ТЕМА: Дочірні класи.

Загальна будова дочірнього класу

На основі раніше побудованих класів можна створювати нові класи шляхом модифікацій і доповнень існуючих класів. В цьому разі кажуть, що новий клас успадковує інший клас.

Це дозволяє розв'язати дві проблеми:

1) не переписувати ще раз елементи класу, які зарекомендували своє позитивне значення, і можуть бути використані повторно;

2) будувати на початку загальні класи для властивостей, характерних для сукупності споріднених класів, а пізніше їх поступово розширювати

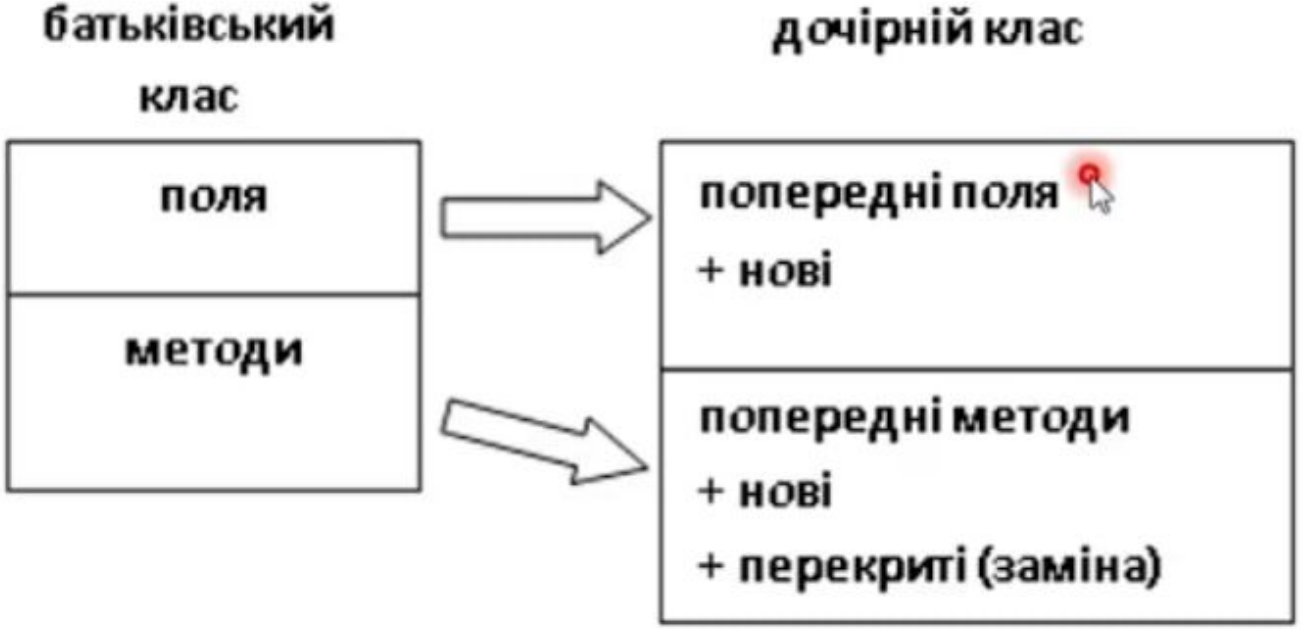
В стандартній термінології C++ клас, який є основою ієрархії, називають базовим, а клас, який успадковує властивості базового класу, - називають похідним. Похідні класи, своєю чергою, можуть бути базовими стосовно інших класів.

Терміни «базовий»-«похідний» ще позначають термінами «батьківський»-«дочірній».

Нова задача. Постановка початкової задачі змінилась.

Окрім визначеного раніше змісту об'єкта, потрібно мати додатково характеристику масиву, позначену однією літерою.

Загальна схема будови дочірніх класів є така:



Базові засади успадкування.

1) В дочірніх класах поля можна лише додавати, замінити неможливо.

2) Методи можна як додавати, так і замінити – перекрити.

3) При цьому не успадковують: всі конструктори; деструктор; перевантажений оператор присвоєння. Їх треба щоразу визначати в дочірніх класах.

4) Для успадкованих методів треба мати на увазі наступне. Методи батьківського класу були складені для параметрів і результатів цього ж класу `Stat`, наприклад:

```
Stat operator+ (const Stat & ob) // перевантаження операції +
{ // до власного об'єкта додати об'єкт ob
  Stat temp; // це буде результат додавання
  // . . . . . .
  return temp;  // повернути копію об'єкта temp
}
```

Якщо підставляти параметри типу `CharStat` дочірнього класу, то приведення їх до типу `Stat` виконується автоматично. Проте результат типу `Stat` привести до типу `CharStat` не можна за правилами C++ (про це буде далі). Тому треба уважно вивчити зміст кожного успадкованого метода і визначити, чи не треба його доповнити або цілком замінити.

