# Поліморфізм та віртуальні функції.

Похідні класи мають з базовим класом зв'язки двох видів. **Перший** з них полягає в тому, що екземпляри похідних класів використовують всі відкриті члени базового класу – зокрема методи базового класу.

```
class Base
{ public:
      void meth ()
      {
        cout << "In Base: meth() " << endl;
      }
};
class SubBase : public Base
{ // …
};
int main (void)
{
 Base    b;              // екземпляр базового класу
 SubBase sb;             // екземпляр похідного класу
 b.meth ();              // виклик методу базового класу
 sb.meth ();             // виклик методу базового класу
 return 0;
}
```

**Другий** вид зв'язку полягає в тому, що:

- екземпляр базового класу можна створити як екземпляр похідного;
- посилання на базовий клас може посилатись на похідний;
- вказівник на базовий клас може вказувати на похідний.

Всі ці операції виконуються без явного приведення типів і є реалізацією відношення «is-a».

```
int main (void)
{
 Base    b;
 SubBase sb;
// екземпляр базового класу створюється як похідний
 Base bb = SubBase ();
// екземпляру базового класу присвоюється похідний
 b = sb;
// посилання на базовий клас посилається на похідний
 Base & bbb = sb;
// вказівник на базовий клас вказує на похідний
 Base *p = &sb;
// sb = b;  // таке присвоєння неможливе!
 return 0;
}
```

Цілком зрозуміла заборона присвоєнь у зворотному напрямку – адже якщо екземпляр похідного класу створюється як базовий, то виникає проблема із викликом методів похідного класу, яких немає у базовому:

```
class Base
{ public:
  void meth ()
  { cout << "In Base: meth()" << endl; }
};
class SubBase : public Base
{
  void meth_SubBase ()
  { cout << "Власний метод похідного класу" << endl;}
};
int main (void)
{
 Base    b;              // екземпляр базового класу
 SubBase sb = b;         // припустимо це можливим
 b.meth ();              // виклик методу базового класу
 sb.meth_SubBase (); // в базовому класі метод відсутній
 return 0;
}
```

Та обставина, що посилання та вказівники базового класу можуть вказувати на екземпляри похідних класів, приводить до низки цікавих можливостей – зокрема методи, які мають параметрами посилання або вказівник на базовий клас, можуть викликатись із аргументами-екземплярами похідних класів:

```
void fun (Base & b)
{
  b.meth();
}
int main (void)
{
 Base    b;
 SubBase sb;
 fun (b);     // так можливо
 fun (sb);    // і так теж можливо
 return 0;
}
```

Але в будь-якому разі, функція **fun()** викликатиме метод **meth()** базового класу.

Проте, можлива ситуація, коли успадковані методи похідних класів повинні поводити себе інакше, ніж методи базового класу. Така поведінка називається “поліморфною”. (Поліморфний – такий, що має багато форм). Реалізація поліморфного спадкування здійснюється одним із двох способів.

1. ***Перекриття методів базового класу у похідному класі*** :

```
class Base
{ public:
  void meth ()
  { cout << "In Base: meth() " << endl; }
};
class SubBase : public Base
{
// цей метод перекриває відповідний метод базового класу
  void meth ()
  { cout << "In SubBase: meth() " << endl;}
};
int main (void)  {
 Base    b;            // екземпляр базового класу
 SubBase sb;           // екземпляр похідного класу
 b.meth ();            // виклик методу базового класу
 sb.meth ();           // виклик методу похідного класу
 return 0;
}
```

В усіх попередніх прикладах зв'язування екземпляру із конкретним методом (функцією-членом класу) відбувалось на етапі компіляції (тобто ще до початку її виконання). Ця процедура, як відомо, називається **раннім зв'язуванням**. Альтернативний спосіб – **пізнє зв'язування** (інколи – динамічне зв'язування, в C# - динамічний поліморфізм) дозволяє асоціювати об'єкт із методом саме **під час виконання програми**.

2. ***Використання віртуальних методів*** . Пізнє зв'язування стосується функцій-членів (методів), які називаються **віртуальними функціями**. Віртуальна функція (`virtual`) оголошується в базовому класі і може бути перевизначена у похідних класах. Сукупність класів, в яких визначається і перевизначається віртуальна функція, називається **поліморфним кластером**. У межах цього кластеру об'єкт пов'язується із конкретною віртуальною функцією-членом під час виконання програми. Звичайна функція-член також може бути перевизначена у похідному класі, як у попередньому прикладі. Проте без атрибуту `virtual` до неї буде застосоване лише раннє зв'язування.

# Приклад.

```
class Base
{
public:
   virtual void virt () // віртуальний метод
   {
   cout << "In class Base" << endl;
   }
};
class SubBase : public Base
{
public :
//віртуальний метод заміщається у похідному класі
//слово virtual у похідному класі – не обов'язкове
   virtual void virt ()
   {
   cout << "In class SubBase" << endl;
   }
};
```

Тепер, якщо визначити зовнішню функцію **`fun (Base & b)`**, як у попередньому прикладі, то ми побачимо реалізацію пізнього зв'язування:

```
void fun (Base & b)
{
  b.virt();
}
int main (void)
{
 Base    b;
 SubBase sb;
 fun (b); // виклик методу virt()базового класу
 fun (sb);// виклик методу virt()похідного класу
 return 0;
}
```

Рішення про те, який саме метод **`virt()`** базового чи похідного класу має бути викликаний, приймається під час виконання програми – це пізнє звязування.

