

ایجاد رابط کاربری خط فرمان در برنامه های شی گرا تونلی به دنیای قدیم

امیربهاالدین سبط الشیخ دریافت دستور در برنامه های مختلف، به شیوه های مختلف انجام می پذیرد. 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 برنامه ها با تایپ دستور، زمانی به کار می آید 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 برنامه مان به حساب خواهد آمد.

در این مقاله قصد نداریم به سراغ دستورهای پیچیده برویم و تلاش بر این است که با مفاهیم شیء گرای فعلی، یک پارسر برای خط فرمان بنویسیم. با بسط و گسترش همین کدها می توان هر برنامه ای را به موتور تشخیص دستور از راه خط فرمان مجهز کرد. نخست آن که وجه شباهت تمامی دستورها با همدیگر، این است که همه شان دارای پارامترهای ورودی هستند و همه آنها باید به شیوه ای صحیح اجرا شوند. به عبارت دیگر وقتی دستوری وارد می شود، باید تشخیص داده شده، اجرا شود و در صورت عدم شناسایی دستور، پیغام خطا ارسال کند. قدم نخست برای راه اندازی یک خط فرمان، تولید کلاسی پایه به صورت 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 به عبارت بهتر یک شیء از یک دستور است. بقیه دستورات از این کلاس، ارث بری می کنند و مستلزم بازنویسی متدهایی هستند که به صورت مجازی (Virtual) تعریف شده اند. این کلاس دو فیلد با دسترسی خصوصی (Private) دارد. دلیل این امر این است که اشیاء دیگر نیازی به دانستن مقدار آنها و دیدن آنها از شیء دیگر ندارند و در واقع این در این فیلدها مشخص می شود که چه دستوری چگونه باید اجرا بشود. فیلد اول argc تعداد آرگومان های ورودی هر دستور را مشخص می کند و فیلد argv یک آرایه دو بعدی از کاراکترها (آرایه ای از رشته ها) است که شامل دستور و پارامترهای ورودی آن دستور است.

این کلاس دوسازنده دارد که یکی از آنها مقدارهای argc و argv را از ورودی می گیرد و دیگری هیچ آرگومانی را نمی پذیرد و از آن به سازنده پیش فرض (Default Constructor) یاد می شود. متد

```
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 می گیرد. نام این کلاس را Helper می گذاریم، این کلاس قرار است دستوراتی را که کاربر وارد می کند، 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 و دیگری argc که قبلا در مورد آنها و کاربردشان صحبت کرده ایم و دیگری یک شیء از baseCommand. همانطور که گفتیم کلاس baseCommand کلاس پدر همه کلاس های مربوط به دستور است و مقدار آن می تواند تمامی دستورها را قبول کند، متد Run این کلاس در واقع دستوری که کاربر وارد کرده است، را تجزیه و تحلیل می کند و فرمان مربوطه را اجرا می کند. در این متد ابتدا عبارت وارد شده توسط متد parse تجزیه و تحلیل می شود و فیلدهای argv و argc مقدار دهی می شوند و سپس متد createCommand اجرا می شود و بر اساس مقدار argv فیلد internalCommand مقدار دهی می شود و سپس متد execute از کلاس baseComman اجرا می شود. با این کار دستور مورد نظر انجام شده و خروجی ها چاپ می شود. 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 کاربر عهده متد main یا متد اجرایی برنامه است.

با این پیاده سازی، یک محیط خوب برای برنامه خود دارید که می توانید در آن دستورهایی مختلف را پیاده کنید. در این مقاله، دستورهایی اجرایی، همان دستورهایی سیستم عامل ویندوز بود. آیا می توانید توابع و کلاس های داخلی سیستم خود را با استفاده از این روش، راه اندازی و اجرا کنید؟ موفق باشید.

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

۱- متدهای CopyFile و MoveFile از سطح کلاس دستورها جدا خواهد شد و بطور مستقل کار می کنند. این کار debug (اشکال زدایی) برنامه را راحت تر و نگهداری کد را ساده تر می کند.



عکس: photographycorner

در ضمن طراحی یکپارچه ای خواهیم داشت. بدین ترتیب کلاس File را برای کار با فایل ها به صورت زیر می نویسیم:

```
class File{
const static int BUF_SIZE = 4096;
public:
void Copy(char* source,char* destination);
static
void Move(char* source,char* destination);
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 پیاده سازی متد execute از کلاس مربوط به دستور copy به صورت زیر خواهد شد:
```

```
void execute(){
->contain("/X") || this->contain("/x")}{
if(this
```

execute دستور را اجرا می کند و وظیفه آن مدیریت اجرای صحیح دستور است. و متد دیگر printHelp است. این متد زمانی اجرا می شود که دستور با پارامتر /? اجرا شود و هدف آن نشان دادن اطلاعاتی در مورد دستور است. متد دیگر contain است این متد بررسی می کند آیا پارامتر خاصی در argv موجود است یا خیر. بسیار خوب، حالا به یک فرمت کلی دست پیدا کردیم و می دانیم برای پیاده سازی هر دستور به چه چیزهایی نیاز است و هر دستور باید از چه الگویی باید پیروی کند. حالا می توان به شیوه ای دیگری به هر

دستور نگاه کرد تا پیاده سازی راحت تری داشته باشد. هر دستور، یکسری پارامتر را به عنوان ورودی دریافت می کند. هر پارامتر برای انجام عمل خاصی است، پس در کلاس های مربوط به هر دستور علاوه بر دستورات فوق باید متدهایی برای انجام هر عمل نوشته شود. به طور مثال دستور Copy، این دستور باید دو پارامتر را به صورت پیش فرض دریافت کند. اولین پارامتر فایل منبع و دومی فایل مقصد را مشخص می کند. حال اگر دستور copy با فرمان /x اجرا شود باید فایل منبع به مقصد منتقل شود و کپی گرفته نشود. بنابراین، دو متد به نام های 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 به چه 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 به صورت private تعریف شوند. اما اگر دستور دیگری داشتیم که نیاز به کپی و کات کردن فایل ها داشت نتیجه چه می شد؟ باید این متدها هر بار نوشته شوند؟ یا اینکه آنها را بصورت عمومی تعریف کنیم؟

