ایجاد رابط کاربری خط فرمان در برنامه‌های شی‌گرا

تو نلی به دنیای قدیم

امیربهاالدین سبط‌الشیخ

دریافت دستور در برنامه‌های مختلف، به‌شیوه‌های مختلف انجام می‌پذیرد. ممکن است از طریق صفحه نمایش چند لمسی دستور به برنامه وارد شود، ممکن است از طریق ماوس یا حتی ممکن است با تایپ دستور، بخواهیم برنامه را کنترل کنیم. کنترل برنامه‌ها با تایپ دستور، زمانی به‌کار می‌آید که دسترسی مستقیم یا دسترسی گرافیکی به برنامه نداریم و یا باید از طریق کنسول با برنامه خود ارتباط برقرار کنیم و پاسخ را دریافت کنیم. در این حالت، قابلیت اجرا از طریق خط فرمان یک مزیت برای برنامه‌مان به‌حساب خواهد آمد.

در این مقاله قصد نداریم به‌سراغ دستورهای پیچیده برویم و تلاش بر این است که با مفاهیم شیءگرای فعلی، یک پارسر برای خط فرمان بنویسیم. با بسط و گسترش همین کدها می‌توان هربرنامه‌ای را به موتور تشخیص دستور از راه خط فرمان مجهز کرد. نخست آن‌که وجه شباهت تمامی دستورها با همدیگر، این است که همه‌شان دارای پارامترهای ورودی هستند و همه آنها باید به‌شیوه‌ای صحیح اجرا شوند. به‌عبارت دیگر وقتی دستوری وارد می‌شود، باید تشخیص داده شده، اجرا شود و در صورت عدم شناسایی دستور، پیغام خطا ارسال کند. قدم نخست برای راه‌اندازی یک خط فرمان، تولید کلاسی پایه به‌صورت زیر است:

```
class baseCommand{
public :
baseCommand(int argc,char** argv);
baseCommand();
virtual void execute();
virtual bool contain(const char* arg);
virtual void printHelp();
private :
int argc;
char** argv;
}
```

این کلاس چیست و چه کاری قرار است انجام دهد؟

این کلاس تمامی متدها و فیلدهای مورد نیاز برای اجرا و نگهداری هر دستور را در خود دارد و به‌عبارت بهتر یک شیء از یک دستور است. بقیه دستورات از این کلاس، ارث‌بری می‌کنند و مستلزم بازنویسی متدهایی هستند که به‌صورت مجازی (Virtual) تعریف شده‌اند. این کلاس دو فیلد با دسترسی خصوصی (Private) دارد. دلیل این امر این است که اشیا دیگر نیازی به دانستن مقدار آنها و دیدن آنها از شی دیگر ندارند و در واقع این در این فیلدها مشخص می‌شود که چه دستوری چگونه باید اجرا بشود. فیلد اول argc تعداد آرگومان‌های ورودی هر دستور را مشخص می‌کند و فیلد argv یک آرایه دو بعدی از کاراکترها (آرایه‌ای از رشته‌ها) است که شامل دستور و پارامترهای ورودی آن دستور است.

این کلاس دو سازنده دارد که یکی از آنها مقدارهای argc و argv را از ورودی می‌گیرد و دیگری هیچ آرگومانی را نمی پذیرد و از آن به سازنده پیش فرض (Default Constructor) یاد می‌شود. متد execute دستور را اجرا می‌کند و وظیفه آن مدیریت اجرای صحیح دستور است. و متد دیگر printHelp است. این متد زمانی اجرا می‌شود که دستور با پارامتر /? اجرا شود و هدف آن نشان دادن اطلاعاتی در مورد دستور است. متد دیگر contain است این متد بررسی می‌کند آیا پارامتر خاصی در argv موجود است یا خیر.

بسیار خوب، حالا به یک فرمت کلی دست پیدا کردیم و می‌دانیم برای پیاده‌سازی هر دستور به چه چیزهایی نیاز است و هر دستور باید از چه الگویی باید پیروی کند. حالا می‌توان به‌شیوه‌ای دیگری به هر دستور نگاه کرد تا پیاده‌سازی راحت‌تری داشته باشد. هر دستور، یکسری پارامتر را به عنوان ورودی دریافت می‌کند. هر پارامتر برای انجام عمل خاصی است، پس در کلاس‌های مربوط به هر دستور علاوه بر دستورات فوق باید متدهایی برای انجام هر عمل نوشته شود. به‌طور مثال دستور Copy، این دستور باید دو پارامتر را به‌صورت پیش‌فرض دریافت کند. اولین پارامتر فایل منبع و دومی فایل مقصد را مشخص می‌کند. حال اگر دستور copy با فرمان /x اجرا شود باید فایل منبع به مقصد منتقل شود و کپی گرفته نشود. بنابراین، دو متد به نام‌های CopyFile و MoveFile را باید در کلاس مربوط به دستور copy بنویسیم که عملیات مربوطه را انجام دهند. مشکلی که در این میان ممکن است رخ بدهد، این است که متدهای MoveFile و CopyFile را باید به‌چه صورتی تبدیل کنیم که امنیت کافی داشته باشد؟ آیا این متدها private باشند یا public؟ از آنجا که این متدها برای کلاس مربوط به دستور copy هستند پس باید به‌صورت private تعریف شوند. اما اگر دستور دیگری داشتیم که نیاز به کپی و کات کردن فایل‌ها داشت نتیجه چه می‌شد؟ باید این متدها هر بار نوشته شوند؟ یا اینکه آنها را بصورت عمومی تعریف کنیم؟

