C++ dilində sinif tipinin varislik xüsusiyyətləri

Leyla Elçin qızı Mehdiyeva
Gəncə Dövlət Universitetinin
İnformatika kafedrasının müəllimi
E-mail: n.leyla95@mail.ru

Rəyçilər: p.e.d., prof. Ə.Q. Pələngov,
t.ü.f.d., dos. M.A. Həsənova

Açar sözlər: siniflər, baza sinfi, sinfin varislik xüsusiyyətləri, iyerarxiya, müraciət açarları, virtual metodlar

Ключевые слова: классы, базовый класс, свойства наследования класса, иерархия, ссылочные ключи, виртуальные методы

Key words: classes, base class, class inheritance properties, hierarchy, addressing keys, virtual methods

Sinfi təsvir edərkən başlığında onun bütün baza sinifləri sadalanır. Sinifdə baza siniflərin elementlərinə müraciət *private, protected* və *public* açar sözləri vasitəsilə edilir (1):

class ad: müraciət_açarı baza_sinfin_adı {// sinfin gövdəsi};

Məsələn, bilirik ki, bütün növ üçbucaqlılar, dördbucaqlılar, beşbucaqlılar və s. fiqurlar çoxbucaqlılar ailəsinə daxildirlər. Ch_b sinfini baza sinfi (valideyn, əcdad) qəbul etsək, onda onun törəmələri (varisləri) olan Uch_b və D_b sinifləri aşağıdakı kimi təsvir edilər:

```
class Ch_b {
 //...  };
class Uch_b: public Ch_b {
public: Show ( ) ; };
class D_b: public Ch_b {
public:  Show ( ) ; };
```

Burada törəmə siniflər olan *Uch_b* və *D_b, Ch_b* baza sinfinin bütün elementlərini vərəsə kimi saxlayırlar, lakin onların hər biri ayrıliqda həm də *Show* metoduna malikdirlər. Əgər sinfin baza siniflərinin sayı çox olarsa, bu halda onlar vergüllə bir-birindən ayrılırlar, bu zaman müraciət açarları hər bir sinfin əvvəlində yazıla bilər. Məsələn, paraleloqram dördbucaqlının xüsusi növlərindən biri olduğu üçün o həm *Ch_b,* həm də *D_b* sinfinin varisi hesab oluna bilər:

```
class Ch_b {
 //...  };
class Uch_b: public Ch_b {
public: Show ( ) ; };
class D_b: public Ch_b {
public: Show ( ) ; };
class Paraleloqram: public Ch_b, private D_b{
public: Show ( ) ; };
```

Əlaqəli siniflərin belə yığımı siniflər iyerarxiyası adlanır.

Sinfin elementlərinə *private* və *public* müraciət spesifikatorları vasitəsilə, eləcə də *protected* spesifikatorundan da istifadə etmək olar (2). *protected* spesifikatoru övlad olmayan, yəni

iyerarxiyaya daxil olmayan tək siniflərin elementlərinə müraciətdə işlədilən *private* spesifikatoru ilə eynigüclüdür. Onlar arasındakı fərq yalnız varislik zamanı araya çıxır ki, bu fərqləri də aşağıdakı cədvəldən aydın görmək olar:

Müraciət açarı	**Baza sinfinin spesifikatoru**	**Törəmə (övlad) sinifdə müraciət**
Private	*Private*	Yoxdur
	Protected	*Private*
	Public	*Private*
protected	*Private*	Yoxdur
	Protected	*Protected*
	Public	*Protected*
Public	*Private*	Yoxdur
	Protected	*Protected*
	Public	*Public*

Cədvəldən göründüyü kimi, müraciət açarından asılı olmayaraq törəmə sinifdən baza sinfin *private* elementlərinə müraciət mümkün deyil. Bu elementlərə yalnız baza sinfin metodları vasitəsilə müraciət etmək olar.

