Item 36: Never redefine an inherited non-virtual function

فرض کنید که من به شما بگویم که یک کلاس به نام D به صورت عمومی از کلاسی به نام B مشتق شده است و یک تابع عضو عمومی در کلاس B وجود دارد که نام آن mf می‌باشد. پارامترها و تایپ خروجی mf برای ما مهم نیست، در این صورت اجازه بدهید که هر دو را به صورت void تعریف کنیم. به عبارت دیگر، من اینگونه می‌نویسم:

```
class B{
public:
   void mf();
};
class D:public B{

};
```

حتی اگر هیچ‌چیزی در مورد B و D و mf ندانیم، یک شیء به نام x از نوع D

```
D x;
```

احتمالا خیلی متعجب خواهید شد اگر بدانید که:

```
   B *pB=&x;   //get pointer to x
   pB->mf();   //call mf through pointer
```

با کد زیر زیر خیلی متفاوت خواهد بود:

```
   D *pD=&x;   //get pointer to x
   pD->mf();   //call mf through pointer
```

این به این دلیل است که در هر دو مورد شما تابع عضو mf را از طریق شیء x صدا می‌زنید. و به این دلیل که در هر دو مورد یک تابع را از طریق یک شیء صدا می‌زنیم، پس باید هر دو کد یکسان عمل کنند، درسته؟

در برخی موارد خاص، این دو کد یکسان کار نخواهند کرد، و آن وقتی است که mf به صورت non-virtual باشد و D ورژن خاص خودش را از mf داشته باشد.

```
class D:public B{
public:
   void mf();   //hides B::mf,see Item33
};
 pB->mf();  //calls B::mf
```

```
pD->mf();  //calls D::mf
```

دلیل این که دو رفتار متفاوت داریم این است که توابع non-virtual مثل B::mf و D::mf دارای محدودیت ایستا هستند(آیتم ۳۷ را ببینید). این بدین معنی است که از آنجایی که pB به عنوان یک اشاره‌گر به B اعلان شده است، توابع non-virtual ای که از طریق pB صدا زده شوند، همیشه آن‌هایی هستند که برای کلاس B تعریف شده است، حتی اگر pB اشاره‌گری باشد که به یک شیء از کلاس مشتق شده از B اشاره کند.

از سوی دیگر توابع virtual ، دارای محدودیت پویا هستند(آیتم ۳۷ را ببینید)، در نتیجه از این مشکل رنج نمی‌برند. اگر mf به صورت virtual بود، یک فراخوانی از طریق pB و یا pD منجر به فراخوانی D::mf می‌شود، چرا که چیزی که pB و pD واقعا به آن اشاره می‌کند یک شیء از نوع D می‌باشد.

اگر شما کلاس D را می‌نویسید و یک تابع non-virtual به نام mf را دوباره تعریف کنید که D از B ارث‌بری کرده، در این صورت D ممکن است که رفتار ثابتی نداشته باشد. به طور خاص، هر شیء D در هنگام فراخوانی mf شبیه به B و یا D کار خواهد کرد، و فاکتور تعیین کننده هیچ ربطی به خود شیء نداشته است، بلکه نوع اشاره‌گری که به آن اشاره می‌کند تعیین کننده خواهد بود. رفرنس‌ها هم رفتاری مشابه به پوینترها دارند.

ولی خب این بحث عملی است. می‌دانم که چیزی که شما واقعا می‌خواهید، این است که کمی در مورد تعوری این که چرا نباید توابع non-virtual که ارث‌برده شده را دوباره تعریف کنیم. من به خواسته‌ی شما احترام می‌گذارم.

آیتم ۳۲ توضیح داد که ارث‌بری عمومی یک قاعده‌ی is-a است، و آیتم ۳۴ توضیح داد که چرا اعلان توابع به صورت non-virtual در کلاس به معنای تعریف یک رفتار ثابت است. اگر شما این مشاهدات را روی کلاس B و D و تابع عضو non-virtual اعمال کنید، در نتیجه نتایج زیر را خواهیم گرفت:

- هر چیزی که بر شیء B اعمال شود، همچنین بر روی D نیز اعمال می‌شود، چرا که هر شیء D یک (is-a) شیء B است
- کلاس‌هایی که از B مشتق شده‌اند باید هم رابط و هم پیاده‌سازی mf را به ارث ببرند، چرا که mf یک تابع non-virtual در کلاس B است.

در این صورت، اگر D تابع mf را دوباره‌تعریف کند، یک تضاد در طراحی شما به وجود خواهد آمد. اگر D واقعا نیاز به این دارد که mf را متفاوت از B تعریف کند، و هر شیء B واقعا نیاز به پیاده‌سازی mf دارد، در این صورت به صورت ساده این که هر D یک B است درست نخواهد بود. در این مورد، D نباید به صورت عمومی از B به ارث برده شود. به عبارت دیگر، اگر D واقعا نیاز است که به صورت عمومی به ارث

برده شود، و D واقعا نیاز دارد که mf را متفاوت از B پیاده‌سازی کند، در این صورت این درست نیست که mf نشان‌دهنده‌ی یک رفتار ثابت است. در این مورد، mf باید به صورت virtual باشد. در نهایت، اگر هر D واقعا یک B باشد، و اگر mf واقعا تعریف کننده‌ی یک رفتار ثابت بر روی B است، در این صورت D نباید mf را دوباره تعریف کند.

اگر مطالعه‌ی این آیتم به شما این حس را داد که قبلا جایی این را دیده‌اید، احتمالا به خاطر این است که آیتم ۷ را مطالعه کرده‌اید، که توضیح داده بود که چرا مخرب‌ها در کلاس‌های base چندریختی باید به صورت virtual باشند. اگر شما این موضوع را نادیده بگیرید(یعنی، اگر یک مخرب non-virtual در یک کلاس base چندریختی، تعریف کنید)، چرا که کلاس‌های مشتق شده به صورت غیر قابل تغییر تابع non-virtual ارث‌برده شده را تعریف می‌کنند. این مورد حتی اگر کلاس‌های مشتق شده مخربی را اعلان نکنند نیز اتفاق می‌افتد، چرا که در آیتم ۵ دیدیم که مخرب چیزی است که کامپایلر به صورت خودکار آن را تعریف میکند.