

Інститут спеціального зв'язку та захисту інформації

Національного технічного університету України  
"Київський політехнічний інститут ім. Ігоря Сікорського"

Спеціальна кафедра №5

## **ЛАБОРАТОРНА РОБОТА**

з навчальної дисципліни  
**«Об'єктно-орієнтоване програмування»**

**Тема:** Дослідження механізмів реалізації успадкування в  
додатках, розроблених мовою об'єктно-орієнтованого  
програмування.

**Виконав:** курсант Жванський Роман

**Перевірив:** доцент Спеціальної кафедри №5  
Куліков В.М.

**Київ 2023**

## **Варіант №3**

### **I. Дослідження засобів створення ієрархій класів в мові C++.**

#### **1.1 Завдання**

##### **Завдання 1**

Створити клас машина, який має марку (показчик на рядок), кількість циліндрів, потужність. Визначити конструктори, деструктор і функцію друку. Створити public-похідний клас - вантажівки, який має вантажопідйомність кузова. Визначити конструктори за замовчуванням і з різною кількістю параметрів, деструктори, функцію друку. Визначити функції перепризначення марки і вантажопідйомності.

##### **Завдання 2**

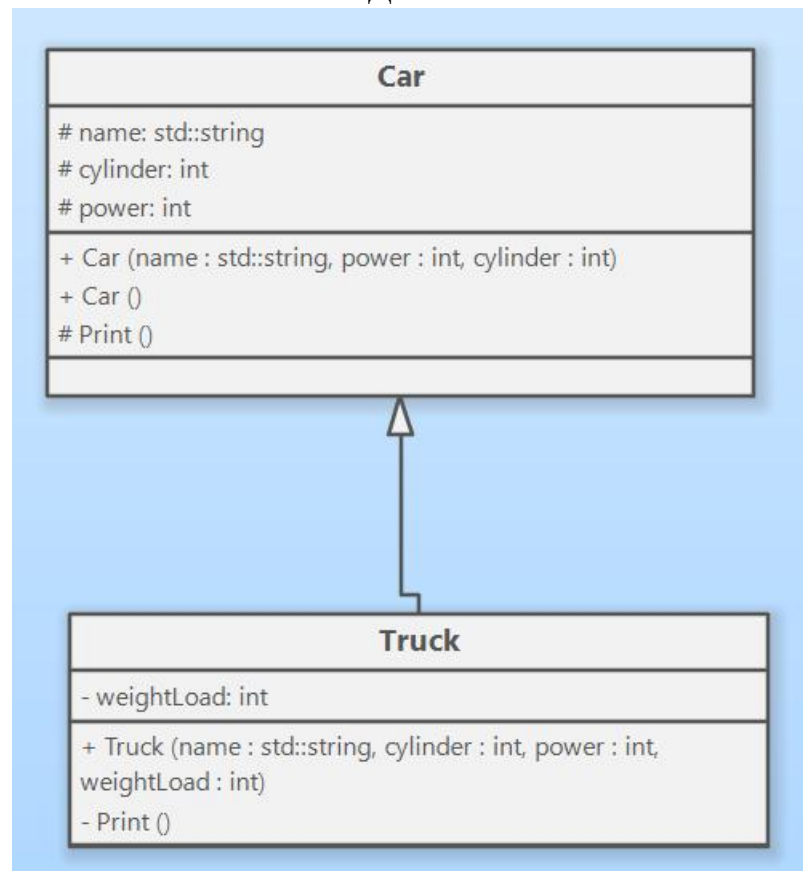
Створити клас двигун, який має потужність. Визначити конструктори і метод доступу Створити клас машина, що містить клас двигун. Додатково є марка (показчик на рядок), ціна. Визначити конструктори і деструктор. Визначити public- похідний клас вантажівка, який має додатково вантажопідйомність. Визначити конструктори, деструктори і функцію друку.