private açarı ilə varislikdə *protected* elementləri törəmə sinifdə *private* elementləri kimi də qalırlar, qalan hallarda onlara müraciət qaydaları dəyişmirlər. Varislik zamanı *public* elementlərinə müraciət uyğun müraciət açarı ilə edilir. Əgər baza sinfinin elementləri *private* açarı vasitəsilə törəmə sinfə vərəsə verilirsə, onda törəmə sinifdə seçməklə onun bəzi elementlərinə müraciət etmək olar (3). Bunun üçün törəmə sinfin *public* bölməsində görünmə oblastına müraciət əməli (::) ilə onları elan etmək lazımdır. Məsələn:

```
    class Ch_b {
 …
public: void f ( );  };
class Uch_b: private Ch_b {
public: Ch_b:: void f( );
        Show ( ) ; };
```

Bəzən baza sinfin metodlarını törəmə sinifdə bir qədər dəyişdirmək lazım gəlir. Metodun dəyişdirilməsi üçün törəmə sinifdə metodu eyni adla (metodun baza sinifdə olan adı ilə) elan etmək lazımdır. Əgər baza sinfin metodunu dəyişdirmədən çağırmaq lazım gələrsə, onda yuxarıdakı nümunədə göstərildiyi kimi görünmə oblastına müraciət əməlindən istifadə olunur, məsələn:

```
#include <iostream>
using namespace  std;
class Valideyn {
public : void Display ( ) {cout<<"Salam, yoldash!"; }};
class Evlad : public Valideyn {
public : void Display ( ) {
Valideyn :: Display ( );    //Baza sinfin metodunun chagrilmasi
cout<<"Necesiniz, yoldash!"; }}; //Baza sinfin metodunun deyishdirilmesi
class Neve : public Evlad{
public : void Display ( ) {
```

```
Evlad :: Display ( ); //Baza sinfin metodunun chagrilmasi
cout<<"Xudahafiz!"<<endl; }}; //Baza sinfin metodunun deyishdirilmesi
int main ( ) {
Neve  x;
x.Display ( ); return 0;}
```

Törəmə sinfin yalnız bir valideyni olduqda belə varislik sadə varislik adlanır. Sinfin müxtəlif metodları üçün müxtəlif varislik qaydaları mövcuddur. Məsələn, törəmə sinfə valideyn sinfin konstruktor və mənimsətmə əməlləri irsən verilmir, destruktorlar isə verilir. Siniflərin varisliyini və bu zaman yaranan problemləri nümunə üzərində nəzərdən keçirək.

Müxtəlif metodların varislik qaydalarına baxaq.

Konstruktorlar irsən ötürülmədiklərinə görə övlad sinifdə öz konstruktorları təyin edilməlidir (1). Konstruktorun çağırılması qaydaları aşağıdakı kimi təyin edilir:

- Əgər törəmə sinifdə baza sinfinin konstruktoruna aşkar müraciət verilməyibsə, onda susmaya görə baza sinfinin konstruktoru avtomatik çağırılır (yəni baza konstruktoru parametrlərsiz çağırılır). Bu daemon sinfinin birinci konstruktorunda edilmişdir.
- Bir neçə səviyyəli iyerarxiyayda baza konstruktorları ən yuxarı səviyyədən başlayaraq çağırılırlar. Bundan sonra sinifdə elan olunma ardıcıllığına görə sinfin obyekt olan elementlərinin konstruktorları, sonra isə övlad sinfin konstruktorları yerinə yetirilir.
- Törəmə sinfin bir neçə baza sinfi olduqda onların konstruktorları elan olunduqları ardıcıllığa görə çağrılırlar.
- Əgər baza sinfinin konstruktoru parametrlərin göstərilməsini tələb edirsə, onda o aşkar şəkildə törəmə sinfin konstruktorunun inisiallaşdırma siyahısında verilməlidir (bu nümunədəki son üç konstruktorda göstərilmişdir).

Mənimsətmə əməli də irsən ötürülmür, odur ki, onu da *daemon* sinfində aşkar şəkildə təyin etmək lazımdır. Funksiya-əməlin yazılışına fikir versək görərik ki, onun gövdəsində birbaşa aşkar şəkildə baza sinfin mənimsətmə funksiya-əməli çağırılmışdır. Çağrılmanın yazılış sintaksisini aydın təsəvvür etmək üçün operator açar sözünü əməl işarəsi ilə birlikdə funksiyanın adı kimi qəbul etmək lazımdır.

