

Об'єктно-орієнтоване програмування

на основі мови C++
2й семестр

- Статичні елементи класу
- Спадкування (Наслідування)

статичні елементи класу.

В C ++ є можливість доступу всіх створених об'єктів конкретного класу до однієї змінної (поля), вміст якої зберігається в одному місці. Для цього оголошують змінну:

static тип ім'я;

Такий клас пам'яті (статичний) може використовуватися не тільки для оголошення статичних полів (змінних класу), але і для методів класу. Пам'ять для цього резервується під час запуску програми до явного створення об'єкта. Тому він є ніби єдиним для всіх копій полів класу. Доступ для такої змінної (: :) можливий тільки після ініціалізації:

тип ім'я_класу :: ім'я_змінної = нач.значеніє;

Не можна формувати статичні члени класу всередині класу, в тілі методів класу. Їх ініціалізація можлива в області видимості файлу. В цьому відношенні статичні члени даних схожі на глобальні змінні.

Приклад 9. Статичні компоненти класу.

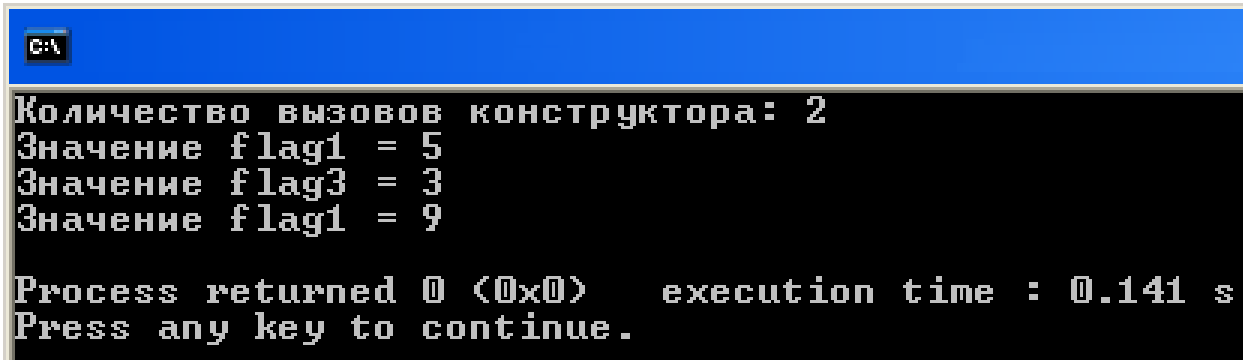
```
#include <iostream>
using namespace std;
class Stat
{
private:
    static int flag2;
    static const int flag3;
public :
    static int flag1;
    Stat () { flag1 ++; }
    int Ret_flag1() { return flag1; }
    static int Ret_flag2() { return flag2; }
    static int Ret_flag3() { return flag3; }
};

int Stat::flag1=0;
int Stat::flag2;
const int Stat::flag3=3;
```

```

int main ()
{
    Stat S1, S2;
    cout<<"Кількість викликів конструктора: "<<Stat::flag1<<endl;
    S1.flag1=4;
    Stat S3;
    cout<<"Значення flag1 = "<<S3.Ret_flag1()<<endl;
    cout<<"Значення flag3 = "<<S1.Ret_flag3()<<endl;
    S2.flag1=9;
    cout<<"Значення flag1 = "<<S3.Ret_flag1()<<endl;
    return 0;
}

```



The screenshot shows a console window with a blue title bar. The output of the program is displayed in white text on a black background. It shows the number of constructor calls as 2, followed by the values of flag1 (5) and flag3 (3) from object S3, and then the value of flag1 (9) from object S1. At the end, it shows the process returned 0 and the execution time was 0.141 seconds.

```

C:\
Кількість викликів конструктора: 2
Значення flag1 = 5
Значення flag3 = 3
Значення flag1 = 9

Process returned 0 (0x0)   execution time : 0.141 s
Press any key to continue.

```

Статичним полям можна задати початкові значення один (і тільки один) раз в області дії файлу. Доступ до відкритих статичним елементів класу можливий через будь-який об'єкт класу або за допомогою імені класу за допомогою бінарної операції дозволу області дії.

- Статичні елементи класу існують навіть тоді, коли не існує ніяких об'єктів цього класу.
- Для забезпечення доступу до закритого або захищеному елементу класу повинен бути передбачений відкритий статичний метод, який повинен викликатися з додаванням перед його ім'ям імені класу і бінарної операції дозволу області дії.
- Метод класу може бути оголошений як `static`, якщо він не повинен мати доступ до нестатичних елементам класу.
- Статичний метод не має покажчика `this`, тому що статичні поля і статичні методи існують незалежно від будь-яких об'єктів класу.
- Статичні поля класу створюються в єдиному екземплярі незалежно від кількості визначених в програмі об'єктів. Всі об'єкти (навіть створені динамічно) поділяють єдину копію статичних полів.