##### **Завдання 3**

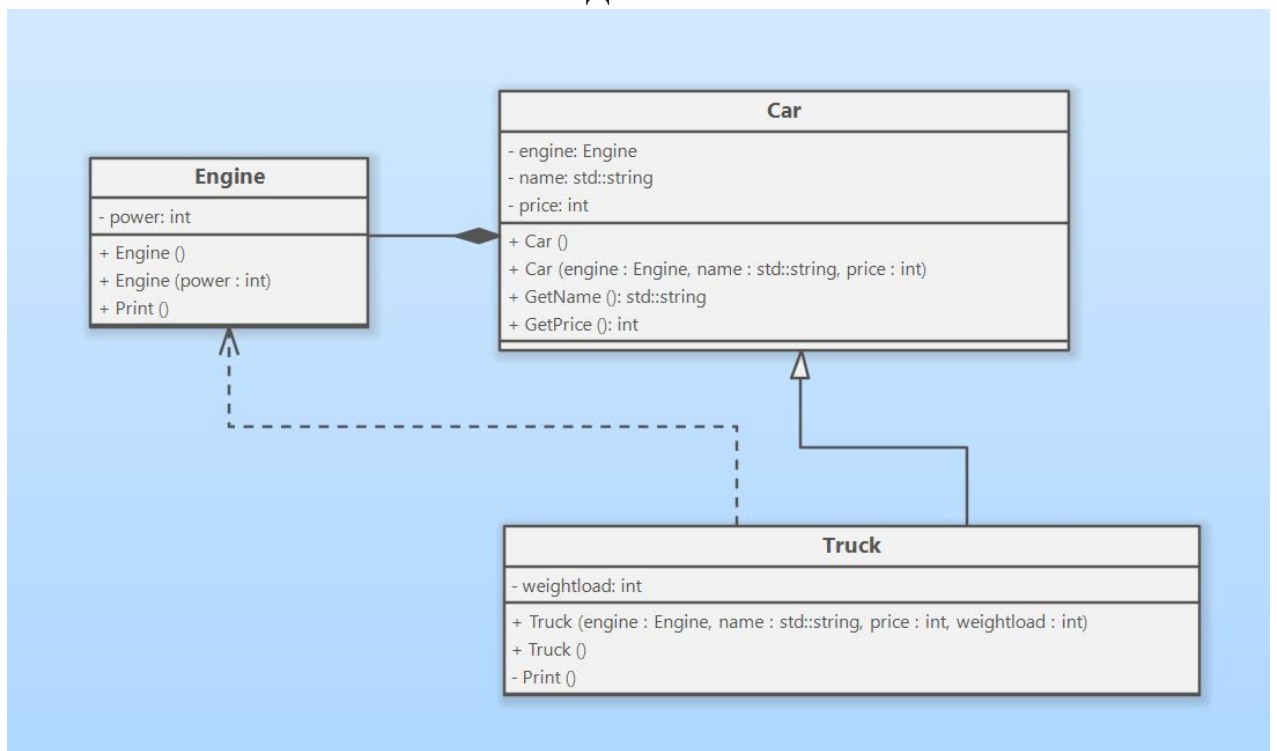
Створити ієрархію класів меблі і стіл. Перевизначити вивід в потік і ввід з потоку, конструктор копіювання, оператор присвоєння через відповідні функції базового класу.

## 1.2 Об'єктні моделі мовою UML (діаграми)

### Завдання 1



### Завдання 2



## Завдання 3

Furniture	Table
# name: std::string	- place: int
+ «create» Furniture (name : std::string)	+ «create» Table (name : std::string, place : int)
+ «create» Furniture (other : Furniture)	+ «create» Table (other : Table)
+ «create» Furniture ()	+ «create» Table ()
+ «operator» = (other : Furniture): Furniture	+ «operator» = (other : Table): Table
+ «operator» >> (input : std::istream, obj : Furniture): std::istream	+ «operator» >> (input : std::istream, obj : Table): std::istream
+ «operator» << (output : std::ostream, obj : Furniture): std::ostream	+ «operator» << (output : std::ostream, obj : Table): std::ostream