Destruktorların varislik qaydaları aşağıda sadalanmışdır:

- Destruktorlar irsən ötürülmürlər, əgər proqramçı törəmə sinifdə destruktoru təsvir etməyibsə, o susmaya görə təyin edilir və bütün baza siniflərin destruktorları çağırılır.
- Konstruktordan fərqli olaraq törəmə sinfin destruktorunun yazılışında baza siniflərin destruktorlarını aşkar şəkildə çağırmaq tələb olunmur.
- Bir neçə səviyyəli iyerarxiyayda destruktor konstruktorların əksinə olaraq birinci sinfin, sonra sinfin elementlərinin, ən sonda isə baza sinfinin destruktorları çağırılırlar.

Törəmə sinifdən baza sinfinin təyin edilmiş metoduna müraciət görünmə oblastına müraciət əməli (::) və metodun adı vasitəsilə edilir (1). Beləliklə, deyə bilərik ki, törəmə sinif baza sinfinin hərəkətlərini nəinki tamamlamaq, hətta onları tənzimləmək imkanına da malikdir. Adətən obyektlərlə iş göstəricilər vasitəsilə yerinə yetirilir. Baza sinfinə istinad edən göstəriciyə istənilən törəmə sinfin obyektinin ünvanını mənimsətmək olar (açıq varislikdə):

```
//  Baza sinfinə istinad edən göstərici təsvir edilir:
monstr *p;
// Göstərici törəmə sinfin obyektinə istinad edir:
p=new daemon;
```

Belə müraciəti istənilən vaxt etmək mümkün olmur. Çünki proqramın icrası prosesində göstərici müxtəlif vaxtlarda iyerarxiyanın müxtəlif siniflərinin obyektlərinə istinad edə bilər

və eləcə də kompilyasiya prosesində konkret sinif naməlum da ola bilər. Məsələn, parametri baza sinfinin obyektinə göstərici olan funksiya. Proqramın icrası zamanı bu göstərici istənilən törəmə siniflərin obyektlərinə də istinad edə bilər.

C++ dilində ilkin əlaqələndirmə ilə yanaşı, gec əlaqələndirmə mexanizmi də reallaşdırılmışdır. Gec əlaqələndirmə mexanizmi proqramın icra mərhələsində metodu çağıran konkret obyektin tipindən asılı olaraq bu metoda istinad etməyə sistem tərəfindən icazə verilməsi prosesidir (2). Əlaqələndirmə mexanizm virtual metodlar vasitəsilə reallaşdırılır. Virtual metodu təyin etmək üçün *virtual* spesifikatorundan istifadə olunur, məsələn:

virtual void draw(int x, int y, int scale, int position)

Virtual metodların yazılış və istifadə qaydalarını nəzərdən keçirək:

- Əgər baza sinfində metod virtual metod kimi təyin edilibsə, törəmə sinifdə həmin adla və parametrlərlə təyin olunan metod avtomatik olaraq virtual metod kimi qəbul edilir.
- Virtual metodlar irsən ötürülürlər, yəni törəmə sinifdə onları əvvəlcədən təyin etmək yalnız o zaman tələb edilir ki, metoda valideyn metoddan fərqli hərəkətlər vermək lazım gəlsin. Bu zaman müraciət qaydasını dəyişmək olmaz.
- Əgər virtual metod törəmə sinifdə əvvəlcədən təyin edilibsə, onda onun obyektləri baza sinfinin metodlarına görünmə oblastına müraciət əməli (::) vasitəsilə müraciət edə bilərlər.
- Virtual metod static modifikatoru ilə elan edilə bilməz, lakin dost metod kimi elan edilə bilər.
- Əgər sinifdə virtual metodun təsviri verilirsə, o "təmiz" virtual metod kimi təyin edilməlidir. Təmiz virtual metod metodun gövdəsi əvəzinə xüsusiyyət=0 informasiyasını saxlayır, məsələn:

virtual void f(int)=0;

Təmiz virtual metod törəmə sinifdə əvvəlcədən təyin edilməlidir.

