# Лекція 7 Спадкування

Спадкування (узагальнення) – це відношення між більш загальною сутністю, яка називається суперкласом, та її конкретним втіленням, що називається підкласом.

Спадкування – один із трьох фундаментальних принципів об'єктно-орієнтованого програмування, оскільки саме завдяки йому можливе створення ієрархічної класифікації. Використовуючи наслідування, можна створити клас, який визначає найбільш загальні характери всіх пов'язаних із ним елементів. Цей клас потім може бути успадкований іншими, спеціалізованими класами з додаванням до кожного з них своїх, унікальних особливостей.

У стандартній термінології мови C++ клас, що успадковується, називається базовим. Клас, який використовує базовий клас, називається похідним. Похідний клас можна використовувати як базовий для іншого похідного класу. Таким шляхом буде багаторівнева ієрархія класів.

Базовий клас успадковується похідним класом. Для цього базовий клас визначається при оголошенні похідного. Найкраще почати з прикладу (Рис. 1)

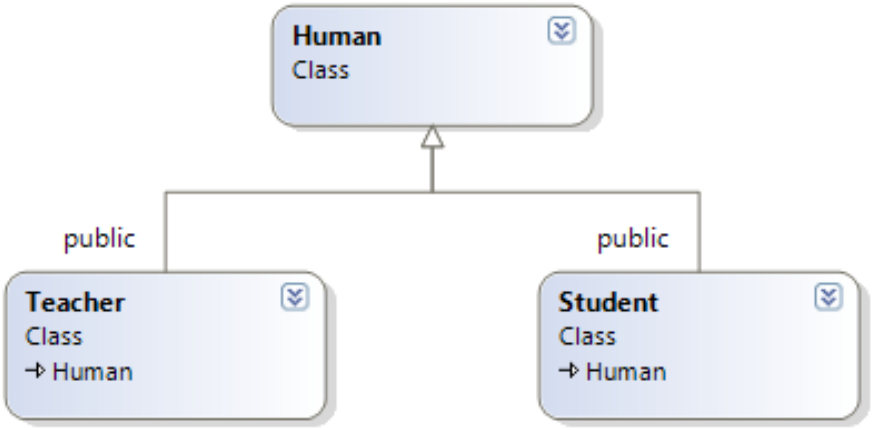



Рис.1. Похідні класи Teacher і Student є нащадками класу Human.

Розглянемо клас Human, який в найзагальніших рисах визначає будь-яку людину. Кожна людина має рік народження та ім’я. Саме ці дані зберігають відповідні поля класу Human.

```
class Human
{
```

```
 int  year_of_birth; // рік народження
 string name; // ім'я
 public:
    void SetYear(int year) { year_of_birth = year; }
    int GetYear () { return year_of_birth; }
    void SetName(string text) { name = text; }
    string GetName () { return name; }

};
```

Це загальне визначення людини можна використовувати для визначення інших осіб. Наприклад, студента і викладача. В наступному фрагменті шляхом спадкування класу Human створюється клас Student (студент).

```
class Student : public Human
{
 int kurs; // курс
 string group; // група
 public:
    void SetKurs(int num) { kurs = num; }
    int GetKurs () { return kurs; }
    void SetGroup(string text) { group = text; }
    string GetGroup() { return group; }
 void Show ();
};
```

Той факт, що клас `Student` успадковує клас `Human`, означає, що клас `Student` успадковує весь вміст класу `Human`. До вмісту класу `Student` клас `Human` додає поле `kurs i group`, а також функції-члени, які визначають ці поля.

Загальний формат для забезпечення спадкування має такий вигляд.

