File Manager

پیاده‌سازی فایل‌منیجر شبیه لینوکس (CLI)

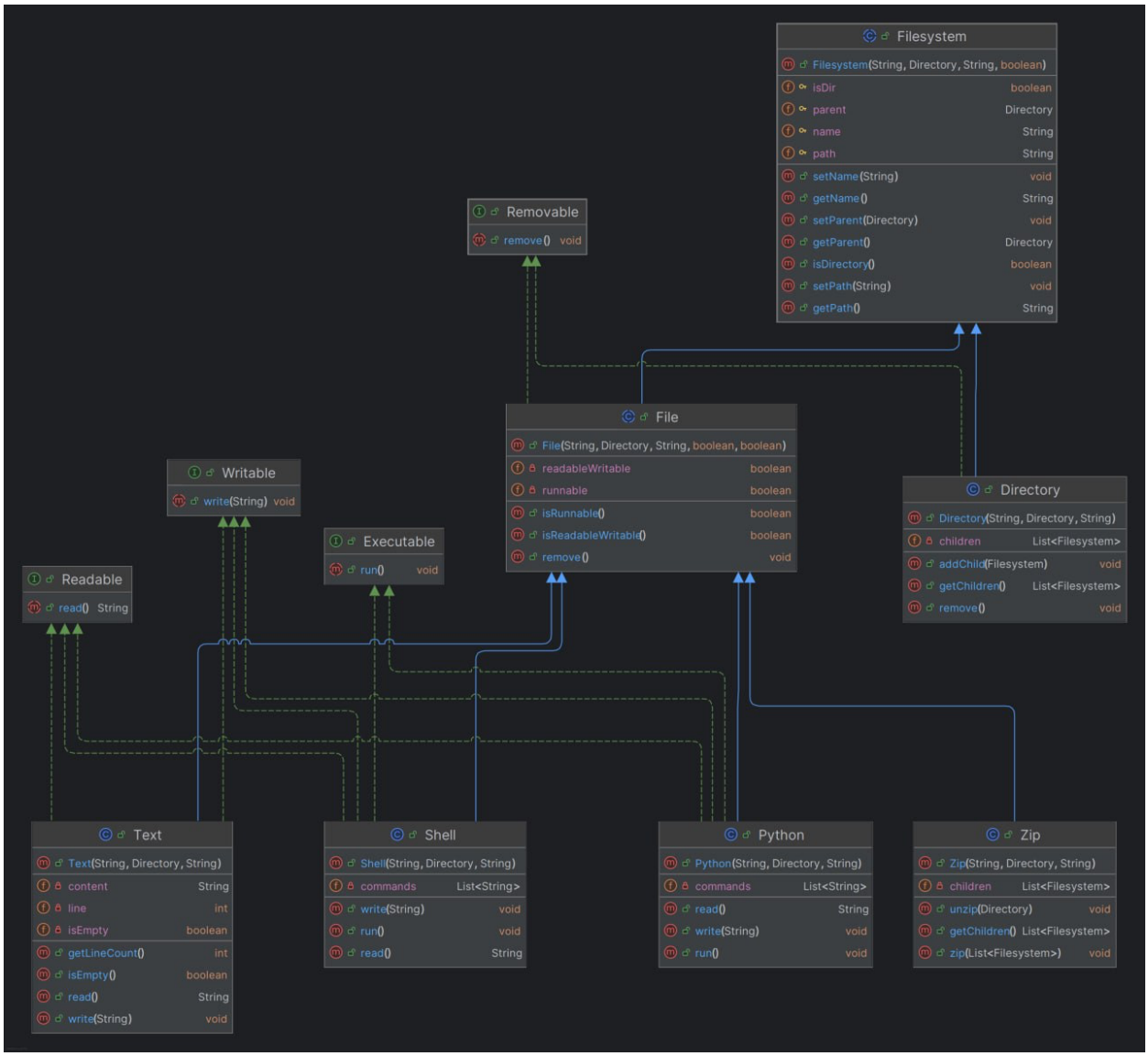
این تمرین با هدف تقویت مهارت‌های برنامه‌نویسی شیءگرا، طراحی شیءگرا، تست‌نویسی، و آشنایی عمیق‌تر با مفاهیم ارث‌بری، چندریختی، اینترفیس‌ها و ریفکتورینگ طراحی شده است. در پایان این تمرین، شما باید بتوانید یک فایل‌سیستم ساده در حافظه ایجاد کنید و مجموعه‌ای از دستورات مشابه لینوکس را روی آن اجرا نمایید.