підрахунок об'єктів класу.

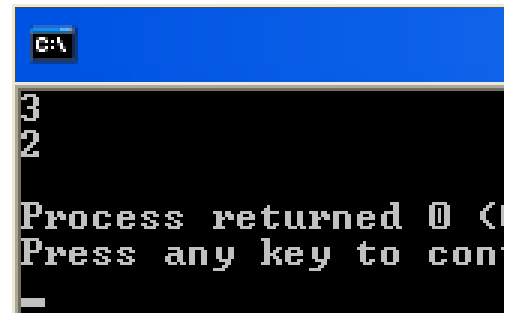
Класичне застосування статичних полів - підрахунок об'єктів. Для цього в класі оголошується статичне поле цілого типу, яке збільшується в конструкторі, а зменшується в деструкції.

Приклад 11. Підрахунок об'єктів класу.

```
#include <iostream>
using namespace std;
class Object
{
    static unsigned int count;
public:
    Object();
    ~Object();
    static unsigned int Count();
};
```

```
unsigned int Object::count = 0;
Object::Object() { ++count; }
Object::~~Object() { --count; }
unsigned int Object::Count()
    { return count; }
```

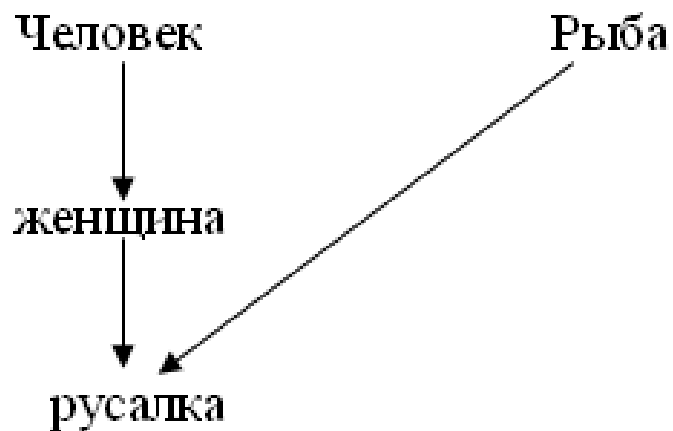
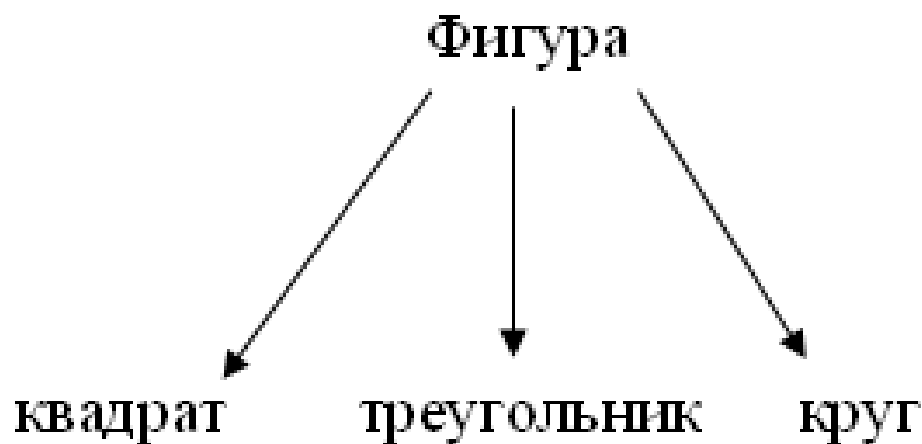
```
int main()
{
    Object a, b, c;
    int i;
    cout<<a.Count()<<endl;
    a.~Object();
    cout<<b.Count()<<endl;
    return 0;
}
```



```
cmd
3
2
Process returned 0 (0x0)
Press any key to continue
_
```

поняття успадкування.

спадкування - це спосіб повторного використання програмного забезпечення, при якому нові класи створюються з уже існуючих класів шляхом запозичення їх атрибутів і функцій і збагачення цими можливостями нових класів. Повторне використання кодів економить час при розробці програм.



Кожен об'єкт похідного класу є також об'єктом відповідного базового класу. Однак, зворотне невірно: об'єкт базового класу не є об'єктом класів, породжених цим базовим класом.

Види спадкування: просте і множинне.

