

# SSD 프로젝트

- SSD 검증 애플리케이션



Dooly 팀

김희준, 박혜록, 송주환, 오지은, 윤다영, 정동혁, 한누리



## 조원 소개 및 역할



# 조원 소개 및 역할

## 박혜록 (Shell)

- 기본 동작 구현
- 주요 기능 개발
- Shell, logger 리팩토링

## 정동혁 (Shell)

- 주요 기능 개발
- logger 연동 개발 / 테스트
- SSD 리팩토링

## 윤다영 (Shell)

- 유닛 테스트 구현
- Test script 구현
- runner 구현

## 김희준 (SSD)

- SSD 클래스 개발
  - write
  - main 함수
  - 입력값 validation 체크 개발
- Buffer manager / 테스트

## 오지은 (SSD)

- SSD 클래스 테스트 함수 개발
- file manager / 테스트 함수 개발
- buffer 실제 동작 구현 및 연동 테스트

## 한누리 (SSD)

- SSD 유닛 테스트
- buffer manager 연동 테스트 구현
- Logger 구현
- buffer 리팩토링



## 송주환 (Shell, SSD)

- 전체 통합 테스트 및 버그 리포트
- buffer, ssd 유닛 테스트 개발

# 기능 구현 소개

```
Shell> help

usage: <command> [<args>]

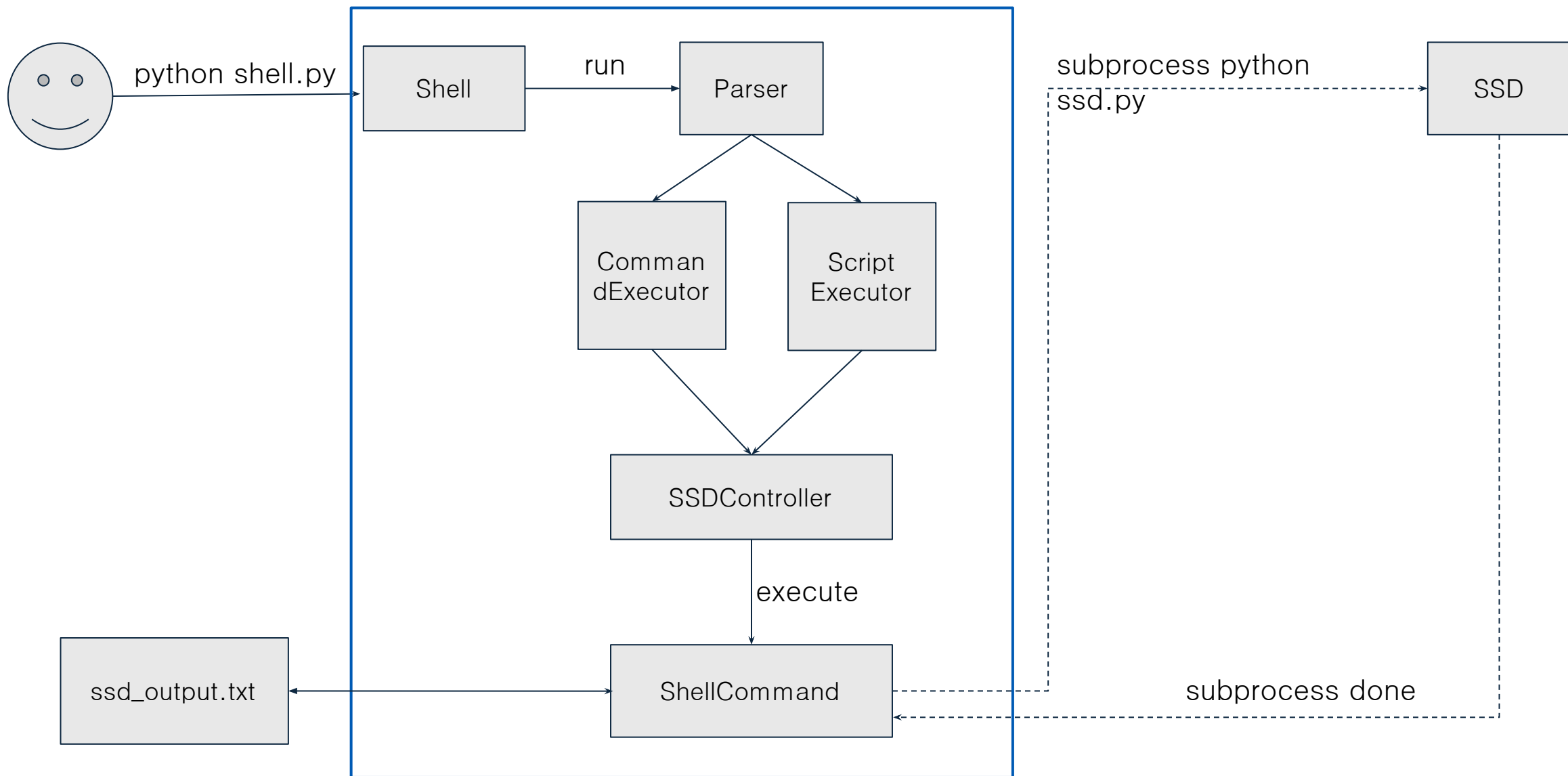
      />  フ
      |  _ _|
      /`  ≡_xノ
      /      |
      /  `   )
      |  |  |  |
      /┐|  |  |  |
      (┐`_ _`_)_)
      \二) ... Dooly Let`s go !!!

read: read a value from the given LBA
```

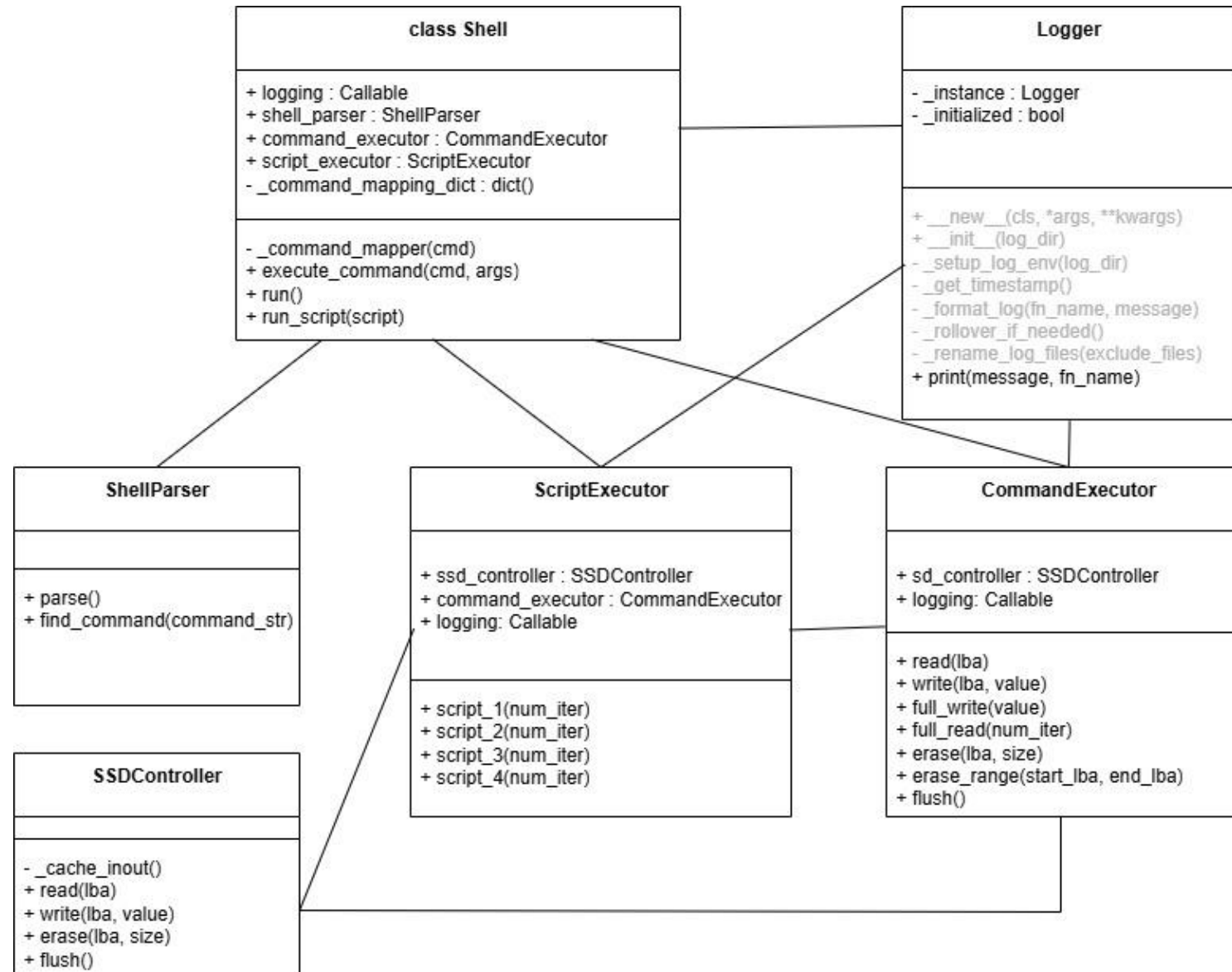
Dooly Team. 돌리보다 더 귀엽게 코드리뷰한다.

