Item 37: Never redefine a function's inherited default parameter value

اجازه بدهید که این بحث را بلافاصله شروع کنیم. دو نوع تابع وجود دارد که شما می‌توانید به ارث ببرید: توابع virtual و توابع non-virtual . اگر چه، دوباره نویسی یک تابع non-virtual اشتباه می‌باشد که در آیتم ۳۶ آن را بررسی کردیم، در این صورت می‌توانیم موضوع بحث را محدود به موقعیتی بکنیم که در آن یک تابع virtual با یک مقدار آرگومان پیش‌فرض به ارث‌برده شود.

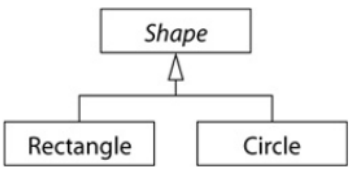
این موردی است که به بررسی آن خواهیم پرداخت، و رویکرد ما در این آیتم خیلی واضح خواهد بود: توابع virtual به صورت dynamically bound هستند، اما پارامتر پیش‌فرض به صورت statically bound هستند.

خب این یعنی چی؟ یک نوع شیء static نوعی است که شما آن را در متن برنامه اعلان می‌کنید. ساختار کلاس زیر را در نظر بگیرید:

```
//a class for geometric shapes
class Shape{
public:
   enum ShapeColor{Red,Green,Blue};

   //all shapes must offer a function to draw themselves
   virtual void draw(ShapeColor color=Red) const=0;
};
class Rectangle:public Shape{
public:
   //notice the different default parameter value---bad!
   virtual void draw(ShapeColor color=Green) const;
};
class Circle:public Shape{
public:
   virtual void draw(ShapeColor Color) const;
};
```

به صورت گرافیکی این ساختار شبیه به حالت زیر است:



حال اشاره‌گرهای زیر را در نظر بگیرید:

```
Shape *ps;                    //static type = Shape*
Shape *pc = new Circle;       //static type = shape*
Shape *pr = new Rectangle;    //static type = shape*
```

در مورد این مثال، ps و pc و pr همگی به صورت اشاره‌گر به Shape اعلان شده‌اند، بنابراین همگی به از نوع static هستند. توجه کنید که هیچ تفاوتی ندارد که واقعا این‌ها به چه چیزی اشاره می‌کنند(نوع استاتیک آن‌ها *Shape بوده).

نوع پویای شیء توسط شیء ای که الان به آن اشاره می‌شود، مشخص می‌شود. نوع پویای آن است که مشخص می‌کند که چطور رفتار کند. در مثال بالا، نوع پویای pc ، مشخصا *Circle می‌باشد و در مورد pr نوع پویا *Rectangle می‌باشد. و در مورد ps ، هیچ نوع پویایی وجود ندارد، چون به چیزی اشاره نمی‌کند(هنوز).

```
ps = pc;        //ps's dynamic type is now Circle*
ps = pr;        //pr's dynamic type is now Rectangle*
```

توابع virtual به صورت dynamically bound هستند، بدین معنی که یک تابع خاص از طریق نوع پویای شیء ای که صدا زده می‌شود، فراخوانی می‌شود:

```
pc->draw(Shape::Red);  //calls Circle::draw(Shape::Red)
pr->draw(Shape::Red);  //class Rectangle::draw(Shape::Red)
```

می‌دانم که این مفهوم چیزی است که قبلا نیز می‌دانستید. مشکل وقتی به وجود میآید که شما توابع virtual را با مقادیر پیش‌فرضشان در نظر بگیرید، چرا که همچنانکه در بالا گفته شد، توابع virtual به صورت dynamically bound هستند، ولی پارامترهای پیش‌فرض به صورت statically bound هستند. این بدین معنی است که شاید شما یک تابع virtual را در یک کلاس مشتق شده ولی با پارامتر پیش فرض از کلاس پایه صدا بزنید.

```
pr->draw();   // calls Rectangle::draw(Shape::Red) !
```

در این مورد، نوع پویای pr اشاره‌گر *Rectangle می‌باشد، بنابراین تابع مجازی Rectangle صدا زده می‌شود، که مورد انتظار ما هم هست. در Rectangle::draw ، پارامتر پیش‌فرض Green می‌باشد. از آنجایی که نوع استاتیک pr اشاره‌گر *Shape می‌باشد، مقدار پیش‌فرض برای این تابع از کلاس Shape گرفته می‌شود، نه از کلاس Rectangle .