***просте спадкування** - кожен клас має тільки один батьківський клас найближчого рівня.*

***множинне спадкування** - клас-нащадок створюється з використанням декількох базових класів-батьків.*

спадкування

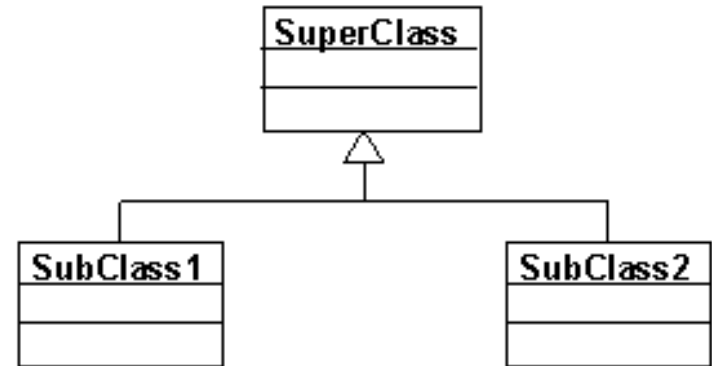
Спадкування є найпотужнішим інструментом ООП і застосовується для наступних взаїмозв'язаних цілей:

- виключення з програми повторюваних фрагментів коду;
- спрощення модифікації програми;
- спрощення створення нових програм на основі існуючих.
- Крім того, спадкування є єдиною можливістю використовувати об'єкти, вихідний код яких недоступний, але в які потрібно внести зміни.

синтаксис успадкування

ключі доступу

```
class ім'я: [Private | protected | public] базовий_клас  
{Тіло класу};
```



```
class A {...};
```

```
class B {...};
```

```
class C {...};
```

```
class D: A, Protected B, public C  
{...};
```

- У спадкоємця можна **описувати нові** поля і методи і **перевизначати** існуючі методи. Перевизначати методи можна декількома способами.
- Якщо який-небудь метод в нащадку повинен працювати абсолютно по-іншому, ніж в предка, метод описується в нащадку заново. При цьому він може мати інший набір аргументів.
- Якщо потрібно внести додавання в метод предка, то у відповідному методі нащадка поряд з описом додаткових дій виконується виклик методу предка за допомогою операції доступу до області видимості.
- Якщо в програмі планується працювати одночасно з різними типами об'єктів ієрархії або планується додавання в ієрархію нових об'єктів, метод оголошується як віртуальний за допомогою ключового слова `virtual`. Всі віртуальні методи ієрархії з одним і тим же ім'ям повинні мати однаковий список аргументів.

Правила спадкування

ключ доступу	Специфікатор в базовому класі	Доступ в похідному класі
private	private protected public	немає private private
protected	private protected public	немає protected protected
public	private protected public	немає protected public

Іншими словами:

- private елементи базового класу в похідному класі недоступні незалежно від ключа. Звернення до них може здійснюватися лише через методи базового класу.
- елементи `protected` при спадкуванні з ключем `private` стають в похідному класі `private`, В інших випадках права доступу до них не змінюються.
- Доступ до елементів `public` при спадкуванні стає відповідним ключу доступу.

Якщо базовий клас успадковується з ключем `private`, можна вибірково зробити деякі його елементи доступними в похідному класі:

```
class Base {  
    ...  
    public: void f ();  
};  
class Derived: private Base {  
    ...  
    public: Base :: void f ();  
};
```

просте спадкування

```
class daemon: public monstr{
    int brain;
public:
    // ----- конструктори:
    daemon (int br = 10) {brain = br;};
    daemon (color sk) : monstr (sk) {Brain = 10;}
    daemon (char * nam) : monstr (nam) {brain = 10;}
    daemon (daemon & M) :monstr (M) {brain =M.brain;}
```

Якщо конструктор базового класу вимагає вказівки параметрів, він повинен бути явним чином викликаний в конструкторі похідного класу в списку ініціалізації

Порядок виклику конструкторів

конструктори не успадковуються, Тому похідний клас повинен мати власні конструктори. Порядок виклику конструкторів:

- Якщо в конструкторі нащадка явний виклик конструктора предка відсутня, *автоматично викликається конструктор предка за замовчуванням.*
- Для ієрархії, що складається з декількох рівнів, конструктори предків викликаються починаючи з самого верхнього рівня. Після цього виконуються конструктори тих елементів класу, які є об'єктами, в порядку їх оголошення в класі, а потім виконується конструктор класу.
- У разі декількох предків їх конструктори викликаються в порядку оголошення.

операція присвоювання

```
const daemon & operator = (daemon & M) {  
    if (& M == this) return * this;  
    brain = M.brain;  
    monstr:: operator = (M);  
    return *this; }
```

Поля, успадковані з класу monstr, Недоступні функцій похідного класу, оскільки вони визначені в базовому класі як private.

