ارث بری

وراثت اجازه می دهد از یک نوع داده موجود نوع جدیدی را ایجاد کنید. کلاس مشتق شده رابط کلاس پایه را دارد می تواند به کلاس پایه upcast شود که در چندریختی مهم است. اگرچه استفاده مجدد کد توسط وراثت برای سرعت پیاده سازی را بالا می برد معمولا می خواهید سلسله مرتب کلاس خودتان را قبل از اینکه به دیگر برنامه نویسان اجازه استفاده بدهید دوباره طراحی کنید.

وراثت ایجاد ایجاد یک کلاس پایه عمومی که اشیا در خصوصیات آن مشترک هستند را می دهد. سایر کلاس ها از آن ارث بری دارند.

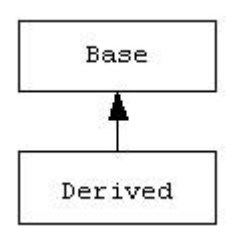
کلاس های پایه و مشتق شده
تعریف کلاس مشتق شده
کنترل دسترسی به اعضای کلاس پایه
وراثت چندگانه
سازنده ها و مخرب ها در وراثت

گاهی ناگذیریم کلاس جدیدی ایجاد کنیم که شباهت هائی با کلاسی دارد که قبلا ایجاد شده است و می خواهیم عملکردهائی را به آن اضافه کنیم این عمل توسط ارث بری (inheritance) امکان پذیر است. وراثت یکی از پایه های برنامه نویسی شیء گرائی است.

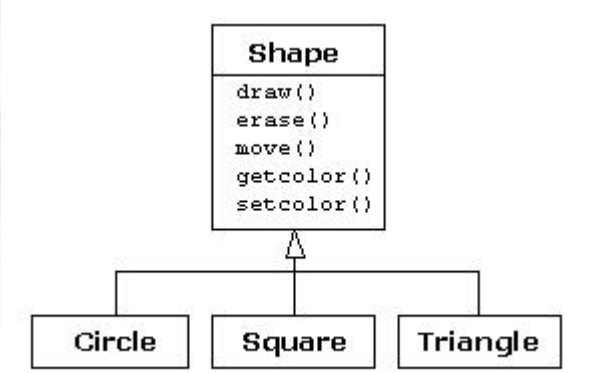
کلاس های پایه و مشتق شده

چند کلاس ممکن است خصوصیات و رفتارهای مشترکی داشته باشند اما هریک شامل خواص و توابع دیگری هم باشد. وراثت اجازه می دهد یک کلاس عمومی تعریف شود که اشیا درخصوصیات آن مشترک هستند و این کلاس می تواند توسط سایر کلاس ها ارث برده شود و خواص جدیدی به آن اضافه شود بدون اینکه تاثیری روی کلاس عمومی داشته باشد.

توارث شباهت بین دو کلاس را با استفاده از مفاهیم کلاس پایه (base) و کلاس مشتق شده (derived) بیان می کند. کلاسی که از آن ارث بری می شود کلاس پایه یا مبنا و کلاس وارث که خصوصیات کلاس پایه را به ارث می برد را کلاس مشتق شده می نامند. کلاس پایه شامل کلیه خواص و رفتارهائی است که بین کلاس های مشتق شده مشترک است.



مثال. کلاس پایه shape را درنظر بگیرید که دارای خاصیت های اندازه، رنگ و موقعیت است. هر شکل می تواند رسم شود، پاک شود، حرکت کند و رنگ شود. هر کدام از اشکال دارای خواص و رفتارهای اضافه تری هستند. برای یک شکل معین بعضی رفتارها ممکن است متفاوت باشد مثلا محاسبه مساحت.



نکته. یک کلاس متشق شده به نوبه خود می تواند کلاس پایه برای سایر کلاس ها باشد.
نکته. اگر کلاس پایه تغییر کند کلاس مشتق شده نیز تحت تاثیر این تغییرات قرار می گیرد.

تعریف کلاس مشتق شده

فرم کلی تعریف یک کلاس مشتق شده به صورت زیر است:

```
class derived : access base
{
   //members of new class;
}
```

derived نام کلاس جدید است که از کلاس پایه base مشتق شده است. قسمت access اختیاری است ولی می تواند public، private یا protected باشد و برای تعیین مجوز دسترسی اعضای کلاس پایه در کلاس جدید بکار می رود. اگر مجوز دسترسی ذکر نشود به این معنی است که کلیه اعضای عمومی کلاس پایه در کلاس مشتق شده به صورت خصوصی خواهند بود.

