Проектування і реалізація програми з ієрархією класів (одиночне і множинне успадкування)

Мета роботи — засвоєння поняття успадкування та його принципів; набуття практичних навичок з оголошення та використання ієрархій класів.

Основні завдання роботи

Розробити та реалізувати програмно ієрархію класів для роботи з рахунком вкладника банку. Для цього слід виконати такі завдання:

1. Розробити та реалізувати програмно базовий клас Rahunok.
2. Розробити та реалізувати програмно базовий клас Vkladnyk.
3. Розробити та реалізувати програмно похідний клас RahunokVkladnyka.
4. У головній частині програми продемонструвати роботу створеної ієрархії класів.

Основні теоретичні відомості

1. Поняття успадкування

Успадкування (inheritance) — це процес, завдяки якому об’єкт може набувати властивостей іншого об’єка з додаванням до них ознак, властивих тільки йому.

Успадкування — один з трьох базових принципів ООП. Завдяки успадкуванню підтримується концепція ієрархії класів. Застосування ієрархії класів робить керованими великі потоки інформації.

2. Доступ до елементів базового класу

Клас, властивості якого успадковуються, називається **базовим**, а клас, який успадковує властивості базового, —

похідним.

Для оголошення успадкування використовується така загальна форма:

```
class ім'я_похідного_класу : доступ
                                    ім'я_базового_класу
{
 ... // елементи похідного класу
}
```

Тут *доступ* — одне з трьох ключових слів: *public, private* або *protected.*

Специфікатор доступу визначає, як елементи базового класу успадковуються похідним класом. Якщо специфікатором доступу успадкованого базового класу є *public,* то всі відкриті члени базового класу стають відкритими і в похідному. Якщо специфікатором доступу успадкованого базового класу є *private*, то всі відкриті члени базового класу стають закритими в похідному. В обох випадках всі закриті члени базового класу залишаються закритими і недосяжними для похідного класу.

Якщо специфікатором доступу є *private*, то відкриті члени базового класу стають закритими в похідному, проте ці члени залишаються доступними для функцій-членів похідного класу.

Приклад 1. Доступ до членів базового класу всередині похідного:

```
class base {
     int x;
  public:
     void setx(int n) { x = n; }
     void showx() { cout << x << '\n'; }
};
// Успадкування базового класу як public
class derived : public base {
     int y;
  public:
    void sety(int n) {y=n;}
/* Зараз буде помилка. Закритий член x базового класу
     недосяжний всередині похідного */
```

```
    void show_sum(){cout << x+y << '\n'; } // Помилка!

    void showy(){cout << y << '\n'; }
    void show() {
      showx(); /* Відкритий член базового класу
                  досяжний всередині похідного */
      showy();
    }
};
```

У цьому прикладі в похідному класі зроблено спробу доступу до змінної *x*, яка є закритим членом *base*. Це помилка, оскільки закриті члени базового класу залишаються закритими, незалежно від того, як він успадковується.

3. Захищені члени класу

Іноді можлива ситуація, коли члени базового класу, залишаючись закритими, були досяжні для похідного класу. Для реалізації цієї ідеї в C++ уведено специфікатор доступу *protected* (захищений).

Специфікатор доступу *protected* еквівалентний *private* з єдиною різницею: захищені члени базового класу досяжні для членів усіх похідних класів цього базового класу. Поза базовим класом або похідними класами захищені члени недосяжні.

Специфікатор доступу *protected* може бути у будь-якому місці оголошення класу, хоча, як правило, його розміщують після оголошення закритих членів та перед оголошенням відкритих членів. Повна загальна форма оголошення класу має такий вигляд:

```
class ім'я_класу
{
   ... // закриті члени
  protected:
   ... // захищені члени
  public:
   ... // відкриті члени
};
```

Коли захищений член базового класу успадковується похідним класом як відкритий, він стає захищеним членом похідного класу. Якщо базовий клас успадковується як закритий, то захищений член базового класу стає закритим членом похідного класу.

Базовий клас може також успадковуватись похідним класом як захищений. У цьому випадку відкриті та захищені члени базового класу стають захищеними членами похідного класу. Закриті члени базового класу залишаються закритими, і вони не досяжні для похідного класу.

Особливості доступу до елементів базового класу при успадкуванні відображено у табл. 1.