진짜 귀엽게 함

# Shell 동작 흐름 및 구조



# Shell Class Diagram



# SSDController Command Pattern

구현 장점:

① 단일 책임

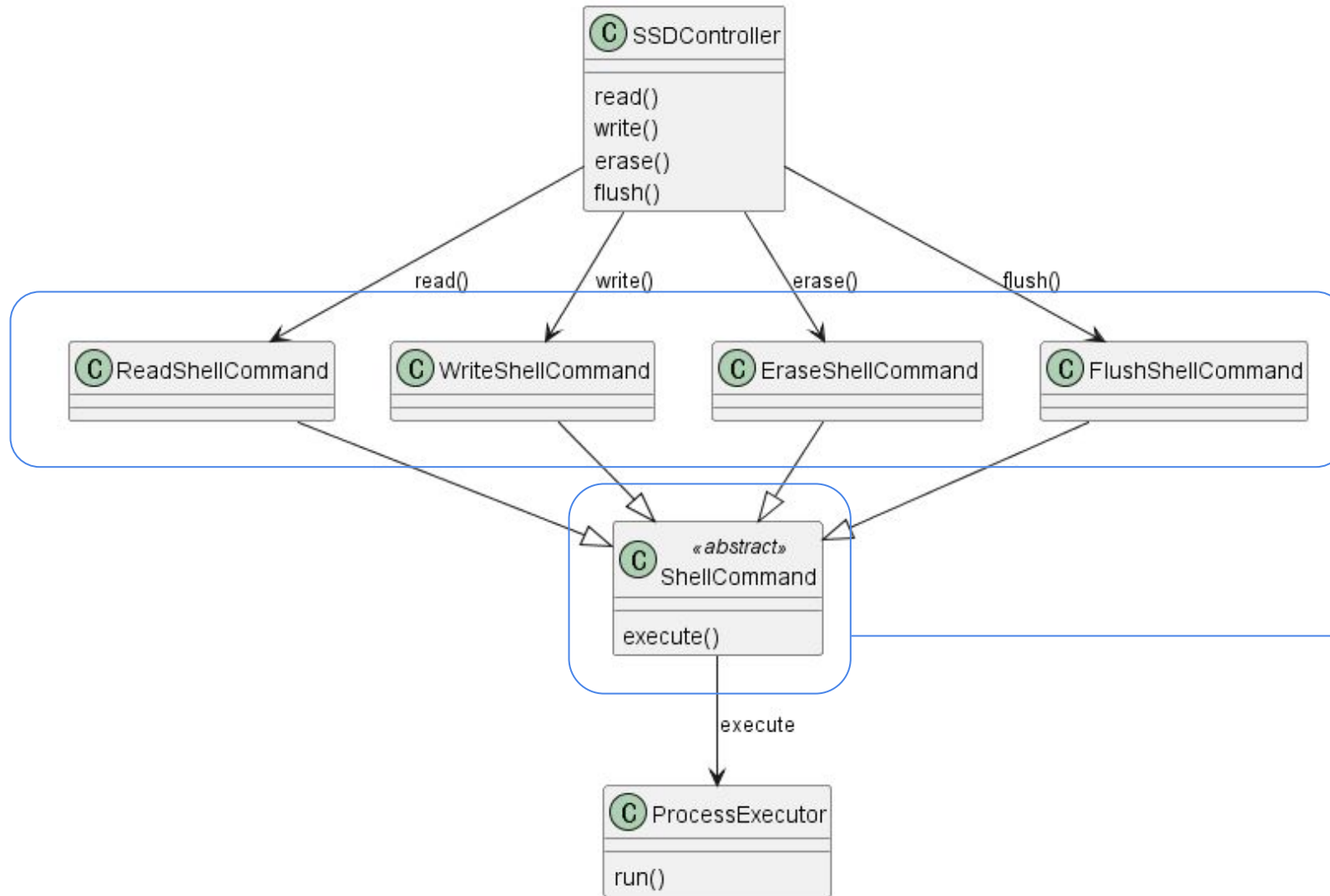
하나의 커맨드가 하나의 작업 담당  
ProcessExecutor가 subprocess 실행 담당

② 확장성

새로운 SSD 명령어 추가시 새 커맨드 클래스만 추가하면 됨.