клас ім'я_похідного_класа: доступ ім'я _базового_класу

{

тіло нового класу

}

Тут елемент *доступ* необов'язковий. При необхідності він може бути виражений одним з модифікаторів доступу: *public*, *private* або *protected*. Детальніше про ці модифікатори буде сказано в цьому розділі. А поки в визначеннях всіх успадкованих класів будемо використовувати модифікатор *public*. Це означає, що всі public - члени базового класу також будуть public - членами похідного класу. Отже, в попередньому прикладі члени класу `Student` мають доступ до відкритих функцій-членів класу `Human`, як ніби ці функції були

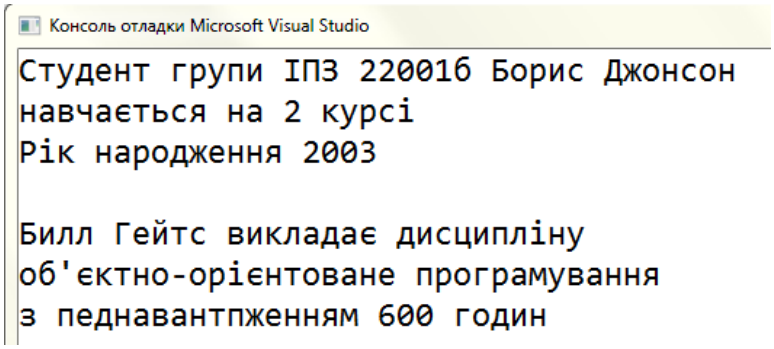
оголошені в тілі класу Human. Однак клас Student не має доступу до private - членів класу Human. Так, для класу Student закритий доступ до члена даних (поля) name і year_of_birth.

Розглянемо програму, яка використовує механізм спадкування для створення двох похідних класів від класу Human: Student і Teacher.

```
// Демонстрація Спадкування.
// Визначаємо базовий клас людина
class Human
{
 int  year_of_birth; // рік народження
 string name; // ім'я
public:
 void SetYear(int year) { year_of_birth = year; }
 int GetYear() { return year_of_birth; }
 void SetName(string text) { name = text; }
 string GetName() { return name; }
};

// Визначаємо клас студент
class Student : public Human
{
 int kurs; // курс
 string group; // група
public:
 void SetKurs(int num) { kurs = num; }
 int GetKurs() { return kurs; }
 void SetGroup(string text) { group = text; }
 string GetGroup() { return group; }
 void Show();
};

// Визначаємо клас викладач
class Teacher : public Human
{
 int hours; // піднавантаження на рік.
 string discipline; // дисципліна
public:
 void SetHours(int num) { hours = num; }
 int GetHours() { return hours; }
 void SetDiscipline(string text) { discipline = text; }
 string SetDiscipline() { return discipline; }
 void Show();
};

void Student::Show()
{
 cout << "Студент групи " << group << " " << GetName() << endl;
 cout << "навчається на " << kurs << " курсі"<<endl;
 cout << "Рік народження " << GetYear() << endl << endl;
```

```
}
void Teacher::Show()
{
cout << GetName() << " викладає дисципліну \n" << discipline <<
endl;
cout << "з педнавантпженням " << hours << " годин" << endl << endl;
}
int main()
{
 setlocale(0, "Ukr");
 Student s;
 Teacher t;
 s.SetName("Борис Джонсон");
 s.SetYear(2003);
 s.SetGroup("ІПЗ 220016");
 s.SetKurs(2);
 t.SetName("Билл Гейтс");
 t.SetYear(1955);
 t.SetHours(600);
 t.SetDiscipline("об'єктно-орієнтоване програмування");
 s.Show();
 t.Show();
}
```

При виконанні ця програма генерує такі результати:




Як видно за результатами виконання цієї програми, основна перевага спадкування полягає в тому, що воно дозволяє створити базовий клас, який потім можна включити до складу більш спеціалізованих класів.

### Управління доступом до членів базового класу

Якщо один клас успадковує інший, члени базового класу стають членами похідного. Статус доступу членів базового класу в похідному класі визначається модифікатором доступу, що використовуються для спадкування базового класу. Модифікатор доступу базового класу виражається одним з ключових слів: *public*, *private* або *protected*. Якщо модифікатор доступу не вказано, то за замовчуванням використовується модифікатор *private*, якщо мова

йде про спадкування типу *class*. Якщо успадковується тип *struct*, то при відсутності явно заданого модифікатора доступу за замовчуванням використовується модифікатор *public*.

Якщо базовий клас успадковується як *public*-клас, його *public*-члени стають *public*-членами похідного класу.

У всіх випадках *private*-члени базового класу залишаються закритими в рамках цього класу і не доступні для членів похідного.