Əgər *monstr* sinfində *draw* metodunu virtual metod kimi təyin etsək, onda göstəricinin istinad etdiyi hansı sinfin metodunun seçilməsi icazəsi obyektin tipindən asılı olaraq sistem tərəfindən veriləcək.

```
monstr *r, *p;
r=new  monstr;            //monstr sinfinin obyekti yaradılır
p=new  daemon;             //daemon  sinfinin obyekti yaradılır
p-> draw (1, 1, 1, 1);     //monstr :: draw metodu çağrılır
p-> draw (1, 1, 1, 1);     //daemon :: draw metodu çağrılır
p-> monstr :: draw (1, 1, 1, 1);  // virtual metodlar mexanizmindən yankeçmə
```

Əgər *daemon* sifinin obyekti *draw* metodu ardıcıl olmayaraq müəyyən vaxtlarda (yəni *monstr* sinfində təyin olunan başqa metodlardan da) çağıracaqsa, onda bu zaman *daemon* sinfinin *draw* metodu çağrılacaq.

Beləliklə, deyə bilərik ki, proqramın yerinə yetirilməsi mərhələsində metoddan istinada icazə verilirsə, belə metod *virtual metod* adlanır ("*virtual*" ingilis sözü olub, burada "*faktiki*" deməkdir, yəni istinada faktik çağırışa görə icazə verilir). Varislik çoxluğu o deməkdir ki, sinif bir neçə baza siniflərinə malikdir. Əgər baza siniflərində eyni adlı elementlər mövcuddursa, bu zaman identifikatorlar arasında dolaşıqlıq yaranar. Bu səhvi aradan qaldırmaq üçün görünmə oblastına müraciət əməlindən istifadə edilir:

```
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 ( );  }
```

Proqram fraqmenti nümunəsindən göründüyü kimi *get_health* metodunun çağırılması üçün onun təsvir olunduğu sinfi aşkar şəkildə göstərmək vacibdir. Sinfin metoduna adi A. *get_health* kimi müraciət etsək, bu proqramda səhvlik yaradacaq, belə ki, kompilyator hansı baza sinfinin metoduna müraciət edildiyini ayırd edə bilməyəcək.

Əgər baza siniflərinin də ümumi bir əcdadı (valideyni) varsa, bu ona gətirəcək ki, baza siniflərinin törəmə sinfi öz əcdadlarının iki nüsxə sahələrinə malik olacaq, bu isə arzuolunmaz haldır. Bundan qaçmaq üçün varislikdə ümumi əcdadı virtual sinif kimi təyin edirlər:

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

baby sinfi *monstr* sinfinin sahələrinin yalnız bir nüsxəsinə malik olur. Əgər baza sinfi virtual sinif kimi irsən verilirsə, onda törəmə sinifdə adi qayda baza sinfinin hər bir sahəsinin bir virtual və bir qeyri-virtual nüsxəsi yaradılacaq.

Varislik çoxluğu törəmə sinifləri iki və daha çox baza siniflərin xüsusiyyətləri ilə təmin etmək üçün istifadə edilir[5]. Əksər hallarda törəmə sinfi baza siniflərindən biri əsas, qalanları isə əlavə xüsusiyyətləri ilə təmin edirlər (onlara çox vaxt qarışdırıcı siniflər də deyirlər). İmkan daxilində qarışdırıcı siniflərin virtual olmaları və parametrsiz konstruktor vasitəsilə yaradılmaları məsləhətlidir (bu rombşəkilli varislikdə (baza siniflərinin də ümumi bir əcdadı olduqda) bir çox problemlərin qarşısını alır).