③ 인터페이스

ShellCommand(ABC)가  
execute(Abstract Method)로  
일관된 subprocess 실행 인터페이스 제공



▲ SSDController 클래스 다이어그램

# Logger Singleton Pattern

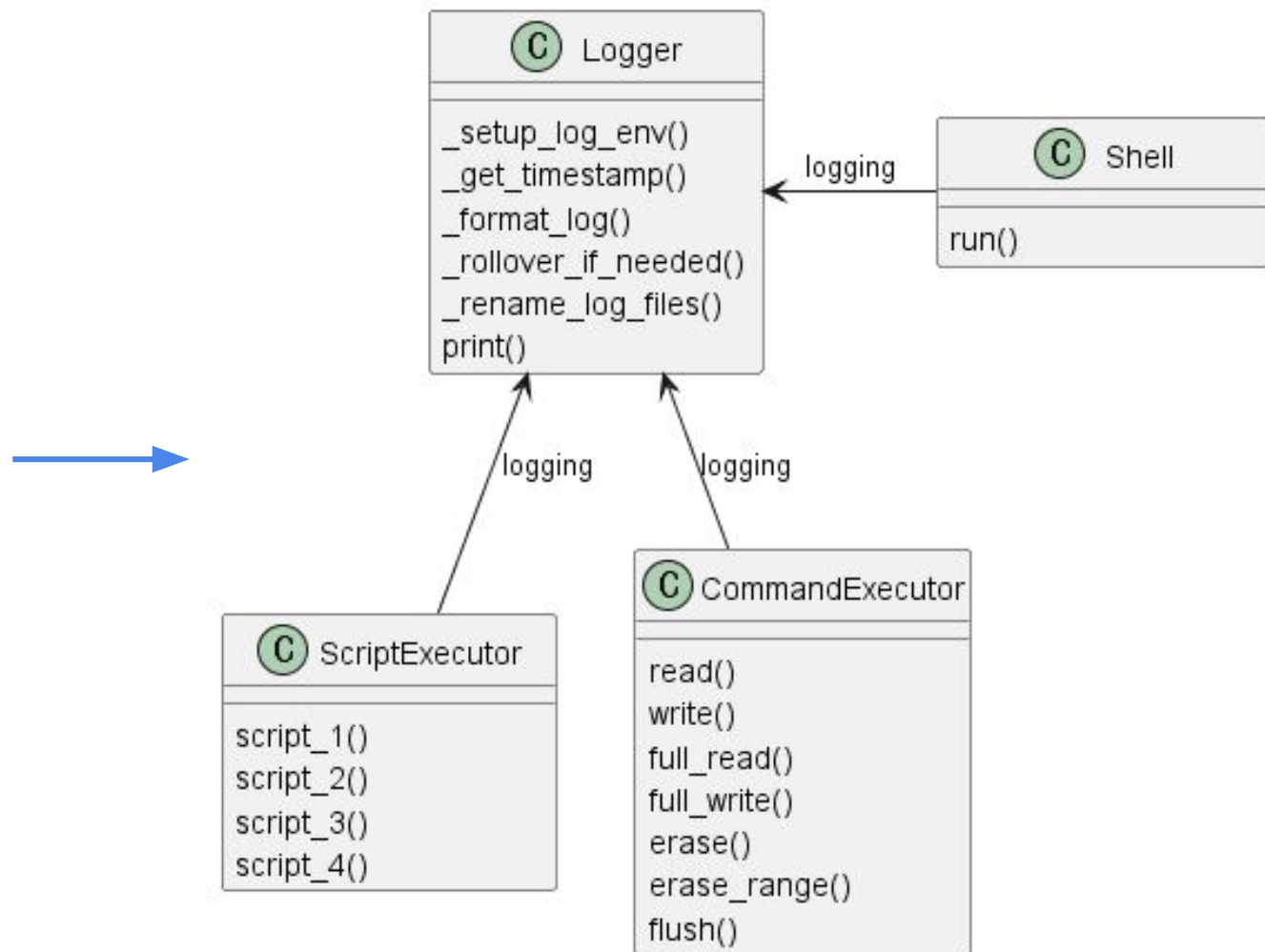
## 기존 문제점

### ① 중복 인스턴스 생성

Shell, ScriptExecutor,  
CommandExecutor  
각각에 생성됨

### ② Logging 환경 생성 및 체크

Logging 폴더, 파일 생성과 체크  
반복됨





# Logger Singleton Pattern

```
class Logger:
    _instance: Logger | None = None
    _initialized: bool = False

    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = super().__new__(cls)
        return cls._instance

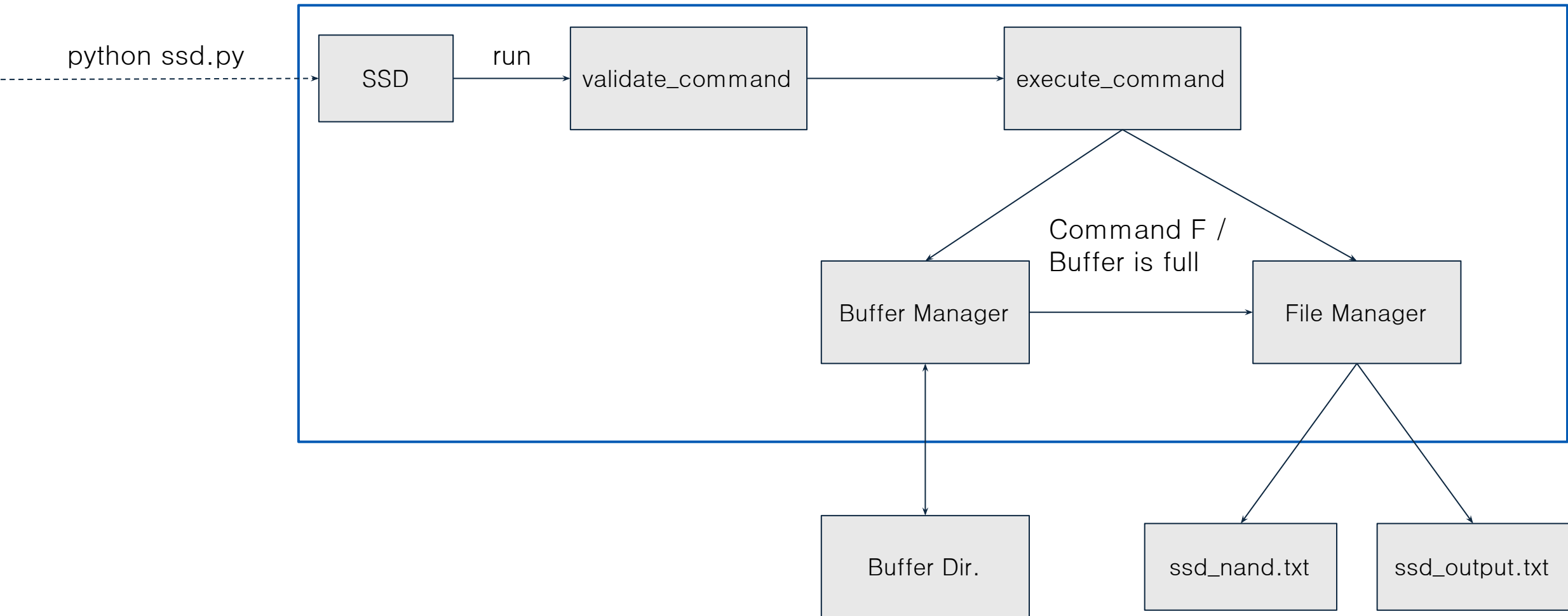
    def __init__(self, log_dir: str = "./log"):
        if self._initialized:
            return

        self._setup_log_env(log_dir)
        self._initialized = True
```

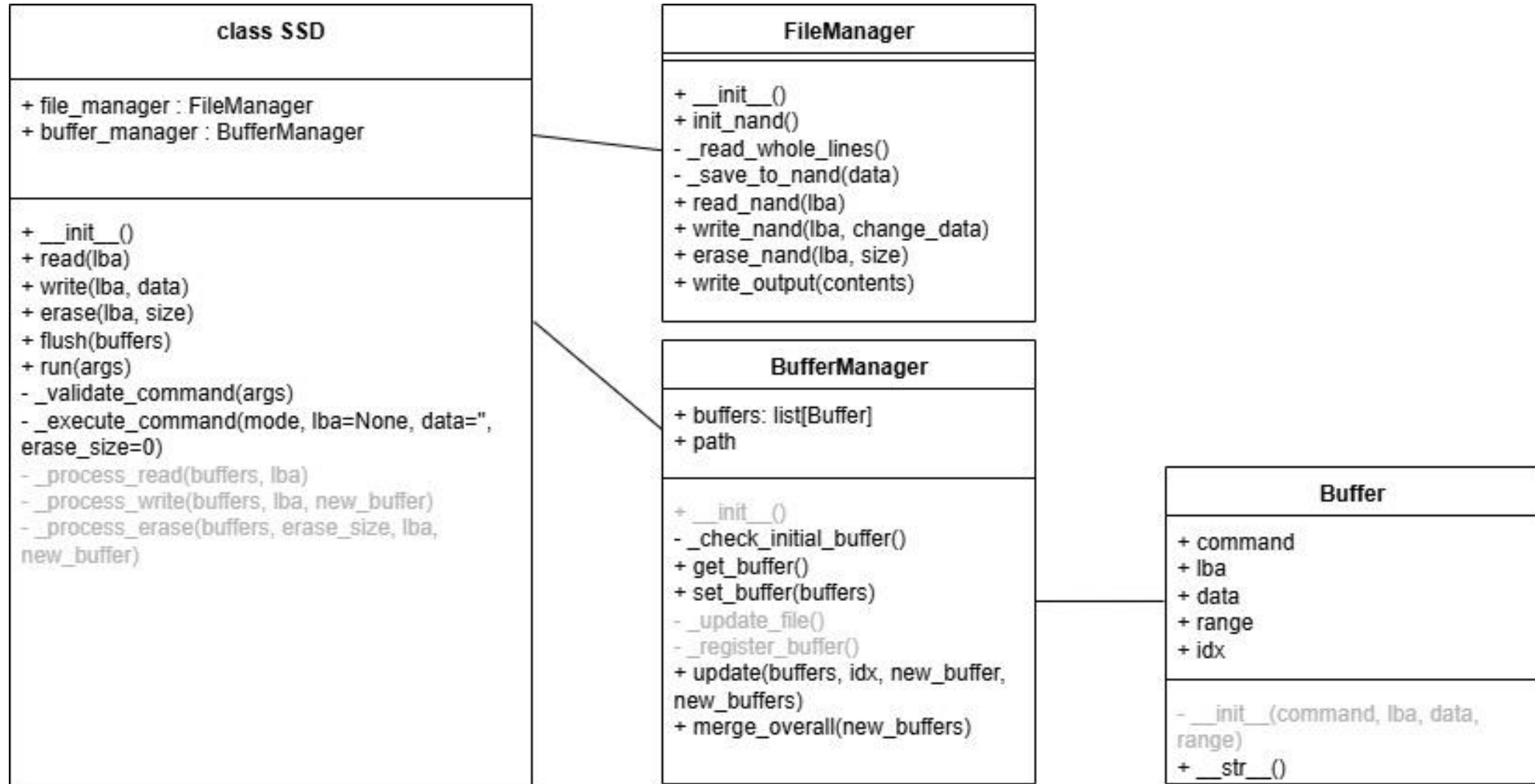
- Shell, ScriptExecutor, CommandExecutor에서 **하나의 Logger 인스턴스** 만을 사용해 로깅할 수 있는 구조

