Design Pattern

최종 보고서

컴퓨터 전자시스템공학부
202004520 최준혁

프로젝트명: VirusTotal CLI Application

2022-1학기 디자인패턴

2022. 06. 13

1. 프로젝트 개요

1) 프로젝트 소개

VirusTotal은 Google의 자회사로써 파일/URL 등을 업로드하면, 여러 종류의 백신들 (Avast, Kaspersky, BitDefender 등... 심지어는 알약/V3도 지원한다.)로 검사를 돌린 후, 그 결과를 알려주는 서비스를 제공한다.

기본적으로는 웹 애플리케이션으로 이용이 가능하고, API가 공개되어 있다.

이 VirusTotal 서비스를 이용하기 위해서는 웹 브라우저를 이용해야 한다. 즉, 리눅스 등 CLI 환경에서는 이용하기가 어렵다는 소리이다.

따라서, CLI 환경에서도 VirusTotal을 이용 가능하도록 하는 애플리케이션을 구현하고자 한다.

2. 목표 및 내용

1) 최종 목표

VirusTotal CLI Application의 구현을 목표로 한다.

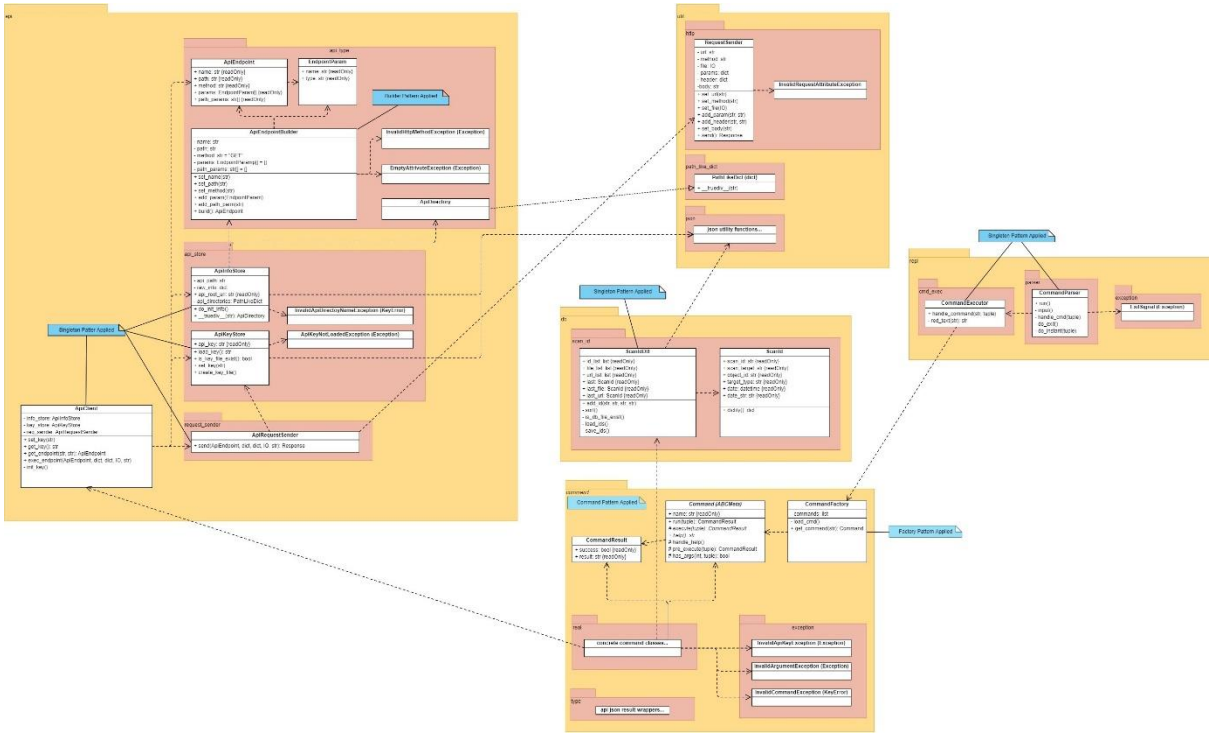
- REPL 구현
  - CLI 환경에서 이용하기 위함.
- API Wrapper 구현
  - VirusTotal의 기능을 이용하기 위함
  - VirusTotal API를 Python 코드상으로 이식

