ЛАБОРАТОРНА РОБОТА №4

Тема: Успадковування класів

Мета: ознайомитись зі способами та механізмами успадкування класів та навчитись використовувати їх для побудови об’єктно-орієнтованих програм.

КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ

Класи використовуються для моделювання концепцій реального та програмного світу. Однак жодна концепція не існує ізольовано. Вона співіснує зі спорідненими концепціями і саме цьому зв'язку вона є сильною. Поняття похідного класу і пов'язані з ним механізми мови призначені для вираження ієрархічних відносин, тобто для відображення спільності класів. Наприклад, концепції кола і трикутника пов'язані тим, що обидва об’єкти є фігурами. Концепція фігури є спільною для них. Тому ми повинні явно визначити, що класи *Circle* і *Triangle* мають загальний базовий клас *Shape*. Представлення понять «коло» і «трикутник» в програмі без введення поняття «фігура» означало б втрату чогось істотного.

Базові та похідні класи

Розглянемо наведений раніше приклад. І коло, і трикутник є фігурами, однак якщо не використовувати спеціальних засобів для явного опису цього факту, компілятор сам не зможе зробити подібний висновок. Відповідно, ми не зможемо, наприклад, помістити покажчики на об'єкти класів *Circle* і *Triangle* в один список. Відношення між класами у якому існують порождуючі (базові) класи називають **успадкуванням.**

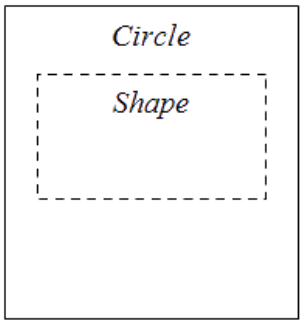
Клас може бути *породжено* з іншого класу, який називається *базовим* класом *похідного* класу. Клас може бути породжено від одного або декількох класів. Породжені класи успадковують властивості базових класів, включаючи дані та функції члени. Крім того, в похідних класах можуть бути оголошені **додаткові** дані та функції члени.

Оголошення похідного класу має наступний синтаксис:

class <*ім'я похідного* ***класу***> **:** <*специфікатор доступу*> <*назва базового класу*>
[, <*Специфікатор доступу*> <*назва базового класу*>]
(...);

Відношення успадкування між класами може бути представлено у графічному вигляді наступним чином:

Популярною та ефективною реалізацією поняття похідних класів є представлення об'єкта похідного класу у вигляді об'єкта базового класу та інформації, що відноситься тільки до похідного класу.



Похідний клас може сам, у свою чергу, служити базовим класом.

Якщо члени базового класу не були перевизначені в похідних класах, тоді вони позначаються і трактуються так само, як і члени похідного класу. В такому випадку кажуть, що члени базового класу **успадковуються** похідним класом. Операція вирішення області **видимості ::** може вживатися для явного посилання на член базового класу. Це забезпечує доступ до імені, яке може бути перевизначеним в похідних класах.

```
class Base
{
public:
      int a, b;
};

class Derived : public Base
{
public:
      int b;
      int c;
};

Derived d;
d.a      = 1;        // Ініціалізація a, успадкованого з класу Base
d.Base::b = 2;       // Ініціалізація b з класу Base
d.b      = 3;        // Ініціалізація b, оголошеного в класі Derived
d.c      = 4;
Base *bp  = &d;  // Перетворимо покажчик на клас Derived у покажчик на
Base
```

Клас називається *безпосереднім базовим класом,* якщо він згадується при оголошенні похідного класу, і *непрямим базовим класом,* якщо він є базовим класом для одного з базових класів оголошуємо класу.

```
class A
{
public:
      void f();
};
class B : public A
{
};
// Клас B є безпосереднім базовим класом для класу C,
// а клас A – непрямим базовим класом для класу C
class C : public B
```

```
{
public:
    void f();
    void ff();
};
```

У записі *<ім'я класу>::<ім'я>* – *ім'я класу* може бути ім'ям непрямого базового класу. *Ім'я класу* визначає клас, в якому починається пошук *імені.*

```
void C::ff()
{
    f();    // Виклик функції f() з класу C
    A::f(); // Виклик функції f() з класу A
    B::f(); // Знову виклик функції f() з класу A,
        // тому що у класі B функцію f() не визначено
}
```

Специфікатори доступу для базових класів

Специфікатор доступу може приймати значення:

- *public* - у цьому випадку публічні члени базового класу стають публічними членами похідного класу, а захищені члени базового класу стають захищеними членами похідного класу;
- *protected* - у цьому випадку публічні і захищені члени базового класу стають захищеними членами похідного класу;
- *private* - у цьому випадку публічні і захищені члени базового класу стають приватними членами похідного класу.

Приватні члени класу недоступні у похідних класах, якщо тільки він не оголошений як дружній.