### 1.3 Тексти програм(з коментарями, що пояснюють відповідність розроблених програм варіанту завдання)

#### Завдання 1

```
#include <iostream>

class Car
{
public:
    Car() : name(new std::string("unknown")), cylinder(0), power(0) {};
    Car(std::string* name, int power, int cylinder) :name(name),
cylinder(cylinder), power(power) {};
    ~Car() {};
    virtual void Print();
protected:
    std::string* name;
    int cylinder, power;
};

void Car::Print()
{
    std::cout << "Name : " << *name << "\n";
    std::cout << "Cyliders : " << cylinder << "\n";
    std::cout << "Power : " << power << " HP\n";
}

class Truck : public Car
{
public:
    Truck(std::string* name, int cylinder, int power, int weightLoad) : Car(name,
power, cylinder), weightLoad(weightLoad) {};
```

```

        virtual void Print();
private:
        int weightLoad;
};

void Truck::Print()
{
    Car::Print();
    std::cout << "Weight load : " << weightLoad << " tons\n";
}

int main()
{
    std::string name = "Audi";
    std::string* name_ptr = &name;
    int cylinder = 10;
    int power = 300;
    int weightLoad = 5;

    Truck truck(name_ptr, cylinder, power, weightLoad);
    truck.Print();
}

```

## Завдання 2

```

#include <iostream>

class Engine
{
public:
    Engine() :power(0) {};
    Engine(int power) :power(power) {};
    void Print() { std::cout << "Engine power: " << power; };
private:
    int power;
};

class Car
{

```

```

public:
    Car() :name(new std::string("unknown")), price(0), engine(0) {};
    Car(Engine& engine, std::string& name, int price) : engine(engine),
price(price), name(new std::string(name)){};
    std::string& GetName() { return *name; };
    int GetPrice() { return price; };
private:
    Engine engine;
    std::string* name;
    int price;
};

class Truck : public Car
{
public:
    Truck(Engine& engine, std::string& name, int price, int
weightload) :Car(engine, name, price), weightload(weightload) {};
    ~Truck(){};
    virtual void Print();
private:
    int weightload;
};

void Truck::Print()
{
    std::cout << "Name: " << GetName()<<"\n";
    std::cout << "Weightload: " << weightload<<" tone \n";
    std::cout << "Price: " << GetPrice() << "$\n";
}

int main()
{
    Engine engine(30);
    std::string name = "Audi";
    Truck truck(engine, name, 100000, 5);
    truck.Print();
    engine.Print();
    return 0;}

```

## Завдання 3

```
#include <iostream>

class Furniture
{
public:
    Furniture() :name("Unknown") {};
    Furniture(std::string name) :name(name) {};
    Furniture(Furniture& other) : name(other.name) {};
    ~Furniture() {};
    virtual Furniture& operator=(Furniture& other);
    friend std::istream& operator>>(std::istream& input, Furniture& obj);
    friend std::ostream& operator<<(std::ostream& output, Furniture& obj);
protected:
    std::string name;
};

Furniture& Furniture::operator=(Furniture& other)
{
    name = other.name;
    return *this;
}

std::istream& operator>>(std::istream& input, Furniture& obj)
{
    input >> obj.name;
    return input;
}

std::ostream& operator<<(std::ostream& output, Furniture& obj)
{
    output << obj.name;
    return output;
}
```

```

class Table : public Furniture
{
public:
    Table() :place(0) {};
    Table(std::string name, int place) :Furniture(name), place(place) {};
    Table(Table& other) : place(other.place) {}
    ~Table() {};
    Table& operator=(Table& other);
    friend std::istream& operator>>(std::istream& input, Table& obj);
    friend std::ostream& operator<<(std::ostream& output, Table& obj);

private:
    int place;
};

Table& Table::operator=(Table& other)
{
    this->place = other.place;
    return *this;
}

std::istream& operator>>(std::istream& input, Table& obj)
{
    input >> obj.name >> obj.place;
    return input;
}

std::ostream& operator<<(std::ostream& output, Table& obj)
{
    output << obj.name << " " << obj.place;
    return output;
}

int main()
{