▲ Singleton 반영된 Logger

# SSD 동작 흐름 및 구조



# SSD Class Diagram





# SSD 프로젝트 시연





TDD 활용



# TDD – Coverage

Coverage report: 88%

Files

Functions

Classes

coverage.py v7.9.2, created at 2025-07-18 10:32 +0900

File ▲	statements	missing	excluded	coverage
buffer_manager.py	94	3	0	97%
commands.py	43	3	0	93%
constant.py	31	0	0	100%
logger.py	62	3	0	95%
shell.py	272	42	0	85%
ssd.py	250	37	0	85%
utils.py	65	14	0	78%
<b>Total</b>	<b>817</b>	<b>102</b>	<b>0</b>	<b>88%</b>

coverage.py v7.9.2, created at 2025-07-18 10:32 +0900

## 1-2일차

- Test Case 생성
- 해당 Test Case Pass를 위한 구현
- Coverage 확보한 뒤 리팩토링 및 추가

## 3-4일차

- Corner case test 통한 안정성 확보

tests

- | test\_buffer.py => command buffer 테스트 구현
- | test\_logger.py => logger 테스트 구현
- | test\_shell.py => shell 테스트 구현
- | test\_ssd.py => ssd 테스트 구현

▲ Final Coverage on Project

# TDD – SSD

## 작은 단계별로 일반화된 솔루션을 찾기

- 기본 기능 테스트 먼저 구현 후 test 통과하도록 기능 구현
- 이후 corner case 추가로 기능 안정성 향상

```
10 @pytest.fixture
11 > def ssd(mock):...
15
16 > def test_write_and_read(ssd):...
25
26 > def test_read_ssd_nand_txt_file_called_by_read(ssd):...
32
33 > def test_read_ssd_nand_txt_file_called_by_write(ssd):...
39
40 > def test_read_method_record_ssd_output_txt(ssd):...
46
47 > def test_write_ssd(ssd):...
51
52 > def test_write_check_file(ssd):...
```

### ▲ 기본 기능 테스트

```
164 > def test_read_from_buffer_when_lba_is_cached():...
180
181 > def test_merge_buffer_commands_when_possible():...
206
207 > def test_buffer_commands_write_when_possible():...
226
227 > def test_read_buffer_commands_when_not_exists():...
243
244 > def test_merge_buffer_commands_when_not_same_index():...
268
269 > def test_merge_buffer_commands_when_same_index():...
294
295 > def test_merge_buffer_commands_when_same_index_with_erase_range_1():...
320
321 > def test_merge_buffer_commands_when_erase_range_2():...
352
353 > def test_flush_buffer_when_mode_is_flush_should_execute_instruction():...
375
376 > def test_flush_buffer_when_buffers_are_full_should_execute_instruction():...
```

### ▲ 다양한 corner case 테스트 추가

# TDD – Shell

## 기능 안정성 향상

- 이전에 정의한 기능들에 대해 지속적으로 테스트를 진행하며 개발 진행

```
def test_read(ssd_py_path):  
    res = Shell.read(lba=0)  
    assert res == "[Read] LBA 00 : 0x00000000"
```

```
def test_write(ssd_py_path):  
    res = Shell.write(  
        lba=3,  
        value="0x00000000"  
    )  
    assert res == "[Write] Done"
```

```
def test_full_read(ssd_py_path):  
    ret = "[Full Read]"  
    for i in range(100):  
        ret += f"\nLBA {i:0>2} : 0x00000000"  
    res = Shell.full_read()  
    assert res == ret
```

```
def test_full_write(ssd_py_path):  
    res = Shell.full_write(value="0x00000000")  
    assert res == "[Full Write] Done"
```



master 에서 test\_help 가 실패하네요

👍 1 😊



## [FEAT] test shell에 mock test들 추가, test\_help 버그 수정 #60

yoondayoung merged 2 commits into master from feature/test-shell-mock yesterday

Conversation 2 Commits 2 Checks 0 Files changed 1



yoondayoung commented yesterday

Collaborator ...

test shell에서 shell에서 subprocess.run 호출 및 파일 접근 동작 확인 목적 mock test들 추가했습니다.



[FEAT] add mock test for shell

89de25e

+ New changes since you last viewed

View changes

[FIX] fix bug in test\_help

b67cf08





Mocking 활용



# Mock 사용 - Shell

## Mock 기반 테스트로 요구사항 도출

- Shell 클래스 구현 전 Mock 객체로 테스트 작성
- 테스트를 통해 필수 개발 항목 리스트업!

```
def test_read(mock: MockerFixture):
    shell = mock.Mock()
    shell.read.return_value = "[Read] LBA 00 : 0x00000000"
    res = shell.read(lba=0)
    assert res == "[Read] LBA 00 : 0x00000000"
    shell.read.assert_called()
```

```
def test_write(mock: MockerFixture):
    shell = mock.Mock()
    shell.write.return_value = "[Write] Done"
    res = shell.write(
        lba=3,
        value="0x00000000"
    )
    assert res == "[Write] Done"
    shell.write.assert_called()
```

```
def test_full_read(mock: MockerFixture):
    shell = mock.Mock()
    ret = "[Full Read]"
    for i in range(100):
        ret += f"\nLBA {i:0>2} : 0x00000000"
    shell.full_read.return_value = ret
    res = shell.full_read()
    assert res == ret
    shell.full_read.assert_called()
```

```
def test_full_write(mock: MockerFixture):
    shell = mock.Mock()
    shell.full_write.return_value = "[Full Write] Done"
    res = shell.full_write(value="0x00000000")
    assert res == "[Full Write] Done"
    shell.full_write.assert_called()
```


# Mock 사용 - Shell

## Mock에서 실제 구현으로 전환

- Mock을 실제 SSD클래스로 교체하여 통합 테스트 수행
- 기존 Mock 테스트 구조 재사용하여 빠른 통합 테스트 가능!

```
def test_read(mock: MockerFixture):  
    shell = mocker.Mock()  
    shell.read.return_value = "[Read] LBA 00 : 0x00000000"  
    res = shell.read(lba=0)  
    assert res == "[Read] LBA 00 : 0x00000000"  
    shell.read.assert_called()
```

```
def test_read(ssd_py_path):  
    res = Shell.read(lba=0)  
    assert res == "[Read] LBA 00 : 0x00000000"
```



# Mock 사용 - Shell

## Subprocess Mock으로 Shell 테스트

- Shell 클래스만의 동작을 검증하기 위해 subprocess를 Mock으로 대체
- 오래 걸리는 shell script의 경우 shell 동작 로직 검증으로 디버깅 시간 단축

```
@pytest.fixture
def file_mock(mock):
    return mock.patch('builtins.open', mock.mock_open(read_data=''))

@pytest.fixture
def shell_mock(mock):
    pat = mock.patch('commands.subprocess.run')
    mock_result = mock.Mock(returncode=0)
    pat.return_value = mock_result
    return pat
```



```
def test_read_mock(file_mock, shell_mock):
    Shell.execute_command(ShellCommandEnum.READ, [0])
    shell_mock.assert_called_once_with(['python', 'ssd.py', 'R', '0'], text=True)
    file_mock.assert_called_once_with('ssd_output.txt', 'r')

def test_write_mock(file_mock, shell_mock):
    Shell.execute_command(ShellCommandEnum.WRITE, [3, 0x00000000])
    shell_mock.assert_called_once_with(['python', 'ssd.py', 'W', '3', 0], text=True)
    file_mock.assert_called_once_with('ssd_output.txt', 'r')

def test_full_read_mock(file_mock, shell_mock):
    Shell.full_read()
    shell_mock.assert_called_with(['python', 'ssd.py', 'R', '99'], text=True)
    file_mock.assert_called_with('ssd_output.txt', 'r')

def test_full_write_mock(file_mock, shell_mock):
    Shell.full_write(3)
    shell_mock.assert_called_with(['python', 'ssd.py', 'W', '99', 3], text=True)
    file_mock.assert_called_with('ssd_output.txt', 'r')
```