У випадку коли специфікатор доступу не вказано (за замовчанням) використовується специфікатор доступу *private.*

```
class Base
{
private:
    int a;
protected:
    int b;
public:
    int c;
};
class Derived1 : public Base
{...  // A недоступний
     // B - захищений член класу Derived1
     // C - публічний член класу Derived1
};
class Derived2: protected Base
{...  // A недоступний
     // B і C - захищені члени класу Derived2
};
class Derived3: private Base
{...  // A недоступний
     // B і C - приватні члени класу Derived3
```

```
};
```

Конструктори і деструктор

Похідні класи можуть також потребувати конструктора. **Конструктори не успадковуються, вони повинні бути визначені в самому класі.** Якщо базовий клас має конструктор, він повинен бути викликаний. Конструктори замовчуванням можуть бути викликані неявно. Однак якщо всі конструктори базового класу вимагають зазначення аргументів, то конструктор цього базового класу повинен бути викликаний явним чином. Аргументи конструктора базового класу вказуються у визначенні конструктора похідного класу.

```
class Base
{
private:
    int a;
protected:
    int b;
public:
    Base(int aa, int bb) : a(aa), b(bb) {}
};

class Derived : public Base
{
private:
    int c;
public:
    Derived(int aa, int bb, int cc)
        : Base(aa, bb)  // Ініціалізація базового класу
        , c(cc)         // Ініціалізація членів похідного класу
    {
    }
};
```

Конструктор похідного класу може мати ініціалізатори для своїх власних членів та членів базового класу, але він не може безпосередньо ініціалізувати члени базового класу.

```
class Derived : public Base
{
private:
    int c;
public:
    Derived(int aa, int bb, int cc)
        : a(aa), b(bb)         // Помилка - a і b не оголошено в класі Derived
        , c(cc)
    {
    }
};
```

Об'єкти класу створюються знизу вгору: спочатку базовий клас, потім похідний, а знищуються в протилежному порядку. Дані базових класів конструюються в порядку їх оголошення в класі і знищуються в зворотному порядку.

Використання захищених членів класу

Проста модель приховування даних «відкритий / закритий» добре працює для конкретних типів. Однак при використанні похідних класів існує два види користувачів класу: похідні класи та всі інші функції. Модель «відкритий / закритий» дозволяє програмісту розрізняти розробників і всіх інших, але вона не передбачає особливого обслуговування для похідних класів.

Мова C++ дозволяє здійснювати гнучке управління доступом до членів класу за рахунок використання трьох специфікаторів доступу. Однак захищеними членами класу можна більше зловживати, ніж приватні. Вміщення значної частини даних у загальний клас, доступний для всіх похідних класів, призводить до ризику руйнування цих даних. Більш того, так само як і відкриті, захищені дані не просто реструктурувати через складності знаходження всіх випадків їх використання. Таким чином, захищені дані часто приводять до проблем супроводу.

На щастя, немає необхідності використовувати захищені дані. Члени класів за замовчуванням закриті і, як правило, це є найкращим варіантом. При розробці похідного класу ви може використовувати інтерфейс базового класу для роботи з приватними членами базового класу.

Зверніть увагу, що всі ці заперечення не мають великого значення для захищених *функцій.* Захищеність - це прекрасний спосіб визначення операцій для використання в похідних класах.

ПОРЯДОК ВИКОНАННЯ РОБОТИ

1. Напишіть програму згідно наступного завдання 1.

Завдання 1. Уявіть собі видавничу компанію, яка торгує книгами і аудіо-записами цих книг. Створіть клас Publication, в якому зберігаються назва (string) і ціна (float) книги. Від цього класу успадковуються ще два класи: Book, який додатково містить інформацію про кількість сторінок у книзі (int), і Type, який додатково містить час запису книги у хвилинах (float). У кожному з цих трьох класів повинен бути метод inputData(), через який можна отримувати дані від користувача з клавіатури, і outputData(), призначений для виведення цих даних.

Напишіть функцію main() програми для перевірки класів Book і Type. Створіть їх об'єкти в програмі і запросіть користувача ввести і вивести дані з використанням методів inputData() і outputData ().

2. Напишіть програму згідно завдання 2.

Завдання 2. До класів з попереднього завдання (попередньо зберігши окремо код) додайте базовий клас Sales, в якому міститься масив, що складається з трьох значень типу float, куди можна записати загальну вартість проданих книг за останні три місяці. Включіть в клас методи inputData() для отримання значень вартості від користувача і outputData() для виведення цих цифр. Змініть класи Book і Type так,

щоб вони стали похідними обох класів: Publication і Sales. Об'єкти класів Book і Type повинні вводити і виводити дані про продажі разом з іншими своїми даними. Напишіть функцію main() для створення об'єктів класів Book і Type, щоб протестувати можливості введення/виведення даних.

3. Реалізувати два класи згідно варіанту індивідуального завдання, які будуть похідними для класу розробленого у лабораторній роботі №2.

Похідні класи повинні містити:

- конструктор по замовчуванню;
- параметризований конструктор;
- конструктор копіювання;
- деструктор;
- inputData() – метод для запит у користувача даних та їх зчитування з клавіатури у поля об’єкта класу;
- print() – константний метод виводу даних на екран;
- методи доступу до закритих даних (сетери і гетери).

