بازنویسی عملگرها در C++ و C#

سادگی یا دورویی؟

امیر بها الدین سبط الشیخ

حتما تا به‌حال دقت کرده‌اید که مغز انسان چه پیچیدگی‌های عجیبی دارد. به‌سادگی قادر است چیزهایی را که هیچ ربطی به‌هم ندارند، به‌هم ربط دهد، تعداد و مقدار را به‌سادگی کنار هم قرار دهد یا حتی با هم جمع کند. زبان‌های شیءگرا[1] که با هدف پیاده‌سازی مدل طبیعی پا به عرصه وجود گذاشته‌اند نیز در اصل به‌دنبال ساده‌سازی کدنویسی و پیاده‌سازی هر چه بهتر دنیای واقعی هستند.

بیایید یک تلفنچی را مثال بزنیم؛ این شخص ممکن است به‌خاطر فشار شدید کاری در یک روزگرم تابستانی، به‌شدت از کوره در رفته باشد و حتی چند تماس را با بدخلقی تمام پاسخ داده باشد، اما وقتی یک تماس شخصی از سوی یکی از اعضای خانواده‌اش دریافت می‌کند، اخلاق خوشی از خود نشان می‌دهد و صورت دیگری از خود را بروز می‌دهد. این قابلیت، یعنی چندریختی[2] یکی از ویژگی‌های پیاده‌شده در برنامه‌نویسی شیءگرا است.

انواع مختلفی از چندریختی ممکن است وجود داشته باشد، یکی از آن‌ها که مثال زدیم، بازنویسی رفتار[3] است. یکی دیگر از انواع بازنویسی، بازنویسی عملگرها[4] است که به‌آن بیشتر خواهیم پرداخت.

هدف از بازنویسی عملگرها، تغییر رفتار عملگرهاست، تا جایی که عملگرها با توجه به نوع عملوندها رفتاری متفاوتی از خود نشان بدهند. برخی از این عملوندها را خود زبان برنامه‌نویسی بازنویسی کرده است، اما برخی دیگر را برنامه‌نویس به‌دلخواه خود بازنویسی می‌کند.

بگذارید یک مثال بزنیم تا بهتر مشخص شود چه کاری می‌خواهیم انجام بدهیم. عملگر جمع (+) را فرض کنید. در زبان سی‌شارپ اگر ۲عملوند عدد صحیح از نوع int باشد، خروجی int خواهد بود و ۲ عدد را جمع می‌کند و حاصل را برمی‌گرداند، حالا اگر ۲ عملوند از نوع رشته متنی باشند، چه؟

در اینجا زبان سی‌شارپ یک بازنویسی عملگر درونی دارد و درآن، ۲رشته ورودی را با هم ترکیب می‌کند و نتیجه به‌صورت یک رشته‌متنی باز می‌گردد، مثلا اگر بنویسیم: "Jamejam" + " Click!" خروجی عبارت Jamejam Click! خواهد بود، در ادامه نحوه بازنویسی عملگرها را برای کلاسی که خودمان تعریف کردیم بررسی می‌کنیم.

پیش از اینکه نحوه بازنویسی کردن عملگرها را توضیح دهیم، لازم است ببینیم کدام عملگرها قابلیت بازنویسی دارند:

عملگرهای دودویی (عملگرهایی که به دو عملوند (Operand) برای اجرا شدن نیاز دارند)، مانند: + (جمع)، - (تفریق)، *، /، %، &، |، ^، << و >>

عملگرهای یگانی (عملگرهایی که تنها به یک عملوند نیاز دارند)، مانند: true,false,++,--,~,!,-,+

بقیه عملگرها قابلیت باز نویسی ندارند ولی بعضی از آن‌ها را می‌توان به‌صورت دیگری بازنویسی کرد؛ مثلا عملگر =+ خلاصه عملگر + است، که خود + قابلیت بازنویسی را دارد یا عملگر () (تبدیل نوع یا cast)، قابلیت بازنویسی را ندارد، ولی شما با استفاده از explicit و implicit که هر دو از keywordهای سی‌شارپ هستند می‌توانید آن‌را بازنویسی کنید.

کار خود را با یک مثال شروع می‌کنیم. ما یک کلاس به اسم Matrix تعریف کردیم. این کلاس عملیات خاصی را روی یک ماتریکس انجام می‌دهد.

مثلا دترمینان یک ماتریکس را حساب می‌کند، یا ۲ماتریکس را جمع یا ضرب می‌کند و ... . حال فرض کنید بدون استفاده از بازنویسی عملگرها بخواهیم این کلاس را بنویسیم. برای جمع ۲ ماتریکس، ۲ روش وجود دارد، نخست این‌که یک متد استاتیک تعریف کنیم تا ۲ ماتریکس را جمع کند و حاصل را به‌صورت یک ماتریکس برگرداند. این متد به‌صورت زیر خواهد بود:

```
System.Math.Matrix mx1 =
new System.Math.Matrix(4,5);
System.Math.Matrix mx2 =
new System.Math.Matrix(4,5);
System.Math.Matrix result =
System.Math.Matrix.Add(mx1, mx2);
```

عکس: وردپرس

این روش اشتباه نیست، اما بگذارید کمی آن را بهتر بنویسیم. در کلاس ماتریکس متد Add را از حالت استاتیک در می‌آوریم، و به‌جای دریافت ۲ پارامتر ورودی، یک پارامتر ورودی به‌آن می‌دهیم و با خودش جمع می‌کنیم:

```
mx1.Add(mx2);
```

به‌هرحال این هم یک روش است که به‌نظر ساده‌تر و منطقی‌تر می‌رسد. حتی از این هم می‌توان زیباتر نوشت. اینجا همانجایی است که بازنویسی عملگرها وارد صحنه می‌شود:

```
mx1 = mx1 + mx2;
```

اگر بتوانیم ماتریکس‌ها را همان‌طور که روی کاغذ می‌نویسیم و جمع می‌زنیم، در کد خود بنویسیم، هم خواندن کد برای ما ساده‌تر می‌شود، هم درکار تیمی جلوه بهتری دارد.