2) 적용된 패턴의 종류 및 방법

- Singleton Pattern
  - REPL 파트의 클래스들과 Store류 클래스들은 복수의 인스턴스가 필요하지 않음.
  - 따라서 Singleton 패턴을 적용하여 리소스 낭비를 줄임
- Command Pattern
  - REPL 파트와 API Wrapper 파트 사이의 종속성을 방지하기 위해 Command 패턴을 적용함
- Builder Pattern
  - API Endpoint 정보에는 많은 속성이 존재함
  - 이를 Data Object로 만들어 줄 때, 속성들을 효율적으로 넣어주기 위해 Builder 패턴을 적용함
- Factory Pattern
  - Command 클래스의 인스턴스를 쉽게 생성하기 위해 Factory 패턴을 적용함

3) 주요 코드 및 설명

Class Diagram



[이미지 원본 링크]

코드는 아래와 같이 크게 두 파트로 나눌 수 있다.

1. REPL
2. API Wrapper

우선 REPL 파트이다.

repl. CommandParser 클래스 내부

```
def __input(self):
    s = input("VT>> ")
    lo = s.lower()

    if lo == "exit" or lo == "stop" or lo == "quit" or lo ==
"bye" or lo == "close":
        return self.__do_exit()

    args = s.split(" ")
    self.__handle_cmd(*args)

def __handle_cmd(self, *args):
    cmd_name = args[0]
    cmd_args = args[1:]
    CommandExecutor().handle_command(cmd_name, *cmd_args)
```

위와 같이 input 메소드와 handle_cmd 메소드를 통해서 명령문을 받아들이고, Argument로 나눠서 repl.CommandExecutor 클래스로 Argument를 전달해주는 모습이다.

repl.CommandExecutor 클래스 내부 handle_command 메소드 내부

```
target_cmd = self.__cmd_fac.get_command(cmd_name)
if target_cmd is None:
    raise KeyError
result = target_cmd.run(*args)
if not result.success:
    print(self.__red_text(result.result))
else:
    print(result.result)
```

cmd_fac 내부에는 command.CommandFactory 인스턴스가 들어가 있다.

CommandFactory 클래스의 get_command 메소드로 커맨드를 가져와 실행시키는 모습이다.

다음은 API Wrapper 부분이다.

api.api_store.ApiInfoStore 클래스의 do_init_info 메소드를 통해서 resource/api-info.json 파일을 불러와 인스턴스 내부에 저장한다.

```
@property
def root_url(self) -> str:
    return self.__api_root_url

def __truediv__(self, other: str) -> ApiDirectory:
    directory = self.__api_directories[other]
    if directory is None:
        raise InvalidApiDirectoryNameException
    return directory
```

ApiInfoStore에서 구현한 __truediv__ 메소드를 통해서 ApiEndpoint에 접근할 수 있도록 하였다.

root_url Setter를 구현하여 API의 Endpoint Root URL에도 접근 가능하다.

위 두 정보를 조합하면 API의 Endpoint에 접근 가능한 정보를 모두 취득 가능하다.

api.ApiClient 클래스의 내부

```
def get_endpoint(self, dir_name: str, endpoint_name: str) ->
ApiEndpoint:
    return self.__info_store / dir_name / endpoint_name
```

get_endpoint 메소드를 통해서 ApiInfoStore에 저장된 Endpoint 정보를 가져올 수 있다.

```
def exec_endpoint(self, endpoint: ApiEndpoint,
                  params: dict = None,
                  path_params: dict = None,
                  file: IO = None,
                  body: str = None):
    return self.__req_sender.send(endpoint, params,
path_params, file, body)
```

이후 exec_endpoint에 get_endpoint로 가져온 ApiEndpoint를 다시 넣어주고, 여러 파라미터들을 넣어주면 api.request_sender.ApiRequestSender 클래스의 send 메소드를 호출하여 VirusTotal API와 통신한다.

이 REPL과 API Wrapper 사이의 의존성을 방지하기 위해, Command Pattern을 적용하였다.

command.Command 클래스 내부

```
def run(self, *args) -> CommandResult:
    pre = self._pre_execute(*args)
    if pre is not None:
        return pre
    try:
        return self._execute(*args)
    except InvalidArgumentException:
        return CommandResult(False, "Error: Invalid Argument.
type ")

@abstractmethod
def _execute(self, *args) -> CommandResult:
    pass
```

execute 메소드를 구현하면, run 메소드 내부에서 execute 메소드를 실행하여 커맨드를 실행된다.

이 run 메소드는 repl.CommandExecutor의 handle_command 메소드를 통해 실행된다.