Похідний клас може не тільки доповнювати, але і коректувати поведінку базового класу. Перевизначати в похідному класі рекомендується тільки віртуальні методи

спадкування деструкторів

Деструктори не успадковуються. Якщо деструктор в похідному класі не описаний, він формується **автоматично** і викликає деструктори всіх базових класів.

У деструкції похідного класу не потрібно явно викликати деструктори базових класів, це буде зроблено автоматично.

Для ієрархії, що складається з декількох рівнів, деструктори викликаються в порядку, строго зворотному виклику конструкторів: спочатку викликається деструктор класу, потім - деструктори елементів класу, а потім деструкція базового класу.

1. Поняття віртуальної функції.

Віртуальна функція (virtual function) являє собою функцію базового класу, яка переопределяється в похідному класі.

virtual тип імя_функції (параметри);

Приклад 19. Віртуальні функції.

```
#include <iostream>
using namespace std;

class base
{
    public: void print ()
    { cout<<"This is base class\n";}
};

class derived: public base
{
    public : void print ( )
    { cout<<"This is derived class\n";}
};
```

```
int main ()
{
    base B, *bp=&B;
    derived D, *dp=&D;
    base *p=&D;
    bp -> print();
    dp -> print();
    p -> print();
    return 0;
}
```

C:\

```
This is base class  
This is derived class  
This is base class  
  
Process returned 0 (0x0)   execution time : 0.094 s  
Press any key to continue.
```

Якщо не використовувати віртуальні функції, то при створенні об'єкта похідного класу і виклику для нього будь-якої перевизначення функції відбудеться виклик функції саме породженого класу, а не базового. Якщо для виклику перевизначення функції використовувати покажчик або посилання на об'єкт базового класу, то відбудеться виклик функції саме базового класу.

Вибір потрібної функції виконується на етапі компіляції програми і визначається типом покажчика, а не його значенням.

Перевизначена функція викликається відповідно до класу показника. церагнє або статичне зв'язування.

Найбільшу гнучкість дає метод пізнього або динамічного зв'язування (За допомогою віртуальних функцій), що реалізується на етапі виконання програми.

При оголошенні віртуальної функції в базовому класі перед її ім'ям вказується ключове слово **virtual**. У похідному класі віртуальна функція переопределяється. Кожне таке перевизначення (overriding) віртуальної функції в похідному класі означає створення конкретного методу. При перевизначенні віртуальної функції в похідному класі ключове слово *virtual* не вказується.

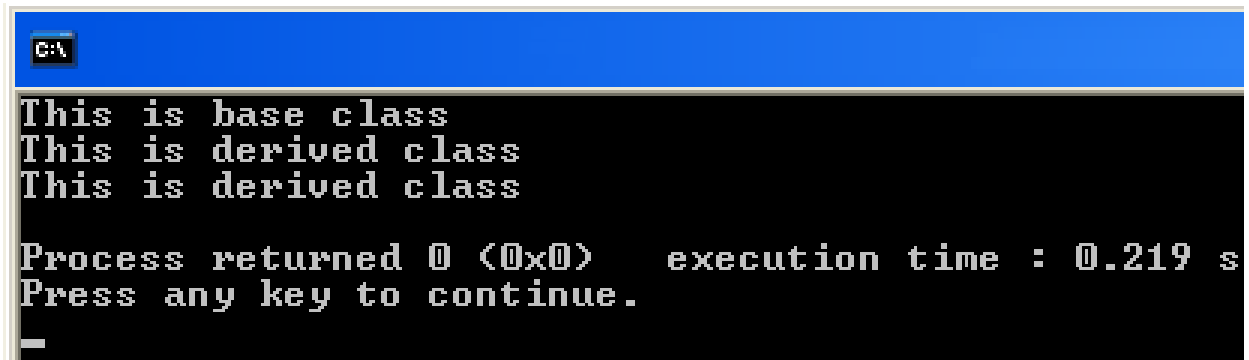
Віртуальну функцію можна викликати як будь-яку іншу компонентну функцію. Однак для підтримки динамічного поліморфізму віртуальні функції викликають через покажчик базового класу, який використовується в якості посилання на об'єкт похідного класу.

Якщо об'єкт похідного класу містить віртуальну функцію і віртуальна функція викликається за допомогою цього покажчика, то при компіляції викликається та функція, яка відповідає типу об'єкта, на який посилається покажчик.