Проект дочірнього класу

Отже, в дочірньому класі потрібно мати додатково характеристику масиву, позначену однією літерою. Складаємо проект доповнень і виправлень:

- додати пове поле `char key` – характеристику масиву;
- додати нові методи доступу до такого поля `SetKey()`, `GetKey()`;
- конструктори визначаємо заново, врахувавши зміст попередніх конструкторів; деструктор визначаємо для нового класу, якщо є потреба в додаткових операціях;
- обидва методи друкування `Print()` не враховують нового поля `key`, отже їх потрібно доповнити, тобто перекрити;
- аналогічно – щодо дружньої функції друкування об'єкта;
- перевантажений оператор присвоєння визначаємо на основі змісту дочірнього класу;
- перекрити методи, які залежать від переліку полів:

```
CreateFromEven() // створити об'єкт з елементів парних номерів
```

- методи, які не залежать від нового поля `char key`, перекривати непотрібно; проте маємо зважити, що успадковані методи повертають результат типу `Stat`, а не `CharStat`; якщо ж все таки треба мати результат `CharStat`, а не `Stat`, їх потрібно також перекрити.

Всі зазначені елементи проекту записуємо в новий окремий файл charstat.h:

```
#pragma once  // підключати цей файл однократно
//  ДОЧІРНІ КЛАСИ  у файлі charstat.h
#include  <iostream>  // читання і запис стандартних потоків
#include  <fstream>  // файловий ввід-вивід
#include  "stat3.h"  // файл батьківського класу Stat
using  namespace  std;

// дочірній клас - похідний
class CharStat : public Stat  // (1) спосіб успадкування, від кого
{
protected:
  char key;  // (2)  "код" масиву
public:
  CharStat(int s, int m, char c) : Stat(s,m)
  { // (3)  конструктор з параметрами
    // спочатку викликають Stat(s,m) конструктор батьківського класу
    key=c; // нове поле дочірнього класу
  }

  CharStat() { key='?'; }   // (4)  конструктор без параметрів
  // спочатку викликають Stat() батьківського класу без параметрів
```

```
// конструктор копіювання
CharStat(CharStat & ob) : Stat(ob)   { this->key = ob.key; }

// деструктор залишаємо за замовчуванням //  5
//    але після нього буде виконаний деструктор батьківського класу

void SetKey(char c) { key=c; }    //  6

char GetKey() { return key; }    //  6

void Print()    //  7
{
  Stat::Print();  // метод Print() батьківського класу
  cout << "Code= " << key << endl;
     // друкуємо окремо додаткове поле key
}
```

```
void Print(char * filename)
{ // друкувати значення у файл filename //  (7)
  // перекрити повністю, бо файл відкриваємо і закриваємо тут
  ofstream outres(filename);  // потік виведення
  outres << "Current values:\n";
  for(int i=0; i<size; i++) outres << mas[i] << "  ";
  outres << endl;
  outres << "ave=" << ave << "  max=" << max << endl;
  outres << "Code= " << key << endl;
  outres.close();
}

friend ostream & operator<< (ostream & os, const CharStat & ob)  (8)
{ // перевантаження дружньої функції друкування об'єкта
  cout << (Stat)ob; // дружня функція для батьк. класу Stat
  cout << "Code= " << ob.key << endl; // нове поле
  return os; // !  повернути об'єкт друкування
             //  для випадку продовження
}
```

```
  CharStat & operator= (const Stat & ob)
  { // перевантаження операції =    // (9)
    this -> Stat::operator=((Stat)ob);  // операція батьк. класу
    if(typeid(ob)==typeid(*this)) // якщо парам. дочірнього класу
    {
      void * p ; // використати вказ. void* для заміни типу далі
      p = (void*)(&ob);
      this->key = ( *(CharStat*)p ).CharStat::key;
                // копіюємо додаткове поле
    }
    else  this->key = '*';  // якщо параметр батьківського класу
    return *this; // ! на випадок множинного присвоєння
  }

  CharStat * CreateFromEven()
  { // новий об'єкт з елементів парних номерів   // (10)
    CharStat * p = new CharStat( (size+1)/2,0,key); // динам. об'єкт
    for(int i=0; i < p->size; i++)  p->mas[i] = mas[2*i];
    p->Calc();  // обчислити за новими значеннями
    return p;
  }
} ; // кінець визначення класу
```

Аналіз будови дочірнього класу

1. В заголовку вказують батьківський клас і спосіб успадкування. Це визначає спосіб зміни статусу елементів батьківського класу при їх переході в дочірній.