اگر به‌صورت عمومی تعریف کنیم، طراحی ما غلط است چون دیگر کلاس‌ها نباید متدهایی را که برای اجرای پارامترهای ورودی هر دستور هستند ببینند. راه حل چیست؟ می‌توانیم با نوشتن یک کلاس جداگانه برای کارکردن با فایل‌ها، این مشکل را حل کنیم. این کار دو مزیت دارد:

۱- متدهای CopyFile و MoveFile از سطح کلاس دستورها جدا خوهد شد و بطور مستقل کار می‌کنند. این کار debug (اشکال زدایی) برنامه را راحت‌تر و نگهداری کد را ساده‌تر می‌کند.

عکس: photographycorner

در ضمن طراحی یکپارچه‌ای خواهیم داشت. بدین ترتیب کلاس File را برای کار با فایل‌ها به‌صورت زیر می‌نویسیم:

```
class File{
const static int BUF_SIZE = 4096;
public:
void Copy(char* source,char* destination);
static
void Move(char* source,char* destination);
static
static void Delete(char* filename);
Rename(char* oldname,char* newname);
static void
static bool Exist(char* filename);
void Create(char* filename,bool overwrite);
;static
}
```

نیازی به توضیح در مورد متدهای کلاس نیست، پس پیاده‌سازی متد execute از کلاس مربوط به دستور copy به صورت زیر خواهد شد:

```
void execute(){
->contain("/X") || this->contain("/x")){
if(this
File::Move(this->argv[1],this->argv[2]);
}
else
 ->argv[1],this->argv[2]);
File::Copy(this
}
```

حالا که موفق شدیم کلاس مربوط به Copy را پیاده کنیم به بررسی کلاس دیگری می پردازیم. این کلاس برای ایجاد رابطه با کاربر مورد استفاده قرار می‌گیرد. نام این کلاس را Helper می‌گذاریم، این کلاس قرار است دستوراتی را که کاربر وارد می‌کند، اعتبارسنجی کند و با توجه به دستورهای وارد شده، عملیات مناسب را اجرا کند. این کلاس بصورت زیر تعریف شده است :

```
class utility{
public :
utility(char* command);
utility();
void Run();
private :
char* command;
char** argv;
int argc;
baseCommand* internaleCommand;
void parse();
void createCommand();
};
```

این کلاس شامل ۳ فیلد است یکی argv ودیگری argc که قبلا درمورد آنها و کاربردشان صحبت کرده‌ایم و دیگری یک شی از baseCommand. همانطورکه گفتیم کلاس baseCommand کلاس پدر همه کلاس‌های مربوط به دستور است و مقدار آن می‌تواند تمامی دستورها را قبول کند، متد Run این کلاس در واقع دستوری که کاربر وارد کرده است، را تجزیه و تحلیل می‌کند و فرمان مربوطه را اجرا می‌کند. در این متد ابتدا عبارت وارد شده توسط متد parse تجزیه و تحلیل می‌شود و فیلدهای argv و argc مقدار دهی می‌شوند و سپس متد createCommand اجرا می‌شود و بر اساس مقدار argv فیلد internalCommand مقدار دهی می‌شود و سپس متد execute از کلاس baseComman اجرا می‌شود. با این کار دستور مورد نظر انجام شده و خروجی‌ها چاپ می‌شود. تا اینجا ما موفق شدیم دستوری را وارد و آن را اجرا کنیم. حال بگذارید شرایطی را بررسی کنیم که بخواهیم دستورها را پشت سر هم وارد کنیم و اینکار تا آنجا ادامه پیدا کند که کاربر دستور exit را وارد کند و از برنامه خارج شود (یعنی یک محیط خط فرمان برای کاربر ایجاد کنیم). وظیفه این کار بر عهده متد main یا متد اجرایی برنامه است.

با این پیاده‌سازی، یک محیط خوب برای برنامه خود دارید که می‌توانید در آن دستورهای مختلف را پیاده کنید. در این مقاله، دستورهای اجرایی، همان دستورهای سیستم عامل ویندوز بود. آیا می‌توانید توابع و کلاس‌های داخلی سیستم خود را با استفاده از این روش، راه‌اندازی و اجرا کنید؟ موفق باشید.