1. شرح مسئله

شما باید یک فایل‌منیجر خط فرمانی (CLI) طراحی و پیاده‌سازی کنید که درون حافظه (in-memory) یک فایل‌سیستم درختی را نگه می‌دارد. کاربر می‌تواند با وارد کردن دستورات مختلف، فایل‌ها و دایرکتوری‌ها را ایجاد، جابجا، حذف و مشاهده کند. تمامی دستورات باید در قالب کلاس‌های جداگانه و با استفاده از اصول OOP پیاده‌سازی شوند.

نکته مهم: برنامه در هنگام شروع اجرا یک دایرکتوری ریشه ('/') می‌سازد و دایرکتوری جاری (Current Working Directory) روی همین ریشه تنظیم می‌شود. بنابراین خروجی اولین دستور pwd برابر '/' خواهد بود.

هر دستور باید یک کلاس مستقل داشته باشد و از یک اینترفیس مشترک مانند Command ارث‌بری کند. کد باید تمیز، قابل‌تست و قابل‌گسترش باشد.



در این تمرین، UML ارائه شده **حداقل ساختار لازم** برای پیاده‌سازی سیستم است و شما موظفید **تمام اجزای موجود در UML را دقیقاً مطابق آن** در کد خود پیاده‌سازی کنید.

به این معنی که:

- هر کلاسی که در UML، **abstract** مشخص شده است، باید در پیاده‌سازی نیز **abstract** باشد.
- هر کلاسی که از یک کلاس **ارث‌بری** دارد، باید **دقیقاً همان رابطه‌ی ارث‌بری** را در برنامه برقرار کند.
- هر کلاسی که یک **interface** را پیاده‌سازی می‌کند، باید همان اینترفیس را **implements** کند.
- نام کلاس‌ها، متدها و اینترفیس‌های موجود در UML باید **بدون تغییر** پیاده‌سازی شوند.

مجاز هستید در صورت نیاز، کلاس‌ها، متدها و فیلدهای بیشتری اضافه کنند؛ اما **نمی‌توانید اجزای موجود در UML را حذف، تغییر یا جایگزین** کنید.

به بیان دیگر، UML حداقل الزامات پروژه است و پیاده‌سازی شما باید *حداقل* همین ساختار را داشته باشد و می‌تواند با رعایت اصول شیءگرایی گسترش یابد.

2. دستورات مورد نیاز

دستورات که * دار هستند امتیازی محسوب میشوند.

- ls — نمایش لیست فایل‌ها/دایرکتوری‌ها در مسیر فعلی
  - * ls <path> — نمایش لیست فایل‌ها/دایرکتوری‌ها در مسیر مورد نظر
- cd <path> — تغییر دایرکتوری جاری
- pwd — نمایش آدرس دایرکتوری جاری
- mkdir <dir name> — ایجاد دایرکتوری در مسیر فعلی
- touch <file name> — ایجاد فایل در مسیر فعلی
  - *touch <file1> <file2> <file3> ... — ایجاد چند فایل همزمان

نکته: پسوندهای مجاز (zip, py, txt, sh) هستند.

- rm <file name> — حذف فایل
- rm -r <dir name> — حذف بازگشتی دایرکتوری
- cat <file name> — نمایش محتوای فایل
  - *cat <file name> -n — نمایش تعداد خطوط فایل

- echo <text> > <file name> — نوشتن محتوا داخل فایل

  echo <text> >> <file name> — چسباندن متن به انتهای فایل و رفتن به خط بعد

نکته: \n در txt به معنی خط بعد و در فایل های اجرایی به معنی دستور بعدی هست

- cp <src> <dest> — کپی فایل یا دایرکتوری
- mv <src> <dest> — جابجایی/تغییر نام

نکته: src اسم فایل یا دایرکتوری در مسیر فعلی

نکته : پیاده سازی مسیر dest تا یک لایه درونی یا بیرونی کافیست مثلا: mv example.txt ../example.txt