Таблиця 1

Доступ під час успадкування

Специфікатор доступу елемента базового класу	Специфікатор доступу під час успадкування базового класу	Успадкування елементів базового класу похідним класом
public	public	public
protected		protected
private		private
public	protected	protected
protected		protected
private		private
public	private	protected
protected		private
private		private

4. Конструктори, деструктори та успадкування

Базовий клас, похідний клас або обидва можуть мати конструктори та/або деструктори. Якщо і у базового, і у похідного класів є конструктори та деструктори, то конструктори виконуються у порядку успадкування, а деструктори — у зворотному порядку. Таким чином, конструктор базового класу виконується раніше, ніж конструктор похідного класу. Для деструкторів правильний

зворотний порядок: деструктор похідного класу виконується раніше від деструктора базового класу.

5. Передача аргументів для конструкторів базового та похідного класів

Розглядаючи поняття успадкування, необхідно звернути увагу на особливості передачі аргументів для конструкторів похідного та базового класів. Коли ініціалізація виконується тільки в похідному класі, аргументи передаються звичайним шляхом. Проте, під час передавання аргумента конструктору базового класу виникають деякі складності. Передусім усі необхідні аргументи базового та похідного класів передаються конструктору похідного класу. Потім завдяки розширеній формі оголошення конструктора похідного класу відповідні аргументи передаються далі в базовий клас.

Синтаксис передавання аргументів з похідного в базовий клас такий:

```
ім'я_похідного_класу ( список_арг1 ) :
                    ім'я_базового_класу ( список_арг2 )
{
  ... //тіло конструктора похідного класу
}
```

Для базового та похідного класів допустимо використовувати одні й ті самі аргументи. Для похідного класу допустимим є також ігнорування всіх аргументів та передача їх напряму в базовий клас.

У більшості випадків конструктори базового та похідного класів не використовують один і той самий аргумент. Тоді у разі потреби передавання кожному конструктору класу одного або більше параметрів слід передати конструктору похідного класу всі аргументи, необхідні конструкторам цих двох класів. Потім конструктор похідного класу просто передає конструктору базового ті аргументи, які йому потрібні.

Приклад 2. Передавання одного аргумента конструктору похідного класу, а другого — конструктору базового:

```
#include <iostream.h>
class base {
      int i;
    public:
      base(int n) {
          cout << ”Робота конструктора базового класу\n”
          i = n;
     }
  ~base() {cout<<”Робота деструктора базового
                                                класу\n”;}
  void showi() {cout << i << ’\n’; }
}
class derived : public base  {
      int j;
    public:
      derived(int n, int m) : base(m)
      // Передача аргумента в базовий клас
      {
        cout << ”Робота конструктора похідного класу\n”;
        j = n;
       }
      ~derived() { cout << ”Робота деструктора похідного
                                                  класу\n”;}
       void showj() {cout << j << ‘\n’;}
};
void main()
{
   derived o(10,20);
   o.showi();
   o.showj();
}
```

Якщо похідному класу деякий аргумент не потрібен, він його ігнорує і просто передає в базовий клас.

6. Множинне успадкування

Є два способи, завдяки яким похідний клас може успадковувати більше, ніж один базовий клас. По-перше, похідний клас може використовуватися як базовий для іншого похідного класу, створюючи багаторівневу ієрархію класів. В

цьому випадку вихідний базовий клас є **непрямим** (inderect) **базовим класом** для іншого похідного класу. По-друге, похідний клас може прямо успадковувати більше, ніж один базовий клас. У такій ситуації комбінація двох або більше базових класів допомагає створенню похідного класу.

Коли клас використовується як базовий для похідного, який, у свою чергу, є базовим для іншого похідного класу, конструктори цих трьох класів викликаються в порядку успадкування. Деструктори викликаються у зворотному порядку.

Якщо похідний клас напряму успадковує множину базових класів, використовується таке розширене оголошення:

```
class ім'я_похідного_класу :
            спец_доступу1  ім'я_базового_класу1,
            спец_доступу2  ім'я_базового_класу2,
            ...
            спец_доступуN  ім'я базового_класуN
 {
   ... // тіло класу
  }
```

Тут *ім'я_базового класу1, ... , ім'я_базового класуN* — імена базових класів, *спец_доступу* — специфікатор доступу, який може бути різним для кожного базового класу. Коли успадковується множина базових класів, конструктори використовуються зліва направо у порядку, що задається в оголошенні похідного класу. Деструктори виконуються у зворотному порядку.