За наявності кількох похідних класів від містить віртуальну функцію базового класу, то при посиланні покажчика базового класу на різні об'єкти цих похідних класів будуть виконуватися різні версії віртуальної функції.

Переобумовленої віртуальна функція повинна мати ті ж тип, число параметрів і тип значення, що повертається. Віртуальна функція повинна бути компонентом класу.

Якщо в прикладі визначити метод print () як віртуальний, тобто додати одне слово virtual, то результат роботи програми буде таким



```
C:\
This is base class
This is derived class
This is derived class

Process returned 0 (0x0)   execution time : 0.219 s
Press any key to continue.
_
```


поліморфний клас - клас, в якому визначені віртуальні функції (хоча б одна).

Ключове слово *virtual* можна писати тільки в базовому класі, в оголошенні функції, при цьому перевизначена функція буде вважатися віртуальною.

Правила опису та використання віртуальних функцій-методів:

- 1.** Віртуальна функція може бути тільки методом класу.
- 2.** Будь перевантажується метод класу можна зробити віртуальним, наприклад операцію присвоювання або операцію перетворення типу.
- 3.** Віртуальна функція успадковується.
- 4.** Віртуальна функція може бути константною.

5. Якщо в базовому класі визначена віртуальна функція, то метод похідного класу з таким же ім'ям і прототипом (включаючи і тип значення, що повертається, і константність методу) автоматично є віртуальним і заміщає метод базового класу.

6. Статичні методи не можуть бути віртуальними.

7. Конструктори не можуть бути віртуальними.

8. Деструктори можуть (частіше - повинні) бути віртуальними - це гарантує коректне звільнення пам'яті через покажчик базового класу.

9. Константний метод вважається відмінним від неконстантного методу з таким же прототипом.

Приклад. Віртуальні функції.

```
#include <iostream>
using namespace std;
class Base
{
public:
    virtual int f() const
    {
        cout<<"Base::f()"<<endl;
        return 0;
    }
    virtual void f(const char &s) const
    {
        cout<<"Base::f(char)"<<endl;
    }
};

class Derive: public Base
{
public:
    virtual int f(int) const
    {
        cout<<"Derive::f(int)"<<endl;
        return 0;
    }
}
```

```
// virtual int f() const
// {
//     cout<<"Derive::f()"<<endl;
//     return 0;
// }
};

int main()
{
    Base b, *pb;
    Derive d, *pd = &d;
    pb = &d;
    pb->f();      pb->f('n');
    pb->f(10);    pd->f(10);
    return 0;
}
```

C:\

```
Base::f()
Base::f(char)
Base::f(char)
Derive::f(int)
```

```
Process returned 0 (0x0)
Press any key to continue.
```



Якщо розкоментувати закоментувавши частина коду, що буде на консолі в результаті роботи програми?

```
Derive::f()  
Base::f(char)  
Base::f(char)  
Derive::f(int)  
  
Process returned 0 (0x0)   execution time : 0.234 s  
Press any key to continue.
```

Через покажчик базового класу можна викликати нові віртуальні методи, визначені тільки в похідному класі.

Явна приведення типу покажчика:

((Derive *) pb) -> f (2);

Віртуальну функцію можна викликати невіртуальну, якщо вказати кваліфікатор класу:

```
Base *cl = new Derive();  
cl->print();           // вызов метода-наследника  
cl->Base::print();     // явно вызывается базовый метод
```

Такий статичний виклик буває потрібен, коли в базовому класі реалізуються спільні дії, які повинні виконуватися в усіх класах-спадкоємців.

При наявності хоча б однієї віртуальної функції розмір класу без полів дорівнює 4 - це розмір покажчика. Кількість віртуальних функцій ролі не грає - розмір класу залишається дорівнює 4.

```
cout<<sizeof(Base)<<endl;  
cout<<sizeof(Derive)<<endl;
```

3. Раннє і пізніше зв'язування.

статичний поліморфізм реалізується через перевантаження функцій і операцій.

динамічний поліморфізм - через використання віртуальних функцій.

Статичний і динамічний варіанти поліморфізму співвідносять з поняттями раннього і пізнього зв'язування.

раннє зв'язування стосується подій етапу компіляції програми, таких як виклик

- звичайних функцій,
- перевантажуються функцій, - невіртуальних компонентних функцій,
- дружніх функцій.

При виклику перерахованих функцій вся необхідна адресна інформація відома при компіляції.

Перевагою раннього зв'язування є висока швидкодія одержуваних здійснених програм.

Недоліком раннього зв'язування є зниження гнучкості програм.

пізніше зв'язування стосується подій, що відбуваються в процесі виконання програми. При виклику функцій з використанням пізнього зв'язування адреса викликається функції до початку виконання програми невідомий. Зокрема, об'єктом пізнього зв'язування є віртуальні функції.