▲ subprocess.run, file open 함수 patch

# Mock 사용 - SSD

- patching 활용하여 검증하고자 하는 동작 검증에만 집중할 수 있었음

```
def test_flush_buffer_when_buffers_are_full_should_execute_instruction():
    ssd = SSD()
    commands = [
        [None, "W", "2", "0x12345678"]
    ]
    initial_buffers = [Buffer(command="W", lba=2, data="0xABCDABCD", range=""),
                       Buffer(command="W", lba=3, data="0xABCDABCD", range=""),
                       Buffer(command="W", lba=4, data="0xABCDABCD", range=""),
                       Buffer(command="W", lba=5, data="0xABCDABCD", range=""),
                       Buffer(command="W", lba=6, data="0xABCDABCD", range="")]
    initial_file_data = '0\t0x11111111\n1\t0x22222222\n2\t0x33333333\n3\t0x33333333\n4\t0x33333333\n5\t0x33333333\n6\t0x33333333\n'
    expected = '0\t0x11111111\n1\t0x22222222\n2\t0xABCDABCD\n3\t0x33333333\n4\t0x33333333\n5\t0x33333333\n6\t0x33333333\n'
    expected += '0\t0x11111111\n1\t0x22222222\n2\t0x33333333\n3\t0xABCDABCD\n4\t0x33333333\n5\t0x33333333\n6\t0x33333333\n'
    expected += '0\t0x11111111\n1\t0x22222222\n2\t0x33333333\n3\t0x33333333\n4\t0xABCDABCD\n5\t0x33333333\n6\t0x33333333\n'
    expected += '0\t0x11111111\n1\t0x22222222\n2\t0x33333333\n3\t0x33333333\n4\t0x33333333\n5\t0xABCDABCD\n6\t0x33333333\n'
    expected += '0\t0x11111111\n1\t0x22222222\n2\t0x33333333\n3\t0x33333333\n4\t0x33333333\n5\t0x33333333\n6\t0xABCDABCD\n'
    expected_lines = expected.splitlines(keepends=True)
```

```
with (patch.object(BufferManager, attribute='get_buffer', return_value=initial_buffers) as mock_get_buffer,
      patch('builtins.open', mock_open(read_data=initial_file_data)) as mock_file, \
      patch.object(BufferManager, attribute='set_buffer') as mock_set_buffer, \
      patch.object(FileManager, attribute='write_output')):
    ssd.run(commands[0]) 실제 확인하고자 하는 동작
```

```
write_calls = mock_file().write.call_args_list
expected_calls = [call(line) for line in expected_lines]
assert write_calls == expected_calls
```

테스트 환경 세팅



# Mock 사용 - Logger

- patching 활용하여 검증하고자 하는 동작 검증에만 집중할 수 있었음

```
def test_rollover_happens_when_log_exceeds_size(fixed_datetime, logger):  
    with patch(target: "os.path.getsize", return_value=LOG_FILE_MAX_SIZE + 1), \  
        patch.object(Logger, attribute: "_get_timestamp", return_value=(fixed_datetime[0], fixed_datetime[1])), \  
        patch("os.rename") as mock_rename, \  
        patch("builtins.open", mock_open()):  
        logger._rollover_if_needed() 실제 확인하고자 하는 동작  
        expected_new_log = ".\\until_240716_12h_34m_56s.log"  
        mock_rename.assert_called_once_with(*args: f".\\{LOG_FILE_NAME}", expected_new_log)
```

} 테스트 환경 세팅

# Fixture 활용 – File Manager

- pytest의 fixture 활용하여 test 코드의 중복 제거

```
@pytest.fixture
def ssd(mock):
    ssd = SSD()
    ssd.file_manager = mock.Mock()
    return ssd

def test_write_and_read(ssd):
    lba = 0
    expected = "0x12345678"
    ssd.write(lba, expected)
    ssd.read(lba)

    ssd.file_manager.read_nand.assert_called_once()
    ssd.file_manager.write_nand.assert_called_once()
    ssd.file_manager.write_output.assert_called()
```



Refactoring을 통한 코드 전후 비교





# 마 이기| Pythonic 이다!

```
class Buffer: 48 usages 👤 donghyuk +2
```

```
def __init__(self, command: str = "", lba: int = 0, data: str = "", range: int = 0) -> None:
    self.command = command
    self.lba = lba
    self.data = data
    self.range = range
    self.idx: Optional[int] = None
```

```
def __str__(self) -> str: 👤 donghyuk +1
    """This is pythonic."""
    if self.command == "W":
        return f"{self.idx}_{self.command}_{self.lba}_{self.data}"
    elif self.command == 'E':
        return f"{self.idx}_{self.command}_{self.lba}_{self.range}"
    else:
        return f"{self.idx}_empty"
```

## ✅ 왜 Magic Method가 Pythonic한가?

Python은 객체지향 언어이며, 내부적으로 많은 기능이 **특수 메서드(magic method)**를 통해 작동합니다. 이 메서드들을 활용하면 **자연스럽고 일관된 인터페이스**를 제공할 수 있고, 이는 Pythonic한 코드로 이어집니다.

## 📖 예시: `__str__`

python

🔍 복사 ✎ 편집

```
class User:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return f"User: {self.name}"
```

```
u = User("Alice")
print(u) # 🖱 자동으로 __str__ 호출됨
```

→ `__str__()` 을 정의하면 `print()` 로 객체를 출력할 때 **사람이 읽기 쉬운 형태**가 되어 Pythonic합니다.

# 마 이끼| Dependency Injection 이다!

평소에 가장 많이 하던 고민

```
class SSD: 28 usages  👤 donghyuk +4
```

```
def __init__(  👤 donghyuk
```

```
    self,
```

```
    file_manager: FileManager = FileManager(),
```

```
    buffer_manager: BufferManager = BufferManager(),
```

```
) -> None:
```

```
    self.file_manager = file_manager
```

```
    self.buffer_manager = buffer_manager
```

장점

설명

✅ 테스트 가능

Mock 주입 가능

✅ 확장성

다양한 구현체 주입 가능

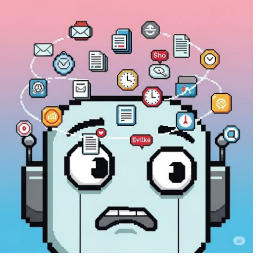
✅ 유지보수

결합도 낮고 변경에 유연

✅ 원칙 준수

DIP(의존 역전 원칙) 구현

# Shell 클래스 단일 책임 원칙



```
class Shell:
    - get_command()
    - find_command()
    - _read_output_file()
    - _read_value()
    - _write_value()
    - read(), write()
    - full_read(), full_write()
    - script_1~4()
    - execute_command()
    - run()
```

▲ 리팩토링 전 Shell

SSD 호출  
담당

명령 파싱  
담당

커맨드  
실행 담당

```
class SSDController:
    - _cache_inout()
    - read(lba)
    - write(lba, value)
    - erase(lba, size)
    - flush()

class ShellParser:
    - parse()
    - find_command(command_str)

class CommandExecutor:
    ssd_controller = SSDController
    - read(lba)
    - write(lba, value)
    - full_write(value)
    - full_read(num_iter)
    - erase(lba, size)
    - erase_range(start_lba, end_lba)
    - flush()
```

Script  
담당

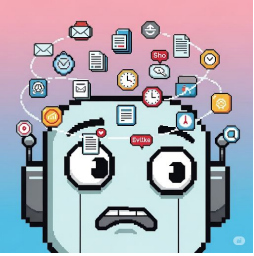
```
class ScriptExecutor:
    ssd_controller = SSDController
    command_executor = CommandExecutor
    - script_1(num_iter)
    - script_2(num_iter)
    - script_3(num_iter)
    - script_4(num_iter)

class Shell:
    shell_parser = ShellParser
    command_executor = CommandExecutor
    script_executor = ScriptExecutor
    - _command_mapper(cmd)
    - execute_command(cmd, args)
    - run()
    - run_script(script)
```

Shell 동작  
담당

▲ 리팩토링 후 Shell 관련 클래스

# SSD 클래스 단일 책임 원칙



```
class SSD:
    def __init__(self):
        self.contents = ""
    - read(LBA)
    - write(LBA, data)
    - file_to_dict()
    - dict_to_file(data)
    - write_output_file(str)
```

▲ 리팩토링 전 SSD

버퍼  
데이터  
담당

버퍼 관리  
담당

```
class Buffer:
    __init__(self, command, lba, data, range)
    __str__(self)

class BufferManager:
    __init__(self)
    _check_initial_buffer(self)
    get_buffer(self)
    set_buffer(self, buffers)
    _update_file(self)
    _register_buffer(self)
    update(cls, buffers, idx, new_buffer, new_buffers)
    merge_overall(cls, new_buffers)
```

파일  
입출력  
담당

명령 처리  
담당

```
class FileManager:
    __init__(self)
    init_nand(cls)
    _read_whole_lines(cls)
    _save_to_nand(cls, data)
    read_nand(cls, lba)
    write_nand(cls, lba, change_data)
    erase_nand(cls, lba, size)
    write_output(cls, contents)

class SSD:
    __init__(self)
    _read(self, lba)
    _write(self, lba, data)
    _erase(self, lba, size)
    _flush(self, buffers)
    run(self, args)
    _validate_command(self, args)
    _execute_command(self, mode, ...)
    _process_read(self, buffers, lba)
    _process_write(self, ...)
    _process_erase(self, ...)
```