ВАРІАНТИ ЗАВДАНЬ

Варіант 1.	Варіант 2.

class EducationalToy : public Toy {
  string subject; //навчальний предмет
  int complexityLevel;//рівень складності
public:
  // конструктори, сетери, гетери,
  // методи, деструктор
};
 	
class BoardGameToy : public Toy {
  int numberOfPlayers; //
  string rules; //
public:
  // конструктори, сетери, гетери,
  // методи, деструктор
};
	class Mammal : public Animal {
	  string furColor; //колір хутра
	  bool isCarnivorous; // є хижим
	public:
	  // конструктори, сетери, гетери,
	  // методи, деструктор
	};
	
	class Bird : public Animal {
	  float wingSpan; // розмах крил
	  bool canFly; // чи може літати
	public:
	  // конструктори, сетери, гетери,
	  // методи, деструктор
	};

Варіант 3.

```
class Apartment : public House {
   int floorNumber; // номер поверху
   bool hasBalcony; // наявність балкона
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};

class Villa : public House {
   double gardenSize; // площа саду
   bool hasPool;  // наявність басейну
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
```

Варіант 4.

```
class Flower : public Plant {
   string color;  // колір
   int numPetals; // кількість пелюсток
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};

class Tree : public Plant {
   string treeType; // тип дерева
   int age; // вік дерева
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
```

Варіант 5.

```
class CapitalCity : public City {
    string country; // країна, до якої належить столиця
    int population; // населення столиці
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};

class IndustrialCity : public City {
    // Обсяг промислового виробництва
    float industrialProduction;
    // Рівень безробіття
    float unemploymentRate;
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
```

Варіант 6.

```
class City : public Country {
   string cityName; // назва міста
   int area;       // площа міста
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};

class State : public Country {
   string capital;    // столиця штату
   int numRegions;   // к-сть областей
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
```

Варіант 7.

```
class MilitaryFighter : public Airplane{
   int missiles;  // кількість ракет
   bool stealthMode; // режим невидимості
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};

class CargoAirplane : public Airplane {
   int cargoCapacity; // вмістимість
вантажу
   string cargoType; // тип вантажу
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
```

Варіант 8.

```
class Fruit : public Food {
    string vitamins; // вітаміни
    string taste;    // смак
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};

class Dessert : public Food {
    int sugarContent; // вміст цукру
    string ingredients; //склад
десерту
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
```

Варіант 9.

```
class Fruit : public Vegetable {
    string fruitType; // тип фрукту
(яблуко, банан)
    string taste; // смак фрукту
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};

class LeafyVegetable: public Vegetable {
    string leafyType; // тип листяного
овоча (салат, шпинат),
    int moisture; // вологість
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
```

Варіант 10.

```
class Chair : public Furniture {
   string material; // матеріал
   int numberOfLegs;  // кількість
ніжок
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};

class Sofa: public Furniture {
   int seatingCapacity; // к-сть місць
для сидіння
   int hasSleeper;  // наявність
механізму для розкладання в ліжко
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
```

Варіант 11.

```
class NavigableRiver : public River {
   bool isNavigable; // чи є річка судноплавною
   int pollutionLevel; // рівень забруднення
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
class Tributary : public River { //
   char *parentRiver; // назва батьківської річки
   int temperature; // середня температура води
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
```

Варіант 12.

```
class Server : public Computer {
    int numCores; // кількість ядер
    int ramSize; // об'єм оперативної пам'яті в ГБ
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};

class Laptop : public Computer {
    float screenSize; // розмір екрану
    int batteryLife; // час роботи від акумулятора в годинах
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
```

Варіант 13.

```
class HourlyWorker : public Worker {
private:
    int hoursWorked; // к-сть годин роботи
    float allowanceFactor; //коефіцієнт надбавки
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
class SalariedWorker : public Worker {
private:
    int monthlySalary; місячна зарплата
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
```

Варіант 14.

```
class SoftwareFirm : public Firm {
    int numberOfDevelopers;
    long authorizedCapital; //статутний капірал
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
class ConstructionFirm : public Firm {
   int numberOfProjects; //к-сть проектів
   int customersNumber; //к-сть клієнтів
public:
   // конструктори, сетери, гетери,
   // методи, деструктор
};
```

Варіант 15	Варіант 16

class Shirt : public Clothing {
int size; // розмір сорочки
string style; // стиль одягу
public:
// конструктори, сетери, гетери,
// методи, деструктор
};
 	
class Sweater : public Clothing {
string material; // матеріал светра
string pattern; / візерунок светра
public:
// конструктори, сетери, гетери,
// методи, деструктор
};
	class SportsCar : public Car {
	int maxSpeed; // максимальна швидкість
	string engineType; // тип двигуна
	public:
	// конструктори, сетери, гетери,
	// методи, деструктор
	};
	class ElectricCar : public Car {
	double range; // діапазон їзди
	string batteryType; // тип батареї
	public:
	// конструктори, сетери, гетери,
	// методи, деструктор
	};

ОФОРМЛЕННЯ ЗВІТУ:

1. Титульний лист.
2. Мета роботи.
3. Короткі теоретичні відомості.
4. Текст завдання згідно варіанту.
5. Код програми.
6. Знімок вікна виконання програми.
7. Висновки.