Work Flow는 아래와 같다.

1. repl.CommandParser 클래스가 명령문을 입력받고, 명령어와 인자를 나눔.

2. repl.CommandExecutor 클래스가 명령어와 인자를 받고, command.CommandFactory로부터 Command 클래스를 구현한 커맨드 (이하 Concrete Command)를 받아와서 실행.

3. Concrete Command가 db.ScanIdDB에서 필요한 정보를 가져온 후, api.ApiClient를 이용하여 VirusTotal API에 Request를 보냄.

4. VirusTotal API로부터 Response를 받고, Concrete Command가 Response를 통해 Business Logic 실행 후 그 결과 반환.

5. 결과를 받아온 CommandExecutor가 그 결과를 화면에 출력.

3. 결과 및 느낀점

1) 결과 화면 및 설명

1. 첫 실행

```
 ~#@〉  python main.py
VirusTotal CLI /w Python
Type help for Command List.
Type exit for Close Application.
VT>>
```

실행하면 REPL이 시작되면서 명령문을 입력할 수 있다.

2. help

```
VT>> help
Command: file-report -> Show Your File Report / Usage: report [last | {FILE_ID}] [verbose]
Command: file-scan -> Upload File and Request File Scan / Usage: file-scan {FILE_PATH}
Command: help -> Show Command List / Usage: help [{COMMAND_NAME}]
Command: id -> Get Your Scan IDs / Usage: id [url | file] Or id last [url | file]
Command: key -> Set API Key or Get API Key / Usage: key [help | get | set {API_KEY}]
Command: url-report -> Show Your URL Report / Usage: url [last | {URL_ID}] [verbose]
Command: url-scan -> Scan URL / Usage: url-scan {URL}
```

help 커맨드를 통해 모든 커맨드 목록을 가져올 수 있다.

3. scan

```
VT>> file-scan main.py
Uploading File...
Fetching File ID...
Successfully Upload File!
File Path: D:\Study\university\HUFS.DesignPattern.TermProj\main.py
Scan ID: MWIwYWZlNzNkNzBiNDJiMzA5MzUzMTAyZDFjNWE4ZmE6MTY1NTEyNTAzMg==
File ID: c01b32f4c91749aa53a9a84077e129d8107601d0938f05910b3e5e292c7dde14
Type "file-report last" for Result.
```

file-scan/url-scan을 통해 파일/URL을 업로드 할 수 있다.

업로드 한 파일/URL의 검사 결과는 file-report/url-report 커맨드를 통해 확인할 수 있다.

4. report

```
VT>> file-report last
Fetching Report Data...
File: main.py (java / Java) Size: 111B
Undetected: 56
Harmless: 0
Malicious: 0
Suspicious: 0
Failure: 0
Timeout: 0
Type Unsupported: 15
```

file-report/url-report를 통해 파일/URL의 검사 결과를 확인할 수 있다.

```
Scan Results
Bkav (1.3.0.9899) : undetected
Lionic (7.5) : undetected
tehtris (v0.1.4) : type-unsupported
DrWeb (7.0.56.4040) : undetected
ClamAV (0.105.0.0) : undetected
CMC (2.10.2019.1) : undetected
CAT-QuickHeal (14.00) : undetected
McAfee (6.0.6.653) : undetected
Malwarebytes (4.3.3.37) : undetected
```

verbose 인자를 추가해서 위와 같이 각 엔진별 검사 결과도 확인 가능하다.

추가로, key 커맨드를 통해 API Key의 변경이 가능하고,

exit를 통해 프로그램을 종료할 수 있다.

2) 느낀 점

이전에도 Java를 이용하면서 SOLID Principle이나 몇몇 디자인 패턴은 알고 있었는데, 설계패턴 수업과 텀 프로젝트를 진행하며 복습도 되고, 이전에 몰랐던 여러 패턴들도 새로 알게 되는 좋은 기회가 되었다.

이전에는 Java, Javascript, Typescript에서만 OOP를 적극 활용한 프로그래밍을 해왔는데, 텀 프로젝트를 통해 패턴을 적용하며 이번에는 Python으로 코딩을 하는 경험을 할 수 있게 되어 새롭고 좋았다.

4. 추가 링크

Source Sode Github: https://github.com/jhchoi123/HUFS.DesignPattern.TermProj

Class Diagram Image:
https://github.com/jhchoi123/HUFS.DesignPattern.TermProj/blob/master/document/class_diagram_final.jpg