▲ 리팩토링 후 SSD 관련 클래스

# Refactoring – Naming

- 파일 이름을 명확하게 변경

```
1 - from buffer import BufferManager
```

```
1 + from command_buffer import BufferManager
```

- snake 형식의 변수로 통일

```
75         if "empty" in fileName:
76             continue
77
78 -         newBuffer = Buffer()
79         chunks = fileName.split("_")
80         if len(chunks) < 4:
81             continue
82
```

```
75         if "empty" in fileName:
76             continue
77
78 +         new_buffer = Buffer()
79         chunks = fileName.split("_")
80         if len(chunks) < 4:
81             continue
82
```



# Refactoring – Naming

- snake 형식 함수명 통일 (함수명에 대문자 -> 소문자로 변경)

<pre>64 65 - def test_execute_command_When_write_normal_Should_write_value(): 66     ssd = SSD(FileManager()) 67     initial_file_data = '\0\t0x11111111\n1\t0x22222222\n2\t0x33333333\n' 68     mocked_open = mock_open(read_data=initial_file_data) @@ -78,24 +78,24 @@ def test_execute_command_When_write_normal_Should_write_value(): 78     mocked_open().write.assert_any_call("") 79 80 81 - def test_execute_command_When_args_is_invalid_Should_write_error(): 82     run_execute_command_and_assert([None, "W"], 'w', 'ERROR') 83     run_execute_command_and_assert([None], 'w', 'ERROR') 84 85 86 - def test_execute_command_When_read_out_of_range_Should_write_error(): 87     run_execute_command_and_assert([None, "R", "100"], 'w', 'ERROR') 88     run_execute_command_and_assert([None, "R", "-1"], 'w', 'ERROR') 89 90 91 - def test_execute_command_When_write_invalid_value_Should_write_error(): 92     run_execute_command_and_assert([None, "W", "100", "0x00000000"], 'w', 'ERROR')</pre>	<pre>64 65 + def test_execute_command_when_write_normal_should_write_value(): 66     ssd = SSD(FileManager()) 67     initial_file_data = '\0\t0x11111111\n1\t0x22222222\n2\t0x33333333\n' 68     mocked_open = mock_open(read_data=initial_file_data) 78     mocked_open().write.assert_any_call("") 79 80 81 + def test_execute_command_when_args_is_invalid_should_write_error(): 82     run_execute_command_and_assert([None, "W"], 'w', 'ERROR') 83     run_execute_command_and_assert([None], 'w', 'ERROR') 84 85 86 + def test_execute_command_when_read_out_of_range_should_write_error(): 87     run_execute_command_and_assert([None, "R", "100"], 'w', 'ERROR') 88     run_execute_command_and_assert([None, "R", "-1"], 'w', 'ERROR') 89 90 91 + def test_execute_command_when_write_invalid_value_should_write_error(): 92     run_execute_command_and_assert([None, "W", "100", "0x00000000"], 'w', 'ERROR')</pre>
--	---

# Refactoring – Naming

- 여러 곳에서 쓰는 변수 네이밍 통일 (LBA, read\_idx, loc 등을 모두 lba로 통일)

<pre>53 54 - def read(self, LBA): 55 -     if LBA &lt; 0 or LBA &gt; 99: 56         self.file_manager.write_output_file("ERROR") 57         return 58 -     read_value = self.file_manager.read_nand_txt(LBA) 59     if read_value == "": 60         self.file_manager.write_output_txt("ERROR") 61     else:</pre>	<pre>59 60 + def read(self, lba): 61 +     if lba &lt; 0 or lba &gt; 99: 62         self.file_manager.write_output_file("ERROR") 63         return 64 +     read_value = self.file_manager.read_nand_txt(lba) 65     if read_value == "": 66         self.file_manager.write_output_txt("ERROR") 67     else:</pre>
<pre>@@ -95,11 +101,11 @@ def check_hex(data):</pre>	
<pre>95 96     ssd = SSD(FileManager()) 97     mode = sys.argv[1] 98 -     LBA = int(sys.argv[2]) 99     if mode == "W" and len(sys.argv) == 4: 100 101 -     ssd.write(LBA, sys.argv[3]) 102     elif mode == "R" and argument_len == 3: 103 -     ssd.read(LBA) 104     else: 105         print("Invalid argument")</pre>	<pre>101 102     ssd = SSD(FileManager()) 103     mode = sys.argv[1] 104 +     lba = int(sys.argv[2]) 105     if mode == "W" and len(sys.argv) == 4: 106 107 +     ssd.write(lba, sys.argv[3]) 108     elif mode == "R" and argument_len == 3: 109 +     ssd.read(lba) 110     else: 111         print("Invalid argument")</pre>
<pre>tests/test_ssd.py</pre>	
<pre>@@ -21,8 +21,8 @@ def test_read_ssd_nand_txt_file_called_by_read(mock: MockerFixture):</pre>	
<pre>21     mock_file_manager = mocker.Mock() 22     ssd = SSD(mock_file_manager) 23 24 -     read_idx = 0 25 -     ssd.read(read_idx)</pre>	<pre>21     mock_file_manager = mocker.Mock() 22     ssd = SSD(mock_file_manager) 23 24 +     lba = 0 25 +     ssd.read(lba)</pre>

# Refactoring – Naming

- 주석과 조건문을 통해 이해해야 하는 코드를 적절한 함수 이름으로 리팩터하여 가독성 개선

<pre>buffers = self.buffer_manager.get_buffer() # flush 조건 체크 if len(buffers) == 5:     self.flush(buffers)     buffers = self.buffer_manager.get_buffer()</pre>		<pre>194 buffers = self.buffer_manager.get_buffer() 195 + buffers = self._flush_when_buffer_are_full_or_flush_mode(buffers, mode) 196 + if mode == "F": 197 +     self.file_manager.write_output_txt("") 198 +     return 199 +</pre>
<pre># Buffer에 접근 먼저 해서 알고리즘 동작하게 하기. # R</pre>		<pre>200 201 # Buffer에 접근 먼저 해서 알고리즘 동작하게 하기. 202 # R</pre>
<pre>-300,6 +299,14 @@ def _execute_command_new(self, mode, lba: int, data='', erase_size=0):</pre>		
<pre>self.buffer_manager.set_buffer(new_buffers) self.file_manager.write_output_txt("")</pre>		<pre>299 self.buffer_manager.set_buffer(new_buffers) 300 self.file_manager.write_output_txt("") 301</pre>
	<pre>302 + 303 + 304 + 305 + 306 + 307 + 308 +</pre>	<pre>def _flush_when_buffer_are_full_or_flush_mode(self, buffers, mode):     # flush 조건 체크     if len(buffers) == 5 or mode == "F":         self.flush(buffers)         buffers = self.buffer_manager.get_buffer()     return buffers</pre>



# Refactoring – Naming

- 함수 naming 더 명확하게 개선  
(execute\_command\_new → execute\_command\_with\_buffers)

```
if mode == "F":
    buffers = self.buffer_manager.get_buffer()

    self.flush(buffers)
    self.file_manager.write_output_txt("")
    return

lba = int(args[2])
if mode == "W":
    hex = args[3]
    self.execute_command_new(mode=mode, lba=lba, data=hex)
elif mode == "R":
    self.execute_command_new(mode=mode, lba=lba)
elif mode == "E":
    size = self._parse_int_or_empty(args[3])
    self.execute_command_new(mode=mode, lba=lba, erase_size=size)
```

```
106
107     if mode == "F":
108 +         self._execute_command_with_buffers(mode=mode)

jenny113oh yesterday Collaborator
추상화(?) 레벨이 잘 맞춰진 것 같습니다! 👍

Reply...

Resolve conversation

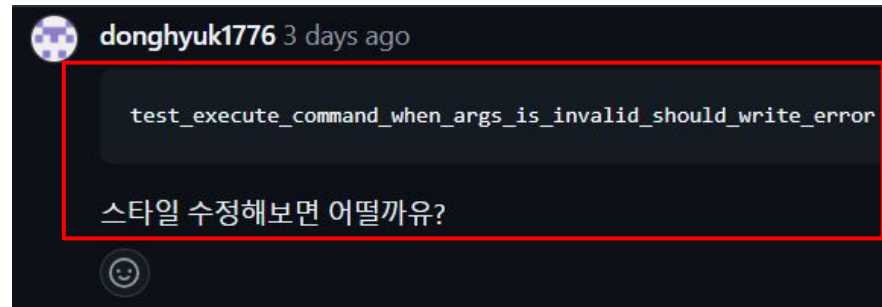
109 +     else:
110 +         lba = int(args[2])
111 +         if mode == "W":
112 +             hex_string = args[3]
113 +             self.execute_command_with_buffers(mode=mode, lba=lba, data=hex_string)
114 +         elif mode == "R":
115 +             self.execute_command_with_buffers(mode=mode, lba=lba)
116 +         elif mode == "E":
117 +             size = self._parse_int_or_empty(args[3])
118 +             self.execute_command_with_buffers(mode=mode, lba=lba, erase_size=size)
```