```

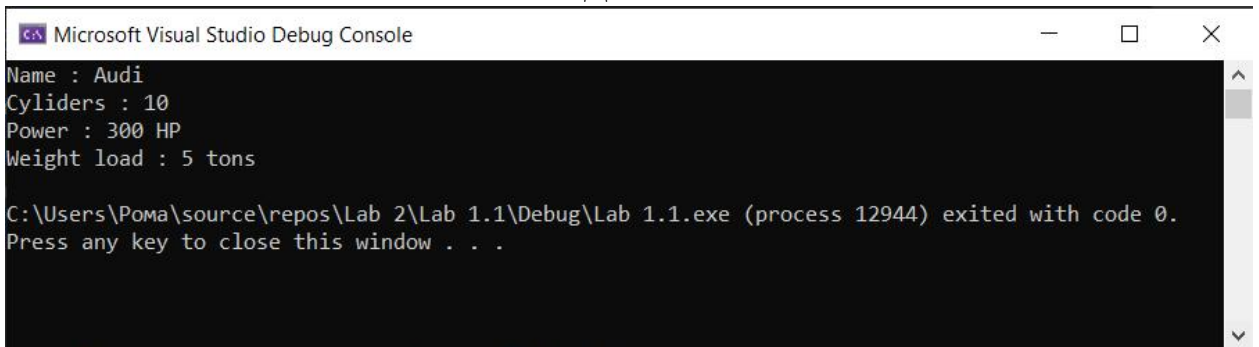


```

    Table t1("Table1", 4);
    Furniture f("Abrams");
    std::cout << t1 << std::endl;
    std::cin >> t1;
    std::cout << t1 << std::endl;
    std::cout << f << std::endl;
    return 0;
}

```

## 1.4 Висновки: Завдання 1



```

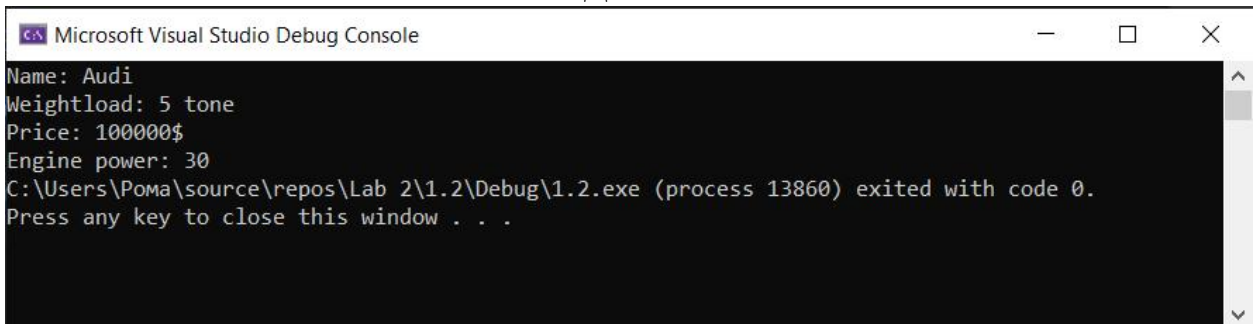
Microsoft Visual Studio Debug Console
Name : Audi
Cylinders : 10
Power : 300 HP
Weight load : 5 tons

C:\Users\Роман\source\repos\Lab 2\Lab 1.1\Debug\Lab 1.1.exe (process 12944) exited with code 0.
Press any key to close this window . . .

```

Клас працює вірно. Програма може зберігати інформацію про машину(марку, кількість циліндрів, потужність) також є додатковий клас, який може зберігати інформацію про вантажопідйомність кузова.