این حقیقت که ps,pc و pr اشاره‌گر هستند، هیچ پیامدی ندارد. حتی اگر آن‌ها به صورت رفرنس بودند نیز این مشکل پیش میآمد. تنها چیز مهم این است که draw تابع مجازی می‌باشد، و یکی از پارامترهای پیش‌فرض آن در کلاس مشتق شده دوباره‌نویسی شده است.

چرا C++ به این صورت عمل می‌کند؟ پاسخ در پرفرمنس runtime می‌باشد. اگر پارامترهای پیش‌فرض به صورت dynamically bound بودند، کامپایلرها مجبور بودند که چندین روش ارایه دهند تا مقدار مناسب پیش‌فرض برای توابع virtual را در runtime مشخص کنند، که منجر به کاهش سرعت و افزایش پیچیدگی مکانیزم فعلی که در طی کامپایل مشخص می‌شود. این تصمیم گرفته شد تا سرعت و سادگی برنامه بالا باشد، و نتیجه‌ی آن است که شما از رفتار سریع برنامه‌های C++ لذت میبرید، اما اگر، به توصیه‌ی این آیتم گوش ندهید ممکن است دچار مشکل و سردرگمی شوید.

خب حال فرض کنید که بخواهیم این قاعده‌ای که در این آیتم گفتیم را رعایت کنیم و همچنین پارامترهای پیش‌فرض را هم برای کاربران هم در کلاس base و مشتق شده ارایه دهیم، نگاه کنید ببینید چه اتفاقی خواهد افتاد:

```
class Shape{
public:
   enum ShapeColor{Red,Green,Blue};

   //all shapes must offer a function to draw themselves
   virtual void draw(ShapeColor color=Red) const=0;
};
class Rectangle:public Shape{
public:
   //notice the different default parameter value---bad!
   virtual void draw(ShapeColor color=Red) const;
};
```

همانطور که می‌بینید code duplication داریم. بدتر این که code duplication همراه با وابستگی‌هایی نیز هست: اگر پارامتر پیش‌فرض در Shape تغییر بکند، همه‌ی کلاس‌های مشتق شده مجبورند که این تغییر را اعمال کنند. در غیر این صورت، قاعده‌ی این آیتم که جلوگیری از دوباره‌نویسی یک مقدار پیش‌فرض به ارث برده شده بود را نقض می‌کنند. چه کار باید کرد؟

وقتی که توابع virtual به صورتی که شما می‌خواهید عمل نمی‌کنند باید طراحی‌های جایگزین را نیز در نظر داشته باشید، و در آیتم ۳۵ ، ما جایگزین‌هایی را برای آن دیدیم. یکی از این جایگزین‌ها NVI بود: داشتن یک تابع عمومی non-virtual در کلاس base که یک تابع virtual خصوصی صدا بزند که

کلاس‌های مشتق شده نیز می‌توانند آن دوباره‌نویسی کنند. در اینجا، ما یک تابع non-virtual داریم که پارامترهای پیش‌فرض را مشخص می‌کند، در حالی که کار واقعی را تابع virtual انجام میدهد:

```
class Shape{
public:
   enum ShapeColor{Red,Green,Blue};
   void draw(ShapeColor color=Red) const              //now a non-virtual
   {
      doDraw(color);                                  //calls a virtual
   }

private:
   virtual void doDraw(ShapeColor color) const=0;   //the actual work is done in this func
};
class Rectangle:public Shape{
public:

private:
   virtual void doDraw(ShapeColor color) const;       //note! lack of default param
};
```

از آنجایی که توابع non-virtual هرگز توسط کلاس‌های مشتق شده، کنسل نمی‌شوند(آیتم ۳۶ را ببینید)، این طراحی به گونه‌ای است که پارامتر پیش‌فرض برای draw همیشه Red است.