Наприклад, в наступній програмі public-члени класу base стають public-членами класу derived. Отже, вони будуть доступні і для інших частин програми.

```
class Base {
int i, j;
public:
void Set (int a, int b) {i = a; j = b; }
void Show () {cout << i << " " << j << " "; }
};

class Derived: public Base {
int k;
public:
Derived (int x) {k = x; }
void Showk () {cout << k << " "; }
};

int main ()
{
Derived ob (3);
ob.Set (1, 2); // доступ до членів класу base
ob.Show (); // доступ до членів класу base
ob.Showk (); // доступ до члена класу derived
}
```

Оскільки функції Set() і Show() (члени класу Base) успадковані класом Derived як public-члени, їх можна викликати для об'єкта типу Derived в функції main(). Оскільки члени даних i та j визначені як private-члени, вони залишаються закритими в рамках свого класу Base.

Якщо базовий клас успадковується як private-клас, його public-члени стають private-членами похідного класу.

**Використання захищених членів**

Член класу може бути оголошений не тільки відкритим (public) або закритим (private), але і захищеним (protected). Крім того, базовий клас в цілому може бути успадкований з використанням модифікатора protected. Ключове слово protected додано в C ++ для надання механізму спадкування більшої гнучкості.

Закриті члени базового класу не доступні ніяким іншим частинам програми, включаючи і похідні класи. Однак з захищеними членами все інакше. Якщо базовий клас успадковується як public-клас, захищені члени базового класу стають захищеними членами похідного класу, тобто доступними для похідного класу. Отже, використовуючи модифікатор protected, можна створити члени класу, які закриті в рамках свого класу, але які може успадкувати похідний клас, причому з отриманням доступу до них.

Розглянемо наступний приклад програми.

```
class Base {
protected:
 int i, j; // Ці члени закриті в класі base, але доступні
              //для класу derived.
public:
 void Set(int a, int b) { i = a; j = b; }
 void Show() { cout << i << " " << j << " "; }
};

class Derived : public Base {
 int k;
public:
 // Клас derived має доступ до членів класу base i і j.
 void SetK() { k = i * j; }
 void ShowK() { cout << k << ""; }
};

int main()
{
 Derived ob;
 ob.Set(2, 3); // OK, класу Derived це дозволено.
 ob.Show(); // OK, класу Derived це дозволено.
 ob.SetK();
 ob.ShowK();
}
```

При оголошенні члена класу відкритим (з використанням ключового слова public) до нього можна отримати доступ з будь-якої іншої частини програми. Якщо член класу оголошується закритим (за допомогою модифікатора private), до нього можуть отримувати доступ тільки члени того ж класу. Більш того, до закритих членам базового класу не мають доступу навіть похідні класи. Якщо ж член класу оголошується захищеним (protected-членом), до нього можуть отримувати доступ тільки члени того ж або похідних класів. Таким чином, модифікатор protected дозволяє успадковувати члени, але залишає їх закритими в рамках ієрархії класів.

Якщо базовий клас успадковується з використанням ключового слова public, його public-члени стають public-членами похідного класу, а його protected-члени – protected-членами похідного класу.

Якщо базовий клас успадковується з використанням модифікатора protected, його public і protected-члени стають protected-членами похідного класу.

Якщо базовий клас успадковується з використанням ключового слова private, його public- і protected-члени стають private-членами похідного класу.

Во всіх випадках private-члени базового класу залишаються закритими в рамках цього класу і не успадковуються.

### 5.4 Множинне спадкування

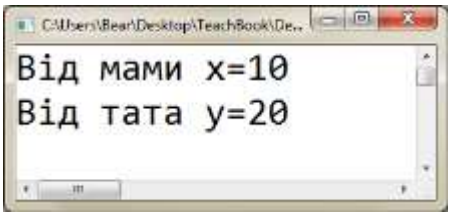
Похідний клас може успадковувати два або більше базових класів (так зване множинне спадкування). Наприклад, у студента є двоє батьків – мамо і тато. Як відомо, студент має різні властивості від обох батьків. Промоделюємо цю ситуацію за допомогою множинного спадкування.