## Завдання 2



```

Microsoft Visual Studio Debug Console
Name: Audi
Weightload: 5 tone
Price: 100000$
Engine power: 30

C:\Users\Роман\source\repos\Lab 2\1.2\Debug\1.2.exe (process 13860) exited with code 0.
Press any key to close this window . . .

```

Клас працює вірно. Програма зберігає деяку інформацію про машину(марку, двигун, ціну, вантажопідйомність кузова)

## Завдання 3

```
Microsoft Visual Studio Debug Console

Table1 4
Drommel
6
Drommel 6
FILD

C:\Users\Роман\source\repos\Lab 2\Lab 1.3\Debug\Lab 1.3.exe (process 13708) exited with
code 0.
Press any key to close this window . . .
```

Клас працює вірно. Програма зчитує з консолі назву столу, та кількість місць за ним. Також виводить цю інформацію в консолі.

## II. Дослідження поліморфічних кластерів і віртуальних функцій.

### 2.1 Завдання

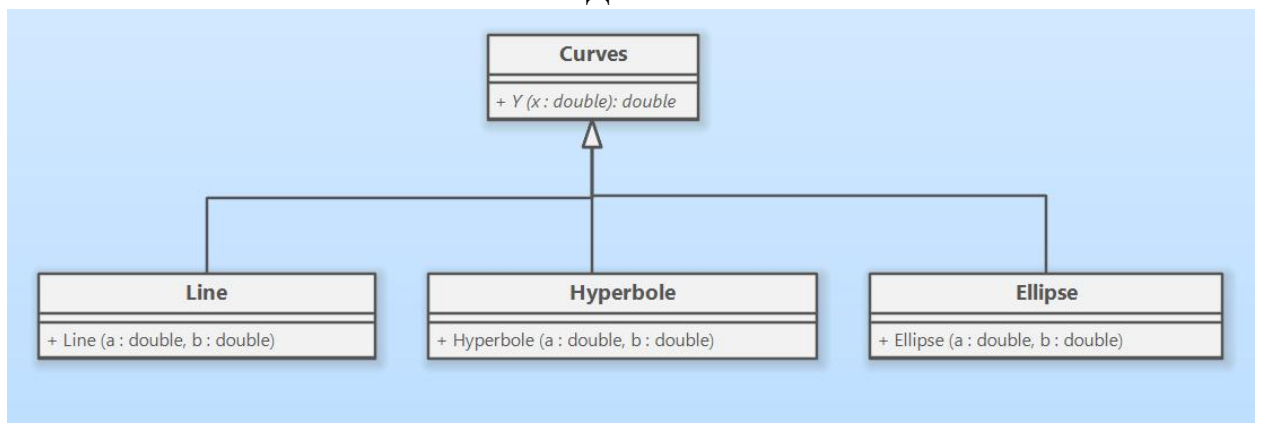
#### Завдання 1

Створити абстрактний клас (криві) для обчислення координати  $y$  для деякої  $x$ . Створити похідні класи: пряма, еліпс, гіпербола із своїми функціями обчислення  $y$  в залежності від вхідного параметра  $x$ .

Рівняння прямої:  $y=ax+b$ , еліпса:  $x^2/a^2+y^2/b^2=1$ , гіперболи:  $x^2/a^2-y^2/b^2=1$

### 2.2 Об'єктні моделі

#### Завдання 1



### 2.3 Тексти програм

#### Завдання 1

```
#include <iostream>
#include <cmath>
```

```

class Curves
{
public:
    virtual double Y(double x) = 0;
};

class Line : public Curves
{
public:
    Line(double a, double b) : a(a), b(b) {};
    double Y(double x) override { return a * x + b; };
private:
    double a, b;
};

class Ellipse : public Curves
{
public:
    Ellipse(double a, double b) : a(a), b(b) {};
    double Y(double x) override { return b * sqrt(1 - (x * x) / (a * a)); };
private:
    double a, b;
};

class Hyperbole : public Curves
{
public:
    Hyperbole(double a, double b) : a(a), b(b) {};
    double Y(double x) override { return sqrt(((b * b) * (x * x)) / (a * a) - (b *
b)); };
private:
    double a, b;
};