При доступі до віртуальної функції через покажчик базового класу при виконанні програми визначається тип засиланого об'єкта і вибирається версія віртуальної функції для виклику.

Перевагою пізнього зв'язування є висока гнучкість виконуваної програми, можливість реакції на події.

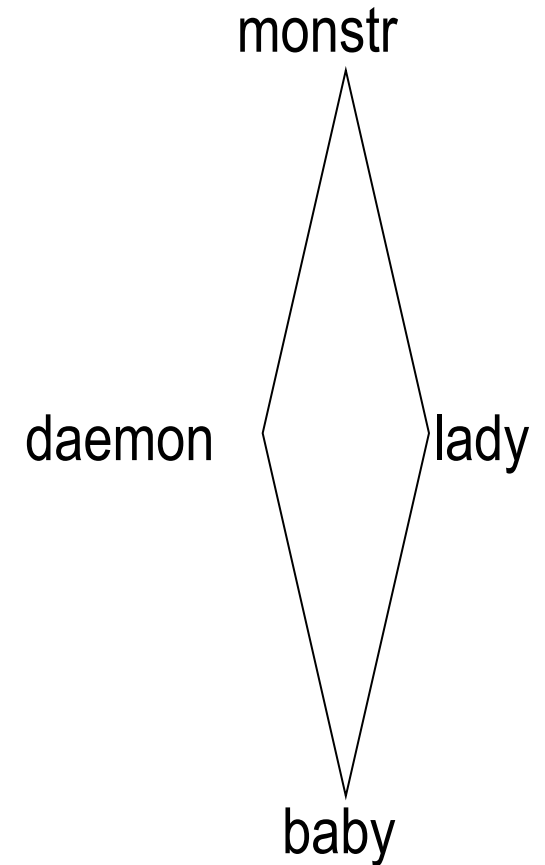
Недоліком є відносно низька швидкодія програми.

МНОЖИННЕ СПАДКУВАННЯ

```
class monstr{  
    public: int get_health(); ...  
};  
class hero {  
    public: int get_health(); ...  
};  
class ostrich: public monstr, Public hero {... };
```

```
int main () {  
    ostrich A;  
    cout << A.monstr::get_health();  
    cout << A.hero::get_health();  
}
```

```
class monstr {  
    ...  
};  
class daemon: virtual public monstr {  
    ...  
};  
class lady: virtual public monstr {  
    ...  
};  
class baby: public daemon, public lady {  
    ...  
};
```



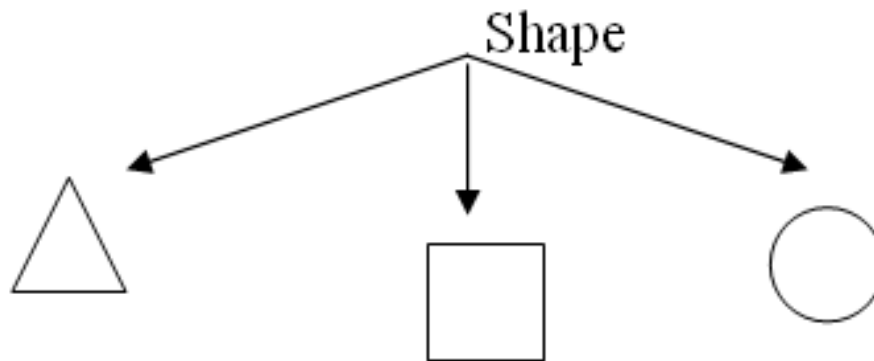
абстрактні класи.

план:

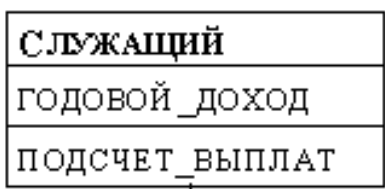
- 1. Поняття абстрактного класу.*
- 2. Чисті віртуальні функції та абстрактні класи.*
- 3. Віртуальні деструктори.*

поняття абстрактного класу.

абстрактний клас - це клас, який висловлює якусь загальну концепцію, яка відобразить основну ідею для використання в похідних класах. Абстрактний клас створюють тільки для того, щоб на його основі створювати інші класи. Створювати екземпляри об'єктів таких класів не можна, тому їх називають абстрактними.