Коли клас успадковує множину базових класів, конструкторам яких необхідні аргументи, похідний клас передає ці аргументи, використовуючи розширену форму оголошення конструктора похідного класу:

```
ім’я_похідного_класу ( список_арг ):
            ім’я_базового_класу1( список_арг1 ),
            ім’я_базового_класу2( список_арг2 ),
                        .  .  .
            ім’я_базового_класуN( список_аргN )
{
 ... // тіло конструктора похідного класу
}
```

Коли похідний клас успадковує ієрархію класів, кожний похідний клас повинен передавати попередньому базовому класу по ланцюжку необхідні аргументи.

7. Віртуальні базові класи

У разі багаторазового прямого успадкування похідним класом одного і того самого базового класу може виникнути проблема. Розглянемо такий варіант (рис. 1).

Тут базовий клас *Базовий* успадковується похідними класами *Похідний1* і *Похідний2*. Похідний клас *Похідний3* прямо успадковує ці класи. Це означає, що клас *Базовий* фактично успадковується класом *Похідний3* двічі — через класи *Похідний1* і *Похідний2***.** Однак, якщо член класу *Базовий* буде використовуватись у класі *Похідний3*, це зумовить неоднозначність, оскільки в ньому буде дві копії класу *Базовий***.** Для попередження такої ситуації в C++ включений механізм віртуального базового класу (virtual base class). У цьому випадку перед специфікатором доступу базового класу необхідно поставити ключове слово *virtual*, наприклад:

```
class derived2: virtual public base
```

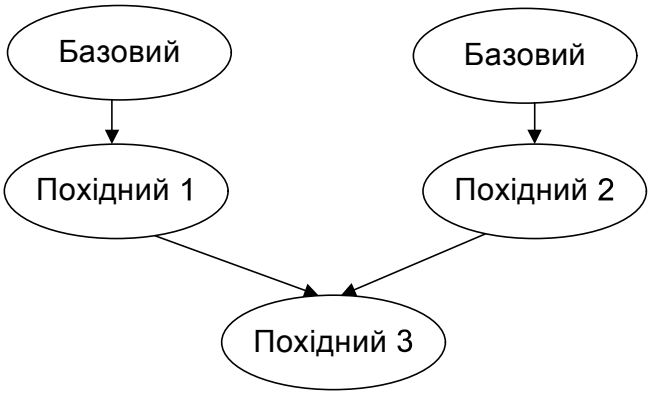

Рис. 1. — Багаторазове успадкування одного класу

Порядок виконання роботи

1. Створити у програмі клас *Rahunok*, який зберігатиме такі дані: унікальний номер рахунка, суму на рахунку (початкове значення задати при створенні рахунка), нарахування відсотків.

2. Додати до класу функції: конструктора з параметрами, відображення рахунка, встановлення додаткових відсотків, перерахування грошей з урахуванням відсотків.

3. Створити у програмі клас *Vkladnyk,* який зберігатиме такі дані: серію та номер паспорту, прізвище, ім'я та по батькові.

4. Додати до класу конструктора з параметрами, функцію відображення даних про вкладника.

5. Створити у програмі похідний клас *RahunokVkladnyka* від попередніх двох класів. Цей клас має містити дані про максимальну суму, яку можна зняти за добу, мінімальний залишок на рахунку після зняття грошей з рахунка, пароль, а також функції: конструктор з параметрами, функцію виведення інформації про рахунок вкладника, функцію зміни пароля, функцію вкладення грошей, функцію зняття грошей. Урахуйте необхідну політику конфіденційності та безпеки даних банку.

6. У головній частині програми продемонструйте роботу створеної ієрархії класів на прикладі такого об'єкта:

```
RahunokVkladnyka Geits(123456789, "Bill Geits",
                       "EC123456", 50000, 500, 500);
```

Контрольні завдання та запитання

1. Поясніть принцип ООП — успадкування. Наведіть приклади.

2. Які ви знаєте специфікатори доступу?

3. Що відбувається з відкритими та закритими членами базового класу, якщо базовий клас успадковується як відкритий похідним?

4. Що відбувається з закритими та відкритими членами базового класу, якщо базовий клас успадковується як закритий похідним?

5. Поясніть, навіщо потрібна категорія захищеності *protected*? Розгляньте два випадки: коли *protected* використовується всередині описання класу та під час успадкування.

6. Якщо один клас успадковується іншим, яким порядок виклику конструкторів та деструкторів? Наведіть приклади.

7. Що таке множинне успадкування? Які ви знаєте можливі варіанти множинного успадкування?