مثال. کلاس جدید Derived از کلاس Base مشتق شده است. در برنامه اصلی تابع change از کلاس Derived فراخوانی شده که خود دو تابع set و read از کلاس Base را صدا می زند.

```
#include <iostream.h>
class Base {
   int i;
protected:
   int read() { return i; }
   void set(int ii) { i = ii; }
public:
   Base()  { i=0; }
   int value(int m) { return m*i; }
};

class Derived : public Base {
   int j;
public:
   Derived() { j=0; }
   void change(int x) { set(x); cout<< read(); }
};

int main() {
   Derived d;
   d.change(10);
   cout << d.value(2) << endl;
   return 0;
}
```

کنترل دسترسی به اعضای کلاس پایه

کلاس مشتق شده کلیه اعضای کلاس پایه را به ارث می برند اما اجازه دسترسی مستقیم به اعضای خصوصی کلاس پایه را ندارند و تنها از طریق توابع عمومی و سازنده به آنها دسترسی دارند.

نحوه دسترسی به اعضای عمومی کلاس پایه در کلاس مشتق شده توسط یکی مجوزهای دسترسی زیر که قبل از نام کلاس پایه ذکر می شود مشخص می شود:

- public
- private
- protected

توارث عمومی

با ذکر کلمه public قبل از نام کلاس پایه اعضای عمومی کلاس پایه به عنوان اعضای عمومی کلاس مشتق شده تلقی می شوند و در اختیار کاربر کلاس مشتق شده قرار می گیرد.

مثال. در مثال قبل تابع value از کلاس Base به اعضای عمومی Derived اضافه می شود بنابراین در برنامه قابل دسترسی است.

توارث خصوصی

با حذف کلمه public یا صریحا با ذکر کلمه private یک کلاس پایه می تواند به صورت خصوصی ارث گرفته شود. در توارث خصوصی کلاس مشتق شده کلیه اعضای کلاس پایه را دارا خواهد بود اما به صورت مخفی و اعضای عمومی کلاس پایه اعضای خصوصی کلاس مشتق شده خواهند شد. بنابراین یک شی به عنوان یک نمونه از کلاس نمی تواند به اعضای کلاس پایه دسترسی پیدا کند.

نکته. توارث خصوصی برای پنهان کردن لایه زیرین پیاده سازی کلاس پایه مفید است.
نکته. در توارث خصوصی کلیه اعضای عمومی کلاس پایه خصوصی می شوند. اگر می خواهید عضوی قابل رویت شود کافی است نام آن را (بدون آرگومان و مقدار برگشتی) در بخش public کلاس مشتق شده ذکر کنید.

مثال. چون وراثت خصوصی است تابع speak از کلاس پایه Pet در برنامه قابل دسترس نیست درحالیکه توابع eat و sleep از کلاس پایه به صورت قابل دسترس درآمده اند.

```
class Pet {
public:
   char eat() { return 'a'; }
   int speak() { return 2; }
   float sleep() { return 3.0; }
   float sleep(int) { return 4.0; }
};

class Goldfish : Pet {        // Private inheritance
public:
   Pet::eat;                   // Name publicizes member
   Pet::sleep;                 // Both overloaded members exposed
};

int main() {
   Goldfish bob;
   bob.eat();
   bob.sleep();
   bob.sleep(1);
//! bob.speak();              // Error: private member function
}
```

توارث محافظت شده

اعضای خصوصی همیشه خصوصی هستند اما گاهی می خواهید اعضائی را از خارج مخفی کنید ولی در کلاس مشتق شده قابل رویت باشند. کلمه protected می گوید که اعضای محافظت شده برای هر کسی که از این کلاس ارث می برد قابل دسترس است و برای بقیه خصوصی است.

مثال. توابع set و read از کلاس Base درمثال قبل در کلاس مشتق شده Derived قابل رویت هستند ولی در برنامه مخفی هستند.

با قرار دادن کلمه protected قبل از نام کلاس مشتق شده اعضای محافظت شده و عمومی کلاس پایه به اعضای محافظت شده کلاس مشتق شده اضافه خواهند شد. بنابراین برای وارثین کلاس مشتق شده در دسترس است و برای بقیه پنهان باقی می ماند

نکته. در کلیه حالات اعضای خصوصی کلاس پایه در وراثت شرکت نمی کنند و خصوصی باقی می مانند.
نکته. معمولا توارث عمومی است تا رابط کلاس پایه همچنان رابط کلاس مشتق شده باشد.
نکته. توارث محافظت شده خیلی استفاده نمی شود و فقط برای تکمیل زبان برنامه نویسی است.
نکته. مناسب ترین روش این است که اعضای داده ای کلاس را صورت خصوصی تعریف کنید تا امکان تغییر پیاده سازی زیرین حفظ شود. و به وارثین کلاس مجوز دسترسی کنترل شده ای به توابع عضو محافظت شده بدهید.