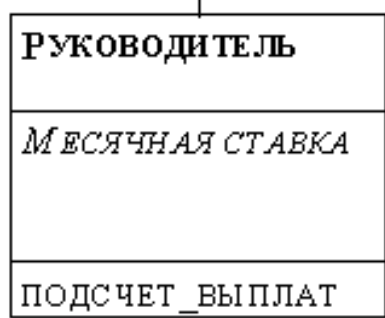
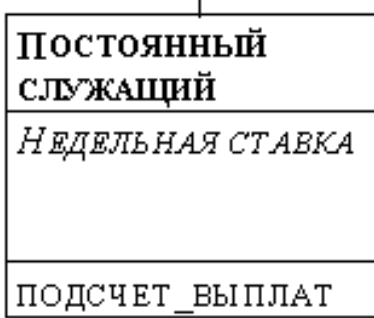
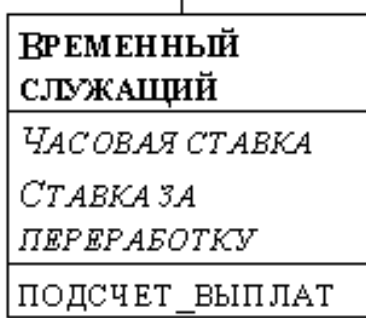
АБСТРАКТНЫЙ
КЛАСС



← атрибут

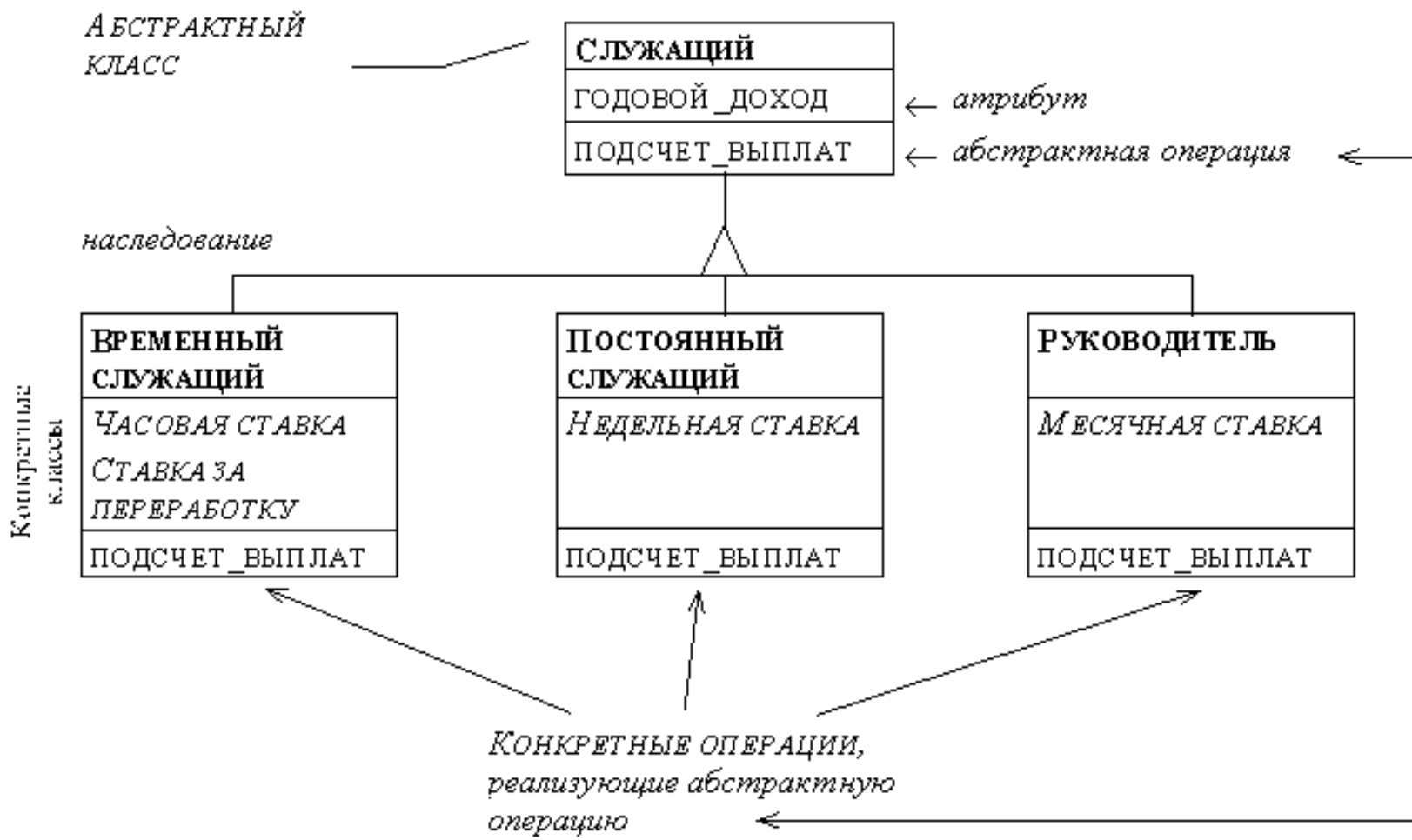
← абстрактная операция

наследование



Конкретные
классы

КОНКРЕТНЫЕ ОПЕРАЦИИ,
реализующие абстрактную
операцию



Щоб клас зробити абстрактним, потрібно оголосити одну або більше його функцій чисто віртуальними.