```

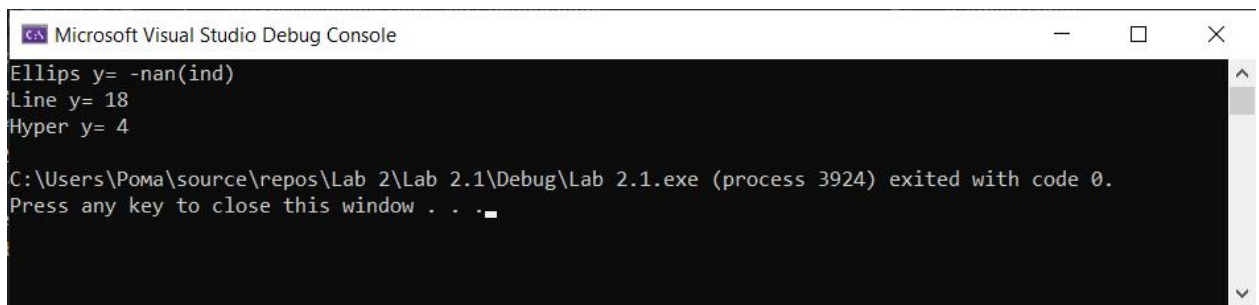
```

int main()
{
    double a = 3, b = 3;
    double x = 5;
    Curves* ellips = new Ellipse(a, b);
    Curves* line = new Line(a, b);
    Curves* hyper = new Hyperbole(a, b);
    std::cout << "Ellips y= " << ellips->Y(x) << "\n";
    std::cout << "Line y= " << line->Y(x) << "\n";
    std::cout << "Hyper y= " << hyper->Y(x) << "\n";
}

```

## 2.4 Висновок

### Завдання 1



```

Microsoft Visual Studio Debug Console
Ellips y= -nan(ind)
Line y= 18
Hyper y= 4

C:\Users\Роман\source\repos\Lab 2\Lab 2.1\Debug\Lab 2.1.exe (process 3924) exited with code 0.
Press any key to close this window . . .

```

Клас працює вірно. Програма може вираховувати значення геометричних фігур(еліпс, пряма, гіпербола).

### Контрольні питання

- 1) Ієрархічні схеми (види) успадкування - в ієрархічному успадкуванні більше одного класу успадковується від одного базового класу.
- 2) Механізм успадкування. Порядок виклику конструкторів -це властивість класу отримувати програмний код (навички) іншого (базового) класу, додаючи до нього свій власний код, тим самим розширюючи його можливості. Якщо два класи утворюють ієрархію спадковості, то при створенні екземпляру похідного

класу спочатку викликається конструктор базового класу який конструює об'єкт похідного класу.

- 3) Керування доступом при успадкуванні-модифікатори доступу `private`, `protected`, `public` можуть застосовуватись до базових класів при їх успадкуванні похідними класами. Дія того чи іншого модифікатора впливає на рівень доступності елементів базових класів.
- 4) Віртуальні функції і поліморфні кластери- це технологія виклику віртуальних функцій, що реалізовані в ієрархічно зв'язаних класах. Віртуальна функція в мові - це особливий тип функції, яка, при її виклику, виконує «найдоцільніший» метод, який існує між батьківським і дочірніми класами.
- 5) Чисті віртуальні функції та абстрактні базові класи - віртуальна функція без коду має назву чиста віртуальна функція. Абстрактний клас – це клас, який містить хоча б одну чисто віртуальну функцію.
- 6) Технічна реалізація віртуальних функцій - віртуальні функції реалізуються за допомогою вказівників на функції. Для позначення перевизначення віртуальної функції використовують “`override`”.