119

# Refactoring – Extract Method

- 테스트 코드의 중복 개선을 위해 함수로 리팩터

```
102 -     args = [None, "R", "100"]
103 -     ssd = SSD(FileManager())
104 -     ssd.execute_command(args)
105 -     mocked_open.assert_called_once_with(FILENAME_OUT, 'w')
106 -     mocked_open().write.assert_called_once_with('ERROR')
107
108 -     mocked_open = mock_open()
109
110 -     with patch('builtins.open', mocked_open):
111 -         args = [None, "R", "-1"]
112 -         ssd = SSD(FileManager())
113 -         ssd.execute_command(args)
114 -         mocked_open.assert_called_once_with(FILENAME_OUT, 'w')
115 -         mocked_open().write.assert_called_once_with('ERROR')
116
```



```
82 +     run_execute_command_and_assert([None, "W"], 'w', 'ERROR')
83 +     run_execute_command_and_assert([None], 'w', 'ERROR')
84
85
86 + def test_execute_command_when_read_out_of_range_should_write_error():
87 +     run_execute_command_and_assert([None, "R", "100"], 'w', 'ERROR')
88 +     run_execute_command_and_assert([None, "R", "-1"], 'w', 'ERROR')
```

# Refactoring – Extract Method

- 가독성을 높이기 위해 코드 블록을 내부 함수로 리팩터

```
if mode == "R":
    for i, b in enumerate(buffers):

        if b.command == "W":
            if b.lba == lba:
                self.file_manager.write_output_txt(b.data)
                return
            if b.command == "E":
                if lba >= b.lba and lba < b.lba + b.range:
                    self.file_manager.write_output_txt("0x00000000")
                    return
        self.read(lba)
    return
```

203      if mode == "R":

204      +      read\_buffer\_success = self.read\_buffer\_first(buffers, lba)

205      +      if not read\_buffer\_success:

206      +      self.read(lba)

207      return

**Comment:** Indent가 많이 줄어서 더욱 가독성이 좋고 여러가지 측면에서 클린해진 것 같습니다!

**Buttons:** Collaborator, Reply..., Resolve conversation


# Refactoring – Extract Method

- 코드 중복을 개선하기 위해 함수(\_error)로 리팩터하고, args validation check와 error message를 dictionary로 관리하여 추가하기 용이하게 개선

```
130
131 +     argument_len = len(args)
132 +     if argument_len < 2:
133 +         return _error("At least one argument are required")

134
135 +     mode = args[1]
136 +     valid_modes = {"W", "R", "E", "F"}
137 +     if mode not in valid_modes:
138 +         return _error("Mode should be in ('W', 'R', 'E', 'F')")
139 +
140 +     # mode별 기대하는 argument 개수와 메시지
141 +     expected_args = {
142 +         "W": (4, "Mode W need lba and value"),
143 +         "R": (3, "Mode R need lba"),
144 +         "E": (4, "Mode E need lba and size"),
145 +         "F": (2, "Mode F need only command"),
146 +     }
147 +
148 +     expected_len, error_msg = expected_args[mode]
149 +     if argument_len != expected_len:
150 +         return _error(error_msg)
```

Comment on lines +141 to +150

 sinamorlgo 19 hours ago Collaborator ...

아주 깔끔합니다!!

100%

- 긴 함수를 적절한 단위로 리팩터하여 가독성 개선 (120라인 → 20라인)

```
def _execute_command_with_buffers(self, mode, lba=None, data='', erase_size=0):
    buffers = self.buffer_manager.get_buffer()
    buffers = self._flush_when_buffer_are_full_or_flush_mode(buffers, mode)
    if mode == "F":
        self.file_manager.write_output_txt("")
        return

    # Buffer에 접근 먼저 해서 알고리즘 동작하게 하기.
    # R
    if mode == "R":
        read_buffer_success = self.read_buffer_first(buffers, lba)
        if not read_buffer_success:
            self.read(lba)
        return

    # W
    new_buffers = []
    new_buffer = Buffer(mode, lba, data, erase_size)
    is_need_append_new_buffer = True
    if mode == "W":
        for i, each_buffer in enumerate(buffers):

            # 1. W인 경우
            if each_buffer.command == "W":
                if each_buffer.lba != lba:
                    new_buffers.append(each_buffer)
                    continue

                is_need_append_new_buffer = False
                break

            each_buffer.lba += 1
            each_buffer.range -= 1
            elif (each_buffer.lba + each_buffer.range - 1) == lba:
                each_buffer.range -= 1
                new_buffers.append(each_buffer)
                continue

    # E
    if mode == "E":
        for i, each_buffer in enumerate(buffers):
            # 1. W인 경우
            if each_buffer.command == "W":
                if lba <= each_buffer.lba < lba + erase_size:
                    continue
```

```
def _execute_command_with_buffers(self, mode, lba=None, data='', erase_size=0):
    buffers = self.buffer_manager.get_buffer()
    buffers = self._flush_when_buffer_are_full_or_flush_mode(buffers, mode)
    if mode == "F":
        self.file_manager.write_output_txt("")
        return
    if mode == "R":
        self._process_read_mode(buffers, lba)

    return

    # W
    new_buffers = []
    new_buffer = Buffer(mode, lba, data, erase_size)
    is_need_append_new_buffer = True
    if mode == "W":
        is_need_append_new_buffer, new_buffers = self._process_write_mode(buffers,
is_need_append_new_buffer, lba,
new_buffer, new_buffers)

    # E
    if mode == "E":
        is_need_append_new_buffer, new_buffers = self._process_erase_mode(buffers, erase_size,
is_need_append_new_buffer, lba,
new_buffer, new_buffers)

    if is_need_append_new_buffer:
        new_buffers.append(new_buffer)
    # 마지막에 rename
    self.buffer_manager.set_buffer(new_buffers)
    self.file_manager.write_output_txt("")
```



# Refactoring – Extract Method

- validation check 항목을 Method로 추출

```
@classmethod
def erase(cls, lba: int, size: int) -> str:
-     if not isinstance(lba, int):
-         return "[Erase] ERROR"
-     if not isinstance(size, int):
-         return "[Erase] ERROR"

-     start, end = (lba, lba + size)
-     step = 10
-     if start < 0:
-         return "[Erase] ERROR"
-     if end > 100:
-         return "[Erase] ERROR"
-     if size < 1 or size > 100:
-         return "[Erase] ERROR"
-
-     for i in range(start, end, step):
-         _start, _end = (i, min(i + step, end))
-         _size = _end - _start
```

```
173     @classmethod
174     def erase(cls, lba: int, size: int) -> str:
175 +         is_valid_args = utils.validate_erase_args(lba, size)
176 +         if not is_valid_args:
177             return "[Erase] ERROR"
178
179         step = 10
180 +         start, end = (lba, lba + size)

def validate_erase_args(lba: int, size: int) -> bool:
    if not isinstance(lba, int):
        return False
    if not isinstance(size, int):
        return False

    start, end = (lba, lba + size)
    if start < 0:
        return False
    if end > 100:
        return False
    if size < 1 or size > 100:
        return False
    return True
```