ما چه کار خواهیم کرد؟ عملگر + (جمع) را برای کلاس ماتریس خودمان بازنویسی می‌کنیم و به‌کامپایلر نشان می‌دهیم که اگر ۲ عملوند عملگر + از جنس ماتریکس بود، فلان کار را انجام بده. که این فلان کار یعنی ۲ ماتریکس را با هم جمع کن و خروجی را از نوع ماتریکس برگردان. خوب این هم کار ما را خیلی ساده‌تر کرد، البته بیشتر برای کسانی که از کلاس ما استفاده می کنند ساده شده است، حال بررسی می‌کنیم ببینیم این عمل چگونه اتفاق می‌افتد یعنی کامپایلر از کجا می‌فهمد که باید هنگام برخورد با +، به‌سراغ متد خاصی برود وآن را اجرا کند؟ بیایید متدی را تعریف کنیم:

```
public class Matrix
  public static Matrix operator +
  (Matrix mx1, Matrix mx2)
  {return mx1.Add(mx2); }
```

همان‌طور که می‌بینید، در این جا ما عملگر + را به‌صورت عملگری دوگانه تعریف کرده که ۲ پارامتر mx1 و mx2 را دریافت می‌کند و mx1 را با mx2 جمع کرده و نتیجه را باز می‌گرداند (با استفاده از متد فرضی Add). و در نهایت کد بسیارساده‌ای خواهیم داشت. اما فراموش نکنید، استفاده از بازنویسی عملگرها، کد را به‌شدت ساده می‌کند، اما همین سادگی می‌تواند کار دست برنامه‌نویس بدهد. به‌فرض، اگر عملگر + را برای یک پارامتر از هر نوعی و یک پارامتر از نوع int بگیرید به‌خاطر تقدم و تاخر پارامترها، C+2 مجاز است، اما C+2 غیر مجاز است چون در ترتیب آرگومان‌ها، بازنویسی ندارد. این اشتباهات ساده ممکن است صدمات بزرگی به کد نوشته شده بزند وآن را کاملا غیرقابل اعتماد بکند.

پی‌نوشت‌ها

1. Object Oriented
2. Polymorphism
3. Method Overloading
4. Operator Overloading

تبدیل بی واسطه

دیگر گفتن این‌که کامپیوتر برمبنای اعداد دودویی فعالیت می‌کند و اعداد طبیعی دهدهی هستند، و باید مبنا را تغییر داد، به یک عادت برای هر استاد مبانی کامپیوتری تبدیل شده است. تبدیل مبناها نه‌تنها در دروس مبانی، بلکه در بیشتر واحدهای درسی، بخش غیرقابل درک کلاس به‌شمار می‌آیند. مبنای دو نوشتن و خواندن سختی دارد، از این رو به‌مبناهای دیگر افراز می‌شود، هگزادسیمال و اکتال، دو مبنای متداول برای راحت‌تر نوشتن و راحت‌تر خواندن اعداد دودویی به‌حساب می‌آیند.

تبدیل مبنای اکتال به هگزادسیمال از طریق تبدیل به یک مبنای میانی (مبنای ۲) و بعد دسته‌بندی به روشی دیگر به‌سادگی ممکن است، اما مساله جالبی می‌تواند این‌طور مطرح شود که چگونه بدون استفاده از مبنای واسط، اکتال را به هگزا دسیمال تبدیل کنیم؟

زیبایی حل این مساله در این است که مبناها، و تفکر در مبناهای مختلف برای ما که دسیمال می‌اندیشیم، ذهن را قلقلک می‌دهد. به‌طور کلی یک قانون وجود دارد که برای تبدیل مبنا از پایه پایین به مبنای ده از آن استفاده می‌شود:

$$abc_n = c \times n^0 + b \times n^1 + a \times n^2$$

حال می‌توانیم این قانون را تعمیم بدهیم، فرض بگیرید عدد اکتال ۱۷۳ را داریم که برابر با ۱۲۳ دسیمال و 7B هگزادسیمال است. اگر بخواهیم این عدد را به‌روش عادی به هگزا دسیمال تبدیل کنیم، باید ابتدا آن را با فرمول بالا محاسبه کنیم تا ۱۲۳ بدست بیاید، سپس تقسیم‌های ۱۶ متوالی داشته‌باشیم. می‌توانیم مستقیما بدون دخالت عدد دسیمال این کار را انجام دهیم، کافی است طور دیگری بیاندیشیم:

می‌دانیم که ۱۰ واحد هگزادسیمال، برابر است با ۲۰ واحد اکتال، در این صورت می‌توانیم با تقسیم متوالی به ۲۰، رقم به رقم اعداد هگزادسیمال را ایجاد کنیم. توجه داشته‌باشید که منظور ۲۰ واحد اکتال است که برابر با عدد ۱۶ دسیمال است. فرض بگیرید عدد ۱۷۳ را داریم:

۱۷۳ اکتال، تقسیم بر ۲۰ اکتال، می‌شود ۷ اکتال در خارج قسمت، بعلاوه ۱۳ اکتال در باقیمانده (که برابر است با B). توجه باید بکنید که ۱۷۳ تقسیم بر ۲۰، برابر با ۸ نخواهد شد، چرا؟ چون ۴ برابر ۲۰ اکتال، برابر با ۱۰۰ اکتال خواهد شد، نه ۸۰ اکتال. (در اکتال ۸ نداریم)