::print()

{

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "TRUCK INFO" <<

"\nName: " << name <<

"\nManufacturer: " << manufacturer <<

"\nMax speed: " << speed <<

"\nPlace: " << place << endl;

cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl << endl;;

}

Bus::Bus(string name, string manufacturer, int speed, int place)

{

this->name = name;

this->manufacturer = manufacturer;

this->speed = speed;

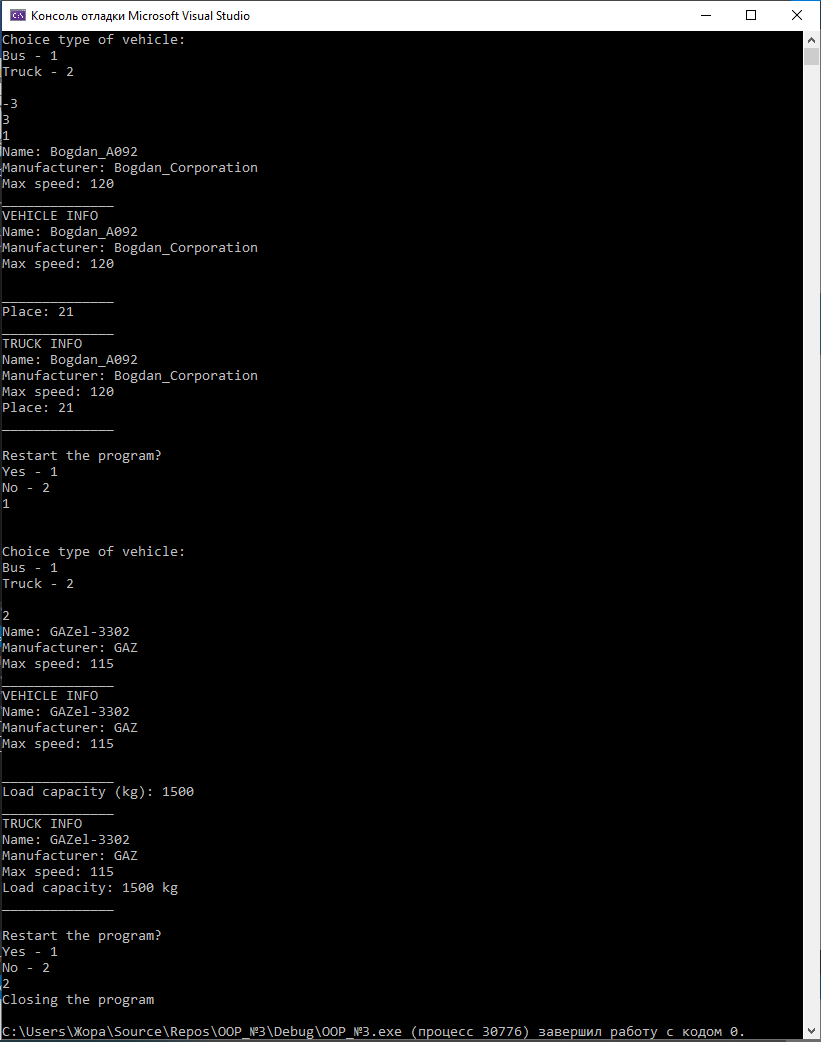
this->place = place;

}

Bus::~Bus()

{

}

**Результати:** 

**Контрольні запитання:**

**1. Що означає поняття спадкування?**

Спадкування  — це один з принципів об'єктно-орієнтовного програмування, який дає класу можливість використовувати програмний код іншого (базового) класу, доповнюючи його своїми власними деталями реалізації.

**2. Який клас називається базовим, а який клас є нащадком?**

Коли один клас наслідується від іншого класу, клас, який наслідується, називають класом-нащадком, а той клас, від якого наслідуються, називають батьківським класом.

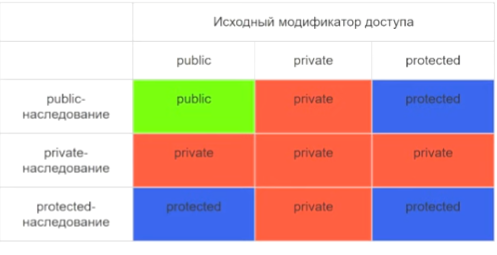
**3. Скільки базових класів може мати похiдний клас?**

При одиночному успадкуванні в кожного **похідного класу** є лише один **базовий клас**, а при множинному — декілька.

**4. Чи може похiдний клас бути базовим?**

Так, похiдний клас бути базовим

**5. Як змінюється доступ до елементів базового класу при спадкуваннi з різними специфікаторами доступу: з розділів класу, із програми, з інших класів,**

****

**6. У чому різниця між простим і множинним спадкуванням?**

Множи́нна спадко́вість — властивість деяких обʼєктно-орієнтованих мов програмування, в яких класи можуть успадкувати поведінку і властивості більш ніж від одного суперкласу (безпосереднього батьківського класу). Це відрізняється від простого спадкування, у випадку якого клас може мати тільки один суперклас.

**7. Чи можна з класу-нащадка одержати доступ до рrivate частини базового класу. якщо специфікатор доступу при спадкуваннi private?**

Ні, з класу-нащадка не можна одержати доступ до рrivate частини базового класу. якщо специфікатор доступу при спадкуваннi private

**8. Чи успадковуються конструктори, деструктори?**

Конструктори і деструктори базових класів не успадковуються. Однак конструктори базових класів викликаються, коли створюється об'єкт похідного класу, а деструктори – коли він знищується.

**9. Що таке вiртуальна функція,**

Віртуальна функція — метод об'єкта в об'єктно-орієнтованому програмуванні, різний для базового класу і класу нащадка. Концепція віртуальної функції вирішує наступну проблему: У ООП, якщо клас-нащадок наслідується від базового класу, об'єкт екземпляр класу-нащадка може використовуватись або як екземпляр батьківського класу (бути приведеним до батьківського класу), або як екземпляр класу-нащадка. Якщо у класі-нащадку є функції, що перекривають (мають таку ж сигнатуру) функції із батьківського класу, то поведінка при виклику таких методів (при використанні даного об'єкта як екземпляра батьківського класу) є невизначеною. Відмінність між віртуальністю і невіртуальністю функцій вирішує цю невизначеність. Якщо функція описана як віртуальна у базовому класі, тоді буде викликана функція із класу нащадка (якщо така існує). Якщо вона не віртуальна, тоді — із батьківського класу.

**10. Чи можна віртуальну функцію визначити як static, вiдповiдь пояснити.**

Ні, бо специфікатор static означає приналежність до певного класу, а не до його конкретного об’єкту/методу. При спробі викликати метод абстратного класу він не буде знати, яку версію похідного класу викликати, бо з ним не зв’язан жодний об’єкт.