*پیاده سازی بیش ازیک لایه dest

- find <file or dir> — جستجو و نمایش مسیر فایل ها/دایرکتوری ها تا یک لایه از مسیر کنونی
  - *find -name "name" — جستجو و نمایش مسیر فایل ها/دایرکتوری هایی که در اسمشان عبارت جستجو شده را دارند
  - *find -empty — نمایش مسیر فایل ها/دایرکتوری های خالی
  - *find -type d — نمایش مسیر دایرکتوری ها
  - *find -type f — نمایش مسیر فایل ها *نکته :* جستجو تا یه لایه کافیست *جستجو بیش از یک لایه
- zip <zip_name> <file1> <file2> <file3> ... — زیپ کردن چند فایل و ساخت فایل زیپ در دایرکتوری فعلی
- unzip <file.zip> — باز کردن فایل های زیپ شده در دایرکتوری فعلی و حذف فایل زیپ
- *tree — نمایش درختی تمام مسیر ها از روت
- ./<exec_file> — اجرا به ترتیب دستورات درون فایل اجرایی

نکته : فایل شل (.sh)به ترتیب دستورات لینوکسی را اجرا میکند و فایل پایتون (.py) فقط دستور print("text") را اجرا میکند که محتوای text چاپ میشود.

خطا ها :

اگر در آرگمان دستورات فایل یا دایرکتوری وجود نداشت -> Error : No such file or directory

اگر مسیری وجود نداشت -> Error : Path does not exist

خطا : اگر دستوری معتبر نبود -> Error : Invalid command

خطا: ساخت یا انتقال فایل/دایرکتوری با نام مشابه -> Error: File or directory already exists

خطا: اگر در دستور touch پسوند فایل معتبر نبود -> Error: Invalid file type

خطا: اگر در دستور cat فایل قابل خواندن نبود -> Error: File is not Readable

خطا: اگر در دستور echo فایل قابل نوشتن نبود -> Error: File is not writable

خطا: اگر در دستور mv در آرگومان dest نوع فایل تغییر کرد -> Error: Cannot modify file type

خطا : اگر در اجار یک فایل اجرایی دستوری معتبر نبود (تا قبل آن دستور اجرا شود) -> Error : Invalid command

خطا : اگر فایل اجرایی نبود -> Error : Cannot execute [file_name]

3. نیازمندی‌های پیاده سازی

- سیستم گرفتن دستور و هدایت به تابع مربوطه
- نوشتن کلاس ها و فیلد و متود های مورد نیاز
- رعایت رابطه ارث بری
- کامل کردن توابع اینترفیس درون کلاس ها
- تابع بررسی معتبر بودن مسیر
- تابع PathResolver برای جدا سازی مسیر های نسبی
- **استفاده از Polymorphism و عدم استفاده از شرط های غیر ضروری**
- **رعایت اصول Clean Code**
- نوشتن Unit Test بری تست کارکد تمام بخش های سیستم (امتیازی)

4. نمونه ورودی/خروجی دستورات

▼ نمونه 1

```
fs> pwd
/

fs> mkdir home
fs> ls
home

fs> cd home
fs> pwd
/home

fs> touch notes.txt
fs> ls
notes.txt

fs> echo Hello > notes.txt
fs> cat notes.txt
Hello

fs> cp notes.txt copy.txt
fs> ls
notes.txt
copy.txt

fs> mkdir pooshe

fs> mv copy.txt pooshe/copy.txt
```

fs> find copy.txt

pooshe/copy.txt

fs> rm -r pooshe

fs> ls

notes.txt

▼ نمونه 2

fs> pwd

/

fs> touch note1.txt note2.txt script.sh

fs> mkdir folder1

fs> ls

note1.txt

note2.txt

script.sh

folder1

fs> echo “zip notes note1.txt note2.txt” > script.sh

fs> echo “mv notes.zip folder1/notes.zip” >> script.sh

fs> ./script.sh

```
fs> ls

script.sh

folder1

fs> cd folder1

fs> pwd

/folder1

fs> ls

notes.zip

fs> unzip notes.zip

fs> ls

note1.txt

note2.txt

fs> touch main.py

fs> echo “print(“hello world”)\nif(a > 1)\nprint” > main.py

fs> cat main.py

print(“hello world”)

if(a > 1)

print
```

```
fs> ./main.py

hello world

Error: Invalid command

fs> tree

├─── folder1
|   ├─── note1.txt
|   ├─── note2.txt
|   └─── main.py
└─── script.sh
```

5. تحویل‌دادنی‌ها

- کد کامل پروژه
- تست‌های JUnit (امتیازی)