В цій короткій програмі клас Student успадковує обидва класу Mamo і Tato.

```
// Приклад використання декількох базових класів.
class Mamo {
protected:
int x;
public:
void ShowX () {cout <<"Від мами x="<< x <<endl; }
};
```

```
class Tato {
protected:
int y;
public:
void ShowY () {cout <<"Від тата y="<< y <<endl; }
};
// Спадкування двох базових класів.
class Student: public Mamo, public Tato {
public:
void Set(int i, int j) {x = i; y = j; }
};

int main ()
{
setlocale(0, "Ukr");
Student ob;
ob.Set(10, 20); // член класу derived
ob.ShowX(); // функція з класу Мама
ob.ShowY(); // функція з класу Papa
}
```

При виконанні ця програма генерує такі результати:




На рисунку 2 показана ієрархічна схема цих класів.

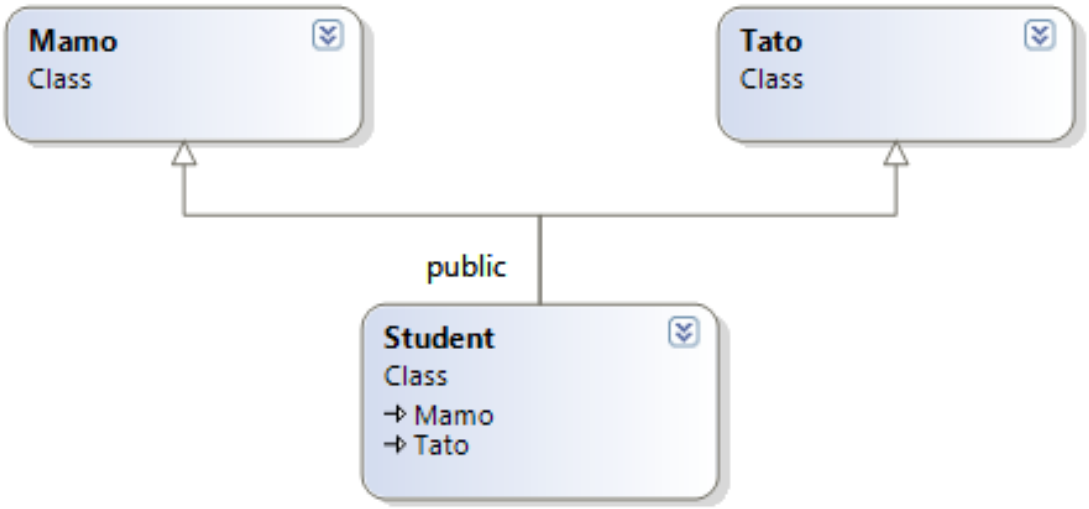



Рис. 2. Клас Student успадковує обидва класа Mamo і Tato.

Як видно з цього прикладу, щоб забезпечити спадкування декількох базових класів, необхідно через кому перерахувати їх імена у вигляді списку.

При цьому потрібно вказати модифікатор доступу для кожного успадкованого базового класу.

У відомого польського письменника-фантаста Станіслава Лема є оповідання, де герой потрапляє на планету, на якій мешкали п'ятистатеві істоти. У створенні потомства брали участь папа, дада, гага, фафа і хаха. Спробуйте самі змоделювати цю ситуацію, використовуючи механізм множинного спадкування.

### 5.5 Віртуальні базові класи

При наслідуванні декількох базових класів в програму може бути внесений елемент невизначеності. У попередньому прикладі студент є спадкоємцем двох класів – мамо і тато, але кожен із цих класів, у свою чергу, є спадкоємцем класу людина (Human). Така схема успадкування зображена на діаграмі (рис. 3)