# Refactoring – Reusable Code

- “latest.log” 등 magic literal, magic number 제거, constant로 재 활용

	3	+ from constant import LOG_FILE_MAX_SIZE, LOG_FILE_NAME, PAST_LOG_FILE_FORMAT, LOG_METHOD_NAME_WIDTH			
3	4				
4	5				
5	6	class Logger:			
6	-	MAX_SIZE = 10 * 1024			
7	-				
8	def __init__(self, log_dir="."):	7	def __init__(self, log_dir="."):		
9	self.log_dir = log_dir	8	self.log_dir = log_dir		
10	-	self.latest_log = os.path.join(log_dir, "latest.log")	9	+	self.latest_log = os.path.join(log_dir, LOG_FILE_NAME)
11	if not os.path.exists(self.latest_log):	10	if not os.path.exists(self.latest_log):		
12	with open(self.latest_log, "w"):	11	with open(self.latest_log, "w"):		
13	pass	12	pass		
@@ -18,13 +17,13 @@	def _get_timestamp(self):				
18		17			
19	def _format_log(self, method_name, message):	18	def _format_log(self, method_name, message):		
20	timestamp_str, _ = self._get_timestamp()	19	timestamp_str, _ = self._get_timestamp()		
21	-	method_name_padded = method_name.ljust(20)	20	+	method_name_padded = method_name.ljust(LOG_METHOD_NAME_WIDTH)
22	return f"[{timestamp_str}] {method_name_padded}: {message}\n"	21	return f"[{timestamp_str}] {method_name_padded}: {message}\n"		
23		22			
24	def _rollover_if_needed(self):	23	def _rollover_if_needed(self):		
25	-	if os.path.getsize(self.latest_log) > self.MAX_SIZE:	24	+	if os.path.getsize(self.latest_log) > LOG_FILE_MAX_SIZE:
26	_, now = self._get_timestamp()	25	_, now = self._get_timestamp()		
27	-	new_name = now.strftime("until %y%m%d %Hh%Mm %Ss.log")	26	+	new_name = now.strftime(PAST_LOG_FILE_FORMAT)

# Refactoring – Reusable Code

- 공통 test cases들은 fixture로 재 활용

```
10
11 def test_logger_init_creates_log_file():
12     with patch("os.path.exists", return_value=False),
13         patch("builtins.open", mock_open()) as m_open:
14         logger(log_dir=".")
15         m_open.assert_called_once_with(".\\latest.log", "w")
16
17 - def test_format_log_output(fixed_datetime):
18 -     logger = Logger(log_dir=".")
19     with patch.object(logger, "_get_timestamp", return_value=
        (fixed_datetime[0], fixed_datetime[1])):
20 -         result = logger._format_log("Shell.run", "hello")
21 -         assert result == "[24.07.16 12:34] Shell.run : hello\n"
22
12
13 + @pytest.fixture
14 + def logger():
15 +     return Logger(log_dir=".")
16 +
17 +
18 + @pytest.fixture
19 + def test_case_for_print():
20 +     return {"method_name": "Shell.run",
21 +             "message": "hello",
22 +             "expected_line": f"[24.07.16 12:34]
23 +             {\"Shell.run\".ljust(LOG_METHOD_NAME_WIDTH)}: hello\n"}
24 +
25 def test_logger_init_creates_log_file():
26     with patch("os.path.exists", return_value=False),
27         patch("builtins.open", mock_open()) as m_open:
28         logger(log_dir=".")
29         m_open.assert_called_once_with(f".\\{LOG_FILE_NAME}", "w")
30
31 + def test_format_log_output(fixed_datetime, logger, test_case_for_print):
32     with patch.object(logger, "_get_timestamp", return_value=
        (fixed_datetime[0], fixed_datetime[1])):
33 +         result = logger._format_log(test_case_for_print["method_name"],
34 +                                     test_case_for_print["message"])
35 +         assert result == test_case_for_print["expected_line"]
```



# Refactoring – Guard Clause

- 곧곧에 나눠진 조건문을 모아서 guard clause로 리팩터

```
103         if argument_len < 3:
104             print("At least two argument are required")
105             self.file_manager.write_output_txt("ERROR")
106 -         return
107 -         if not self._index_valid(args[2]):
108 -             print("The index should be an integer among 0 ~ 99")
109 -             self.file_manager.write_output_txt("ERROR")
110 -             return
111
112         mode = args[1]
113 -         lba = self._parse_int_or_empty(args[2])
114 -         if lba == "":
115             self.file_manager.write_output_txt("ERROR")
116 -             print("Invalid argument")
117 -             if mode == "W" and self.check_hex(args[3]) and argument_len == 4:
118 -                 self.write(lba, args[3])
119 -             elif mode == "R" and argument_len == 3:
120 -                 self.read(lba)
121 -             elif mode == "E" and argument_len == 4:
122 -                 size = self._parse_int_or_empty(args[3])
123 -                 if size == "" or size < 1 or size > 10 or lba + size > 100:
124                     self.file_manager.write_output_txt("ERROR")
125 -                     print("Invalid argument")
126 -                 else:
127 -                     self.erase(lba, size)
128 -                 else:
129                     self.file_manager.write_output_txt("ERROR")
130 -                     print("Invalid argument")
```



```
100
101     def execute_command(self, args):
102 +         if not self._args_valid_guard_clauses(args):
103 +             return
104
105 +         mode = args[1]
106 +         lba = int(args[2])
107 +         if mode == "W":
108 +             hex = args[3]
109 +             self.write(lba, hex)
110 +         elif mode == "R":
111 +             self.read(lba)
112 +         elif mode == "E":
113 +             size = self._parse_int_or_empty(args[3])
114 +             self.erase(int(args[2]), size)
115 +
```

Han-nu-ri marked this conversation as resolved.

# Refactoring – Guard Clause

- 곧곧에 나눠진 조건문을 모아서 guard clause로 리팩터

```
@classmethod
- def _command_mapper(cls, cmd: ShellCommandEnum):
-     if cls._command_mapping_dict is None:
-         cls._command_mapping_dict = {
-             ShellCommandEnum.HELP: lambda: MESSAGE_HELP,
-             ShellCommandEnum.READ: cls.command_executor.read,
-             ShellCommandEnum.WRITE: cls.command_executor.write,
-             ShellCommandEnum.FULLREAD: cls.command_executor.full_read,
-             ShellCommandEnum.FULLWRITE: cls.command_executor.full_write,
```

```
301 @classmethod
302 + def _command_mapper(cls, cmd: ShellCommandEnum) -> Union[Callable, str]:
303 +     if cls._command_mapping_dict is not None:
304 +         return cls._command_mapping_dict[cmd]
305 +
306 +     cls._command_mapping_dict = {
307 +         ShellCommandEnum.HELP: lambda: MESSAGE_HELP,
308 +         ShellCommandEnum.READ: cls.command_executor.read,
309 +         ShellCommandEnum.WRITE: cls.command_executor.write,
```

```
for cmd in cmds:
    cmd = cmd.strip()
    print(f"{cmd:<28}__ Run...", end="", flush=True)
    cmd_enum = cls.shell_parser.find_command(cmd)

    if cmd_enum == ShellCommandEnum.EXIT:
        break

    if cmd_enum is None:

        continue

    ret = cls.execute_command(cmd_enum, [])
    if ret is not None:
        if ret == MESSAGE_PASS:
            print("Pass")
        else:
            print("FAIL!")
        break
```

```
352 for cmd in cmds:
353 +     cmd_enum = cls.shell_parser.find_command(command_str=cmd.strip())
354 +     if cmd_enum is None:
355 +         continue
356 +     print(f"{cmd_enum.value:<28}__ Run...", end="", flush=True)
357     if cmd_enum == ShellCommandEnum.EXIT:
358         break
359 +     ret = cls.execute_command(cmd_enum, args=[])
360 +     if ret is None:
361 +         continue
362 +     if ret == MESSAGE_PASS:
363 +         print("Pass")
364 +     else:
365 +         print("FAIL!")
366 +         break
```



소감



# 소감



박해늑

파트에서 두명이서만 일하다가, 여기서 여러분들과 같은 목표로 동시에 개발을 해보니 코드리뷰가 왜 중요한지 더욱 깨닫게 되었습니다. 모두 감사합니다~!



정동혁

더욱 넓은 마음으로 코드리뷰 할 수 있게 되었고 너무 재미있었습니다. 돌리팀 최고!!!!



김희준

평소에 PR과 리뷰를 대강하는 것을 많이 반성하게 되었습니다!~ 같은 팀원분들과 협업하면서 점점 완성되는 프로젝트를 보며 뿌듯했고, 좀 더 꼼꼼히 리뷰를 할 수 있게 되었어요



한누리

코드와 Unit Test를 함께 작성하니 리팩토링할 때 더 용감하게 할 수 있었던 것 같아요! 재미있었습니다.



윤다영

실제로 TDD 개발 방식을 적용해보며 많이 배울 수 있는 유익한 시간이었습니다!



송주환

코드 리뷰 보다 훌륭한 팀원들이 더 중요한 게 아닌가 느꼈습니다. 다들 알아서 잘해주셔가지고.



오지은

프로젝트를 통해 이론으로 배운 부분을 실제 적용하려니 어려운 부분도 있었는데 함께하는 팀원분들이 잘 알려주시고 도와주셔서 잘 마무리할 수 있었던 것 같습니다. 재미있었습니다 :) 최고!



Q & A