virtual *тіп_возвр_значенія імя_функції (параметри) = Ø;*

```
virtual void print () = 0;
```

*Класи, що мають хоча б одну чисто віртуальну функцію, називають **абстрактними**, і не можуть використовуватися для створення об'єктів.*

```
#include <iostream>
using namespace std;

class Shape
{
public:
    virtual double area () = 0 ;
};

class Circle : public Shape
{
    double radius;
public:
    Circle (double q) { radius = q; }
    double area () { return 3.14*radius*radius; }
};

class Triangle : public Shape
{
    double height, length;
public:
    Triangle (double a, double b) { height = a; length = b; }
    double area () { return height*length*0.5; }
};
```

Приклад.

Абстрактний клас.

```

class Rectangle : public Shape
{
    double height, length;
public:
    Rectangle (double a, double b) { height = a; length = b; }
    double area () { return height*length; }
    void print_area (Shape*ptr) { cout<<"Area = "<<ptr->area()<<endl; }
};

int main()
{
    Circle c(5);
    Triangle t(4, 32);
    Rectangle r(2,7);
    r.print_area (&c);
    r.print_area (&t);
    return 0;
}

```

```

C:\
Area = 78.5
Area = 64

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.

```


2. Чисті віртуальні функції та абстрактні класи.

Віртуальна функція, яка не має визначення тіла, називається **чистої (pure)** і оголошується наступним чином:

virtual тип ім'я (параметри) = 0;

Передбачається, що дана функція буде реалізована в класах-спадкоємців.

Клас, в якому є хоч одна чиста віртуальна функція, називається абстрактним класом. Екземпляри абстрактного класу створювати заборонено.

При спадкуванні абстрактність зберігається: якщо клас-спадкоємець не реалізує успадковану чисту віртуальну функцію, то він теж є абстрактним.

Чисті (pure) віртуальні функції використовуються в разі, коли в віртуальній функції базового класу відсутній значущу дію. При цьому в кожному похідному класі від заданого класу така функція повинна бути обов'язково перевизначена.

Віртуальні деструктори.

*Деструкція класу може бути оголошений **віртуальним**. Коли деструктор базового класу віртуальний, то і деструктори всіх спадкоємців такі ж.*

Деструкція необхідно оголошувати віртуальним, якщо доступ до динамічного об'єкту похідного класу виконується через покажчик базового класу.

В цьому випадку при знищенні об'єкта через покажчик базового класу викликається деструктор похідного класу, а він викликає деструктор базового класу. Якщо деструктор базового класу не віртуальний, то при знищенні об'єкта похідного класу через покажчик базового класу викликається деструктор тільки базового класу.

Приклад. Віртуальні деструктори.

```
#include <iostream>
using namespace std;
class Base
{
public:
    ~Base()
    {    cout<<"Base::Not virtual destructor!"<<endl; }
};

class Derived : public Base
{
public:
    ~Derived()
    {    cout<<"Derived::Not virtual destructor!"<<endl; }
};
```

```

class VBase
{
public:
    virtual ~VBase()
    {   cout<<"Base::Virtual destructor!"<<endl;   }
};

class VDerived : public VBase
{
public:
    ~VDerived()
    {   cout<<"Derived::Virtual destructor!"<< endl;
    }
};

int main()
{
    Base *bp = new Derived();
    delete bp;
    VBase *vbp = new VDerived();
    delete vbp;
    return 0;
}

```

```

C:\
Base::Not virtual destructor!
Derived::Virtual destructor!
Base::Virtual destructor!

Process returned 0 (0x0)   execution time : 0.203 s
Press any key to continue.

```

Деструкція може бути оголошений як чистий віртуальний:

virtual ~ VBase () = 0;

Клас, в якому визначено чистий віртуальний деструктор, є абстрактним, і створювати об'єкти цього класу заборонено. Однак клас-спадкоємець не є абстрактним класом, оскільки деструктор не успадковується, і в наступнику при відсутності явно певного деструктора він створюється автоматично. При оголошенні чисто віртуального деструктора потрібно написати і його визначення.