Якщо записано public, тоді

public –► public, protected –► protected,

private –► не передаються (не доступні в дочірньому класі)

Якщо записано protected, тоді

public –► protected, protected –► protected, private –► не передаються.

Якщо записано private, тоді

public –► private, protected –► private, private –► не передаються.

Спосіб зміни статусу можна не вказувати, тоді за замовчування вважається private.

2. Записуємо нові поля, які додають до полів базового класу.

3. Як було сказано, конструктори і деструктор у спадок не передаються, отже для кожного дочірнього класу їх потрібно визначати заново. В мові C++ діє правило: кожен конструктор дочірнього класу на початку виконання автоматично викликає конструктор батьківського класу, після чого виконує власні оператори. В заголовку конструктора вказуємо, який саме конструктор батьківського класу потрібно викликати і з якими параметрами.

4. Якщо не вказати, який конструктор батьківського класу потрібно викликати, тоді за замовчуванням викликається конструктор без параметрів батьківського класу. Якщо виявиться, що такого немає, буде зафіксована помилка компіляції.

5. Новий деструктор можна не визначати. Тоді за замовчування автоматично створюється порожній для дочірнього класу. Правило щодо деструктора: деструктор дочірнього класу спочатку виконує власні операції, після чого автоматично викликає деструктор батьківського класу. Його вказувати не потрібно, бо він один. Отже, дія деструктора є симетричною до дії конструктора.

6. Для нових полів дочірнього класу додаємо нові методи для доступу, а також за потреби – інші.

7. Перекриті однойменні методи батьківського класу для врахування нового поля `key`. Звернемо увагу на спосіб виклику методу батьківського класу (не обов'язково однойменного) `Stat::Print()`.

8. Дружню функцію друкування об'єкта треба перекрити, щоб врахувати нове поле `key`. Звернемо увагу на спосіб виклику такої ж дружньої функції батьківського класу `cout<<(Stat)ob`.

9. Перевантажений оператор присвоєння визначаємо заново, бо він не успадковується. Звернемо увагу, що параметр операторної функції має тип `Stat`, а не `CharStat`. Це дозволяє комбінувати різні варіанти присвоєння об'єктів батьківського і дочірнього класів. Наступна таблиця пояснює стан виконання операції.

Нехай маємо `Stat A,B;  CharStat C,D;` Тоді:

`A=B;`	викликається `Stat & operator= (const Stat & ob){...}` батьк. класу
`A=C;`	викликається `Stat & operator= (const Stat & ob){...}` батьківського класу, де параметр `C` дочірнього класу `CharStat` автоматично приводиться до типу `Stat`; тобто, приймається до уваги лише спільна частина полів `CharStat` і `Stat`
`D=C;`	викликають `CharStat & operator= (const Stat & ob){...}` дочірнього класу; параметр `C` дочірнього класу `CharStat` автоматично приводиться до типу `Stat`, виконуючи спочатку присвоєння за операцією батьківського класу; якщо тип `typeid(ob)` параметра є `CharStat`, тоді додатково копіюємо поле `key`; звернути увагу на спеціальний спосіб перемикання вказівника від батьківського класу до дочірнього, бо пряме перемикання не дозволено мовою C++
`D=B;`	викликають `CharStat & operator= (const Stat & ob){...}` дочірнього класу; спочатку виконуємо присвоєння операцією батьківського класу; параметр `B` батьківського класу поля `key` немає, тому надаємо полю деякого значення `'*'`

10. Метод `CreateFromEven()` будує новий об'єкт типу `CharStat` і перекриває такий самий метод батьківського класу. Якщо його не перекрити, а успадкувати, тоді результат буде типу `Stat`, де відсутнє поле `key`.

Ієрархія класів

1) Дочірній клас своєю чергою може бути батьківським до наступного похідного. Тоді кажуть, що успадкування класів виконують «вниз».

2) На основі спільного батьківського класу можна будувати різні дочірні. Тоді кажуть, що успадкування класів виконують «в ширину».