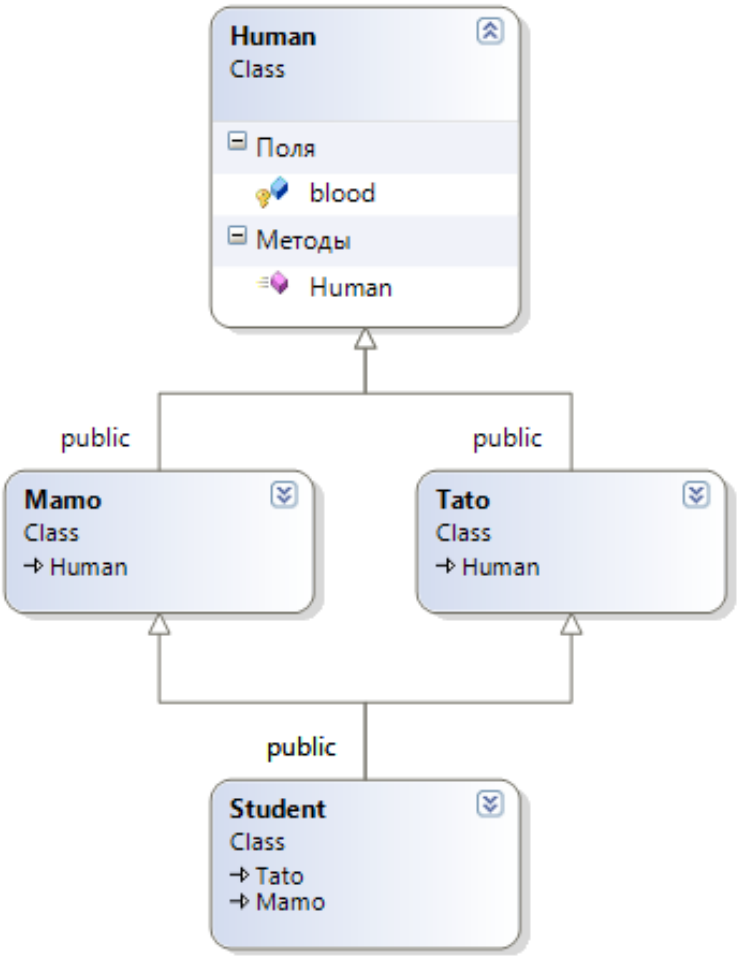



Рис. 3. Схема успадкування на діаграмі класів.

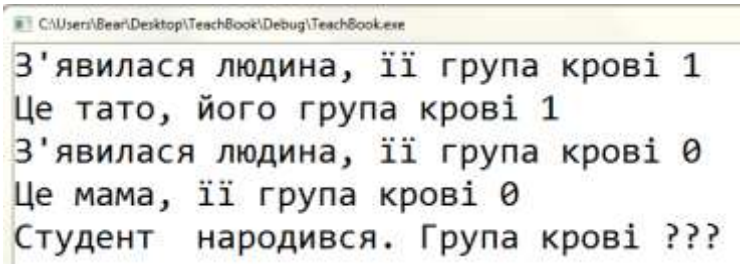
Розглянемо наступний приклад:

Кожна людина має певну групу крові, це змінна blood в базовому класі Human. Клас Mamo і клас Tato успадковують різні крупи крові, бо вони – різні людини і в кожної з них тече своя кров. Але яку групу крові отримає від своїх батьків наш студент? Поки на це питання немає відповіді і це показано в наступної програмі:

```
#include <time.h>
using namespace std;
// Приклад використання декількох базових класів.

class Human
{
protected:
 int blood;
public:
 Human()
 {
      blood = rand() % 4;
      cout << "З'явилася людина, її група крові " <<blood<< endl;
 }
};
class Tato : public  Human //
{
public:
 Tato()
 {
      cout << "Це тато, його група крові "<<blood << endl;
 }
};
class Mamo : public  Human //
{
public:
 Mamo()
 {
      cout << "Це мама, її група крові " << blood << endl;
 }
};
class Student : public Tato, public Mamo
{
public:
 Student()
 {
      cout << "Студент  народився. Група крові ???" << endl;
 }
};
int main()
{
 setlocale(0, "");
```

```
 srand((unsigned)time(0));
    Student vasya;
}
```

Результат роботи програми:




Однак, коли ми спробуємо змінити код, щоб вивести групу крові студента, отримаємо таку помилку:

```
Student()
{
    cout << "Студент  народився. Група крові "<<blood << endl;
}
```

error C2385: неоднозначный уровень доступа "blood"
IntelliSense: "Student::blood" не является однозначным

Обидва класи Mamo і Tato наслідують клас Human. Але клас Student успадковує як клас Mamo, так і клас Tato. В результаті в об'єкті типу Student присутні дві копії класу Human, в такому виразі

```
cout << "Студент  народився. Група крові "<<blood << endl;
```

не ясно, на яку саме копію змінної blood тут дано посилання: на змінну blood, успадковану від класу Mamo або від класу Tato? Оскільки в класі Student присутні обидві копії класу Human, то в ньому існують і два члена blood!