Віртуальні методи можуть перевантажуватись, як звичайні функції:

```
class Base
{
public:
 virtual void virt ()     // віртуальний метод
   {  cout << "In class Base" << endl;  }
 virtual void virt (int i)// ще один вірт. метод
   {  cout << "In class Base " << i << endl;  }
};
class SubBase : public Base
{
public :
  virtual void virt ()
   {  cout << "In class SubBase" << endl;  }
  virtual void virt (int i)
   {  cout << "In class SubBase" << i << endl; }
};
```

Слід зазначити, що використання пізнього зв'язування достатньо складний механізм, який вимагає суттєвих витрат пам'яті. Тому віртуальними слід робити лише такі функції, які дійсно будуть перевизначатись у похідних класах.

Зауваження. Конструктори не можуть бути віртуальними – адже похідний клас не спадкує конструктор базового. А от деструктор може бути віртуальним. Користь віртуального деструктора показує наступний приклад, висновком з якого може бути основне правило:

**–Якщо клас містить хоч одну віртуальну функцію, деструктор класу теж слід зробити віртуальним.**

Взагалі кажучи, якщо клас передбачає спадкування, його деструктор завбачливо мав би бути визначений віртуальним. Проте йти на такі додаткові витрати варто саме тоді, коли в класі є принаймні одна віртуальна функція. Проаналізуємо наступний приклад.

## Приклад.

```
class Base
{
public:              // раніше визначені члени класу
 ~Base ()            // звичайний деструктор
 {cout << "Destructor Base" << endl;}
};
class SubBase : public Base
{
public :             // раніше визначені члени класу
 ~SubBase ()         // звичайний деструктор
 {cout << "Destructor SubBase" << endl;}
};
int main (void)
{
 Base *p = new SubBase ();
 delete p;
 return 0;
}
```

При виконанні цього прикладу можна побачити, що екземпляр класу, що адресується вказівником **`p`**, створюється цілком коректно – спочатку працює конструктор базового класу, а потім конструктор похідного. Таким чином, створений через вказівник екземпляр похідного класу **`SubBase`** потім приводиться до типу вказівника на базовий клас **`Base`**.

Але – при знищенні цього екземпляру операцією **`delete`** спрацьовує лише деструктор базового класу! Це означає втрати пам'яті (**`memory leaks`**), адже не була коректно знищена частина екземпляру, яка відповідає похідному класу. Причина в тому, що об' єкт знищується через вказівник на базовий клас, а базовому класу нічого невідомо про похідний, адже працює раннє зв'язування. Щоб позбутись цього ефекту, конструктор в базовому класі слід визначити як віртуальний. Тоді і знищення екземпляру, адресованого вказівником відбудеться коректно:

## Приклад.

```
class Base
{
public:              // раніше визначені члени класу
 virtual  ~Base ()     // віртуальний деструктор
 {cout << "Destructor Base" << endl;}
};
class SubBase : public Base
{
public :             // раніше визначені члени класу
 virtual ~SubBase () // віртуальний деструктор
 {cout << "Destructor SubBase" << endl;}
};
int main (void)
{
 Base *p = new SubBase ();
 delete p;
 return 0;
}
```

Повернемось ще раз до перевизначення функцій.

Якщо в похідному класі визначається метод, одноіменний з віртуальним методом базового класу, але з відмінною сигнатурою, він перекриває віртуальний метод базового класу. Це означає, що в похідних класах віртуальний метод базового класу не доступний.

```
class Base
{ public:                  // раніше визначені члени класу
  virtual  virt ();        // віртуальний метод
};
class SubBase : public Base
{ public :                 // раніше визначені члени класу
  virt (int i);            // метод - перекриває віртуальний
};
int main (void)
{
 Base b = Base ();
 SubBase sb = SubBase ();
 sb.virt (10); // припустимо
 sb.virt (); //помилка – метод баз. класу недоступний
 return 0;
}
```

Отже, віртуальна функція – це функція-член класу, помічена словом **`virtual`**, для якої можливе перевизначення у всіх або деяких похідних класах. При звертанні до екземпляру похідного класу ***через вказівник або посилання*** на базовий клас буде виконана саме перевизначена (заміщена) у похідному класі віртуальна функція.

# Абстрактний базовий клас (ABC – Abstract Base Class).

Наразі нам відомі правила простого спадкування та більш складного поліморфного спадкування, яке включає використання віртуальних функцій. Наступний рівень складності – абстрактний базовий клас. Необхідність в ньому виникає, коли необхідно описати об'єкти, що мають східну природу, проте їх важко визначити як базовий та похідний класи. Наприклад, розглядаючи такі об'єкти, як прямокутник та ромб, неможливо встановити між ними відношення «Є» (“is-a”), хоча й очевидно, що вони мають багато спільного: наприклад, поняття площі, повороту на площині. У таких випадках необхідно виділити у об'єктів все спільне і створити клас, який буде базовим для них всіх. Якщо реалізація окремих функцій можлива лише на рівні похідних класів, у базовому їх визначають як **чисто віртуальні функції**. Екземпляри такого базового класу неможливо створити, сам клас називається **абстрактним** і використовується лише для створення похідних класів.

## Приклад.

```
class Figure // клас абстрактний – він має чисто віртуальну функцію
{ protected :
    double x_cnt, y_cnt; // координати центру фігури
  public:
  Figure (double x=0, double y=0) : x_cnt (x), y_cnt (y) {}
// чисто віртуальна функція:
  virtual double Square () const = 0;
};
class Rectangle : public Figure // похідний клас прямокутник
{ private :
  double leng, width;
  public :
  Rectangle (double l=0,double w=0,double x=0,double y=0);
  double Square () const    {  return leng*width;  }
};
class Rhombus : public Figure // похідний клас ромб
{ private :
  double len, angle;
  public :
  Rhombus (double l = 0,double a = 0,double x = 0,double y =
   0);
  double Square () const  { return len*len*sin(angle);  }
};
```