وارثت چندگانه

کلاس مشتق شده می تواند از بیش از یک کلاس پایه ارث بری داشته باشد. در توارث چندگانه (multiple inheritance) چند کلاس به عنوان کلاس پایه اضافه می شوند. نام کلاس های پایه توسط کاما از هم جدا می شود. توارث چند گانه مسائلی را پیش می کشد که در مبحث چندریختی توضیح داده خواهد شد.

```
class derived : access base1, access base2 ,.. {  //...
```

سازنده ها و مخرب ها در وراثت

کلاس پایه و مشتق شده هر یک می تواند شامل توابع سازنده و مشتق شده باشند. وقتی شیئی ایجاد می شود کامپایلر تضمین می کند که کلیه سازنده ها فراخوانی می شوند. در سلسله مراتب وراثت فراخوانی سازنده ها از ریشه شروع می شود. در هر سطح ابتدا سازنده کلاس پایه سپس سازنده کلاس مشتق شده فراخوانی می شود. مخرب ها بر عکس ترتیب فراخوانی سازنده ها فراخوانی می شوند.

مثال. در برنامه زیر کلاس Derived2 از کلاس Derived1 که خود از Base1 ارث بری دارد ارث می برد.

```
#include <iostream.h>

class Base1 {
   int x;
public:
   Base1 () {cout << "Base1 constructor\n";}
   ~Base1() {cout << "Base1 destructor\n";}
};

class Derived1 : public Base1 {
   int y;
public:
   Derived1() { cout << "Derived1 constructor\n";}
   ~Derived1() { cout << "Derived1 destructor\n";}
};

class Derived2 : public Derived1 {
   int z;
public:
   Derived2() { cout << "Derived2 constructor\n";}
   ~Derived2() { cout << "Derived2 destructor\n";}
};

int main() {
   Derived2 d2;
   return 0;
}
```

خروجی برنامه به صورت زیر است:

```
Base1 constructor
Derived1 constructor
Derived2 constructor
Derived2 destructor
Derived1 destructor
Base1 destructor
```

نکته. ترتیب فراخوانی سازنده ها و مخرب ها در در توارث چندگانه هم صدق می کند. سازنده ها به ترتیبی که در لیست مشخص شده اند از چپ به راست فراخوانی می شوند. و مخرب ها برعکس.

ارسال پارامتر به سازنده کلاس پایه

زمانی که تنها سازنده کلاس مشتق شده دارای آرگومان است می توان به سادگی و به صورت متعارف آرگومان را به سازنده ارسال نمود. اما برای ارسال آرگومان به سازنده کلاس پایه دچار مشکل می شوید چون سازنده کلاس مشتق شده به داده خصوصی کلاس پایه دسترسی ندارد و نمی تواند آنها را مقداردهی کند. برای این کار C++ گرامری را در اختیار می گذارد که لیست مقداردهی سازنده (constructor initializer list) نام دارد. لیست مقداردهی سازنده امکان فراخوانی صریح سازنده ها از اشیای عضو را می دهد. فرم کلی آن برای سازنده کلاس مشتق شده به صورت زیر است:

```
Derived(arg_list) : Base1(arg_list), Base2(arg_list), ...
{ //body of derived constructor}
```

نام کلاس های پایه توسط کاما از هم جدا می شوند. Base1 و Base2 و ... نام کلاس های پایه هستند که توسط کلاس مشتق شده Derived به ارث برده می شوند. سازنده ها همگی قبل از اینکه وارد بدنه سازنده کلاس مشتق شده شوید فراخوانی می شوند.

مثال. کلاس Circle از کلاس Point مشتق شده است. در سازنده کلاس Circle سازنده Coint فراخوانی می شود.

```
#include <iostream.h>

class Point {
   int x,y;
public:
   Point(int atx,int aty ) {x = atx; y = aty;}
   ~Point(){ cout << "Point Destructor called\n";}
   virtual void Draw() { cout << "Draw point at " << x << " " << y <<endl;}
};

class Circle : public Point {
   int radius;
public:
   Circle(int atx, int aty, int theRadius) ;
   ~Circle();
   virtual void Draw();
};

Circle::Circle(int atx,int aty,int theRadius) : Point(atx,aty) {
   radius = theRadius;
}

inline Circle::~Circle() {
   cout << "Circle Destructor called" << endl;
}

void Circle::Draw( void ) {
   Point::Draw();
   cout << "circle::Draw point " << " Radius "<< radius << endl;
}

int main() {
   Circle ACircle(10,10,5) ;
   ACircle.Draw();
   return 0;
}
```

برخلاف اینکه اغلب نیاز است سازنده ها به صورت مستقیم در لیست مقداردهی فراخوانی شوند، مخرب ها نیازی به فراخوانی صریح ندارند زیرا هر کلاس تنها یک مخرب بدون هیچ آرگومانی دارد. کامپایلر کلیه مخرب ها را از آخرین کلاس مشتق شده به سمت ریشه در کل سلسله مراتب وارثت اجرا می کند.