Тому-то ця інструкція і є невизначеною. Є два способи виправити попередню програму. Перший полягає в застосуванні оператора дозволу контексту (дозволу області видимості), за допомогою якого можна «вручну» вказати потрібну змінну blood:

```
Student()
{
cout << "Студент  народився. Група крові "<<Mamo::blood << endl;
}
```

Наразі невизначеність ліквідована і студент завжди успадковує свою групу крові від матері.

Застосування оператора "::" дозволяє програмі «ручним способом» вибрати версію класу Human (успадковану класами Mamo і Tato). Але після такого рішення виникає цікаве питання: а що, якщо в дійсності потрібна тільки одна копія класу Human? Чи можна якимось чином запобігти включення двох копій в клас Student? Відповідь, як, напевно, ви здогадалися, позитивна. Це рішення досягається за допомогою віртуальних базових класів.

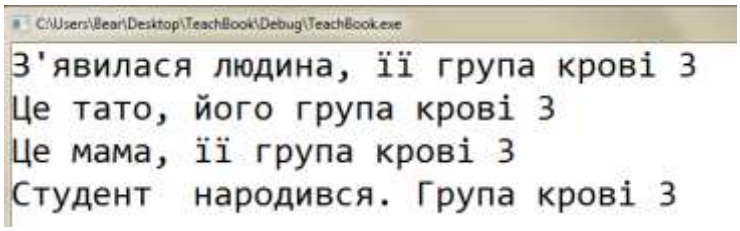
Якщо два (або більше) класа є нащадками загального базового класу, ми можемо запобігти включення декількох його копій в об'єкти цих похідних класів, що реалізується шляхом оголошення базового класу при його спадкуванні віртуальним. Для цього достатньо випередити ім'я успадкованого базового класу ключовим словом virtual.

Для ілюстрації цього процесу наведемо ще одну версію попередньої програми. На цей раз клас Student містить тільки одну копію класу Human.

```
// Ця програма використовує віртуальні базові класи.
class Human
{
protected:
 int blood;
public:
 Human()
 {
      blood = rand() % 4;
      cout << "З'явилася людина, її група крові " <<blood<< endl;
 }
};
class Tato : virtual public  Human //
{
public:
 Tato()
 {
      cout << "Це тато, його група крові "<<blood << endl;
 }
};
class Mamo : virtual public  Human //
{
public:
 Mamo()
 {
      cout << "Це мама, її група крові " << blood << endl;
 }
};
class Student : public Tato, public Mamo
{
public:
```

```
 Student()
 {
      cout << "Студент  народився. Група крові "<<blood << endl;
 }
};


int main()
{
 setlocale(0, "");
 Student vasya;
}
```

Результат роботи програми:




Тепер обидва класи Мамо і Тато успадковують клас Human як віртуальний, і тому при будь-якому множинному наслідуванні в похідний клас буде включена лише одна його копія. Отже, в класі Student присутня лише одна копія класу Human, а інструкція

```
cout << "Студент  народився. Група крові "<<blood << endl;
```

тепер абсолютно допустима і не містить ніякої неоднозначності. Але зверніть увагу, що тепер і мама і тато – це не дві різних людини, а всього лише одна. Що ж, таке трапляється після довгих років спільного життя.

Різниця між звичайним базовим і віртуальним класами стає очевидною тільки тоді, коли цей базовий клас успадковується більше одного разу. (Рис. 3.) Якщо базовий клас оголошується віртуальним, то тільки один його екземпляр буде включений в об'єкт похідного класу. В іншому випадку в цьому об'єкті буде присутній кілька його копій.