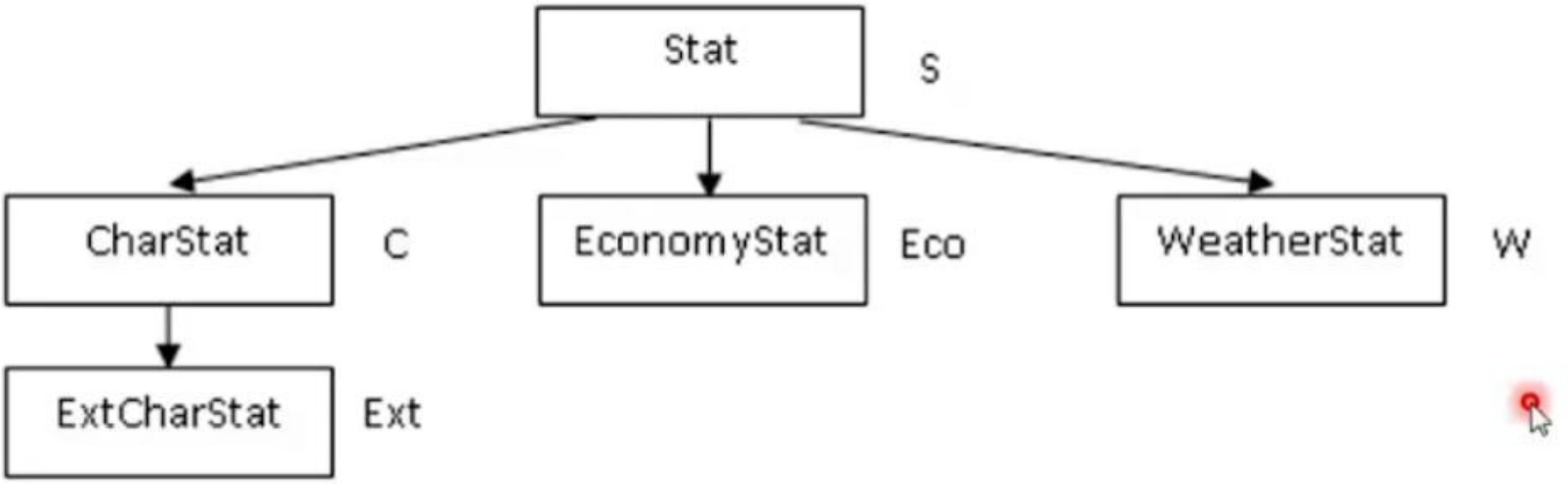
Наприклад, визначимо такий перелік побудованих класів:

```
// наступний нисхідний клас
class ExtCharStat : public CharStat
{ // .......................
} ;

// інші похідні від Stat
class EconomyStat : public Stat
{ // .......................
} ;

class WeatherStat : public Stat
{ // .......................
} ;
```

На основі визначень класів можна зобразити схему ієрархії класів, яка показує їх підлеглість:



Ніяких обмежень на розбудову схеми ієрархії класів немає. Успадкування класів за схемою ієрархії відбувається зверху вниз, але не в ширину.

Нехай тепер маємо об'єкти побудованих класів:

```
Stat S;  CharStat C;  ExtCharStat Ext;
EconomyStat Eco;  WeatherStat W;
```

Поведінка об'єктів визначається їх місцем в схемі ієрархії.

Правило присвоєння об'єктів

В загальному випадку присвоювати об'єкти один одному можна за схемою ієрархії лише в напрямку знизу вверх.

Тобто дочірній об'єкт може бути присвоєний батьківському або вище, навпаки – не можна. Правило стосується як статичних об'єктів, так і вказівників на об'єкти.

Не плутати напрям успадкування з напрямом присвоєння.

Присвоєння об'єктів означає копіювання полів, з врахуванням перевантаженого оператора присвоєння, якщо такий є. Дочірній клас завжди має полів не менше, ніж батьківський. У випадку копіювання дочірнього класу до батьківського будуть скопійовані лише співпадаючі поля.

Якщо пробувати копіювати батьківський до дочірнього, то в загальному випадку може не бути потрібних полів в батьківському класі. Проте, в принципі, це можливо, якщо визначити варіант перевантаженого оператора присвоєння, параметром якого є об'єкт батьківського класу, що ми й зробили для класу `CharStat`.

Отже, маємо такі правильні і неправильні варіанти присвоєнь:

```
S=C;  S=Ext;  C=Ext;  // правильні присвоєння
C=S;  // правильно - побудована відповідна перевантажена операція
Ext=S;  Ext=C;  // неправильні присвоєння
S=Eco;  S=W;  // правильні присвоєння
Eco = W;  // неправильно

// вказівники на об'єкти
Stat * R1 = new Stat(6,500);
CharStat * R2 = new CharStat(4,720,'X');
R1=R2; // правильно
R2=R1; // неправильно
```

Не забувати, що присвоєння вказівників не викликає копіювання об'єктів, а лише перемикання самих вказівників з одного об'єкта на інший.