Yuxarıdakı şərhləri yekunlaşdıraraq deyə bilərik ki, siniflərin varislik xüsusiyyətləri obyektyönlü iyerarxiya yaratmağa imkan verir, yəni törəmə siniflər (övlad tipli obyektlər) valideyn və ya baza siniflərin elementlərini (struktur və hərəkətini) irsən qəbul edir, onların xüsusiyyətlərini tamamlaya və ya dəyişdirə bilmək imkanlarına malik olurlar. Əlaqəsiz siniflərin sayı çox olduqda onları idarə etmək çətinləşir. Varislik bu problemi siniflərin nizamlanması və siraya düzülməsi yolu ilə aradan qaldırır, ümumi xüsusiyyətlərə malik sinifləri vahid bir sinifdə birləşdirərək ondan baza kimi istifadəyə imkan verir.

Beləliklə, iyerarxiyanın yuxarı səviyyələrində olan siniflər, özlərindən aşağıda yerləşən bütün siniflərin ən ümumi xüsusiyyətlərini daşıyırlar. İyerarxiya boyunca aşağı düşdükcə siniflər daha konkret xüsusiyyətlərə malik olurlar. Varislik çoxluğu bir sinfə iki və daha çox valideyn siniflərin xüsusiyyətlərini daşımağa imkan verir.

Məqalənin aktuallığı. Təlim keyfiyyətinin yüksəldilməsi təhsilin qarşısında daim duran problemlərdən biridir. Odur ki, hər hansı proqramlaşdırma dilinin əsas anlayışları və prinsiplərindən ibarət nəzəri əsaslarının öyrənilməsi ilə yanaşı, proqramlaşdırma bacarıqlarının mənimsədilməsi də zəruriyyət təşkil edir. Məqalədə bu istiqamətdə məsələlərə toxunulduğu üçün onun aktuallığı aydın olur.

Məqalənin elmi yeniliyi. Elmi yenilik virtual metodların yazılış və istifadə qaydalarının verilməsi, sinfin elementlərinə olan *private, public* və *protected* müraciət spesifikatorlarının

arasındakı fərqlərin göstərilməsi, müxtəlif metodların varislik qaydalarının araşdırılmasıdır.

Məqalənin praktik əhəmiyyəti və tətbiqi. Məqalədə C++ dilində sinif tipinin varislik xüsusiyyətləri ətraflı araşdırılmışdır. Bu baxımdan da məqalənin məzmunu ali və orta məktəb müəllimləri tərəfindən tədris materialı kimi, eləcə də bakalavr və magistrlər laborator məşğələlərdə mövzu ilə bağlı tapşırıqların yerinə yetirilməsi zamanı istifadə edə bilərlər.

Ədəbiyyat

1. Б. Страуструп. Программирование: принципы и практика использования C++ . Москва-Санкт-Петербург-Киев: Издательский дом «Вильямс», 2013.
2. M.A. Həsənova. C++ proqramlaşdırma dili. Gəncə, 2014.
3. Pələngov Ə. Həsənova M. C++ və Java proqramlaşdırma dilləri: Dərs vəsaiti. Bakı, 2019.
4. Э.А.Ишкова. C++ начала программирования: Учебник для вузов. Москва: Издательство «Бином-Пресс», 2009.
5. Гербет Шилдт. C++: методики программирования Шилдта. Москва-Санкт-Петербург-Киев: Издательский дом «Вильямс», 2009.

Л.Э. Мехтиева

Особенности наследования типа класса в C ++

Резюме

В статье рассмотрены особенности наследования типов классов языка программирования C++. Показано, что свойства наследования классов позволяют создавать объектно-ориентированную иерархию, управление которой становится затруднительным при большом количестве несвязанных классов. Наследование устраняет эту проблему путем упорядочения и сквенирования классов, объединяя классы с общими свойствами в один класс, позволяя использовать его в качестве базы. Кластер наследования позволяет классу нести свойства двух или более родительских классов.

L.E. Mehdiyeva

Inheritance features of a class type in C ++

Summary

The article examines the inheritance characteristics of the class type of C++ programming language. It has been shown that the inheritance properties of classes allow to create an object-oriented hierarchy, making it difficult to manage them when the number of unrelated classes is high. Inheritance eliminates this problem by regulating and sorting classes, combining classes with common features into a single class, allowing it to be used as a base. The majority of inheritance allows one class to carry the properties of two or more parent classes.

Redaksiyaya daxil olub: 02.07.2021