

CRA 양성 과정 실습 발표

C팀: CoDream

팀장 : 조영준

팀원 : 민동학, 박승욱, 이재원, 최일묵, 한재원

목차

- 조원 소개 및 역할
- 기능 구현 소개
- TDD 활용 예시
- Mocking 활용 예시
- 디자인 패턴 활용 예시
- 리팩토링을 통한 클린코드 전후 결과 비교
- 소감



조원 소개 및 역할



조원 소개 및 역할

- 조영준: 조장, Shell
- 민동학: Shell, Logger
- 박승욱: Shell, Logger
- 이재원: Shell, Buffer
- 최일묵: SSD, Buffer
- 한재원: SSD, Buffer


[C조]

CoDream(Code+Dream : 우리가 꿈꾸는 완벽한 코드)
(팀장 : 조영준, 조원: 민동학, 박승욱, 이재원, 최일묵, 한재원)

1. 17시 퇴근 준수
2. Approver 2인으로 지정
3. Push 하기 전 git pull 먼저 하기
4. 의견충돌시, 다수결로 결정하기
5. 자율적으로 쉬는시간 가지되 30분 이상 자리 비우지 않기
6. 식사 후, 12시 30분까지 착석하기
7. commit 시 [fix],[feat],[refactor]말머리 붙이기
8. 리뷰 올릴 때 링크도 같이 올리기
9. 리뷰할 사람은 이모지로 먼저 의사 표현 하기
10. Merge 완료 후 브랜치 지우기

[코드리뷰 전략]-CoDream

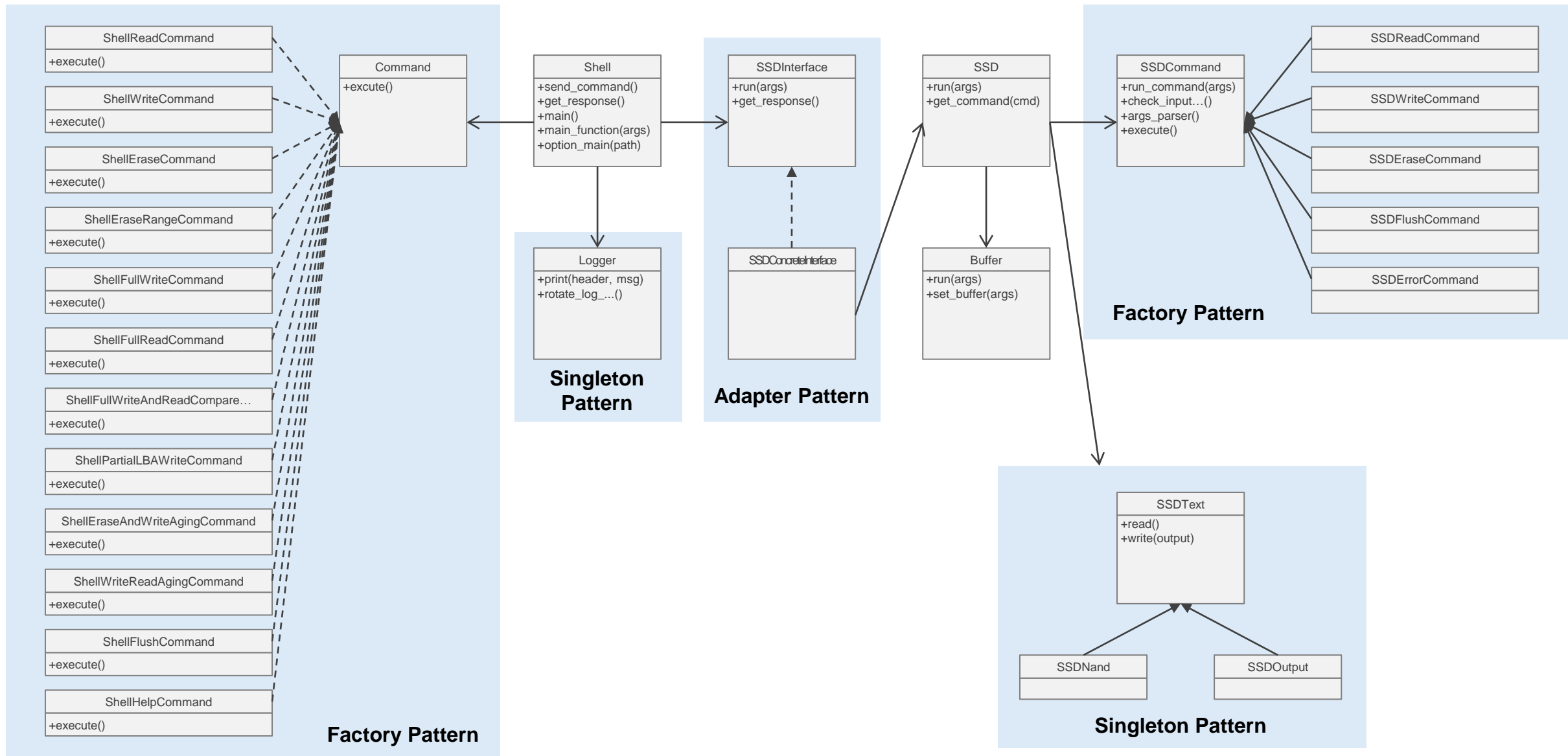
1. PR Description에 의도, 배경 설명 충분히 작성
2. 기능별 commit구분과 적당한 PR단위를 준수하기
3. 쿠션멘트와 긍정어를 포함하여 좋은 방향의 리뷰를 남기도록 하기
4. PR관리를 위해 리뷰 요청시 빠른 대응과 브랜치 관리 하기
5. 정해진 시간에 20-30분 피드백 시간을 갖고 피드백 이후 상처받지 않는 강인한 멘탈을 갖기



기능 구현 소개



기능 구현 소개



기능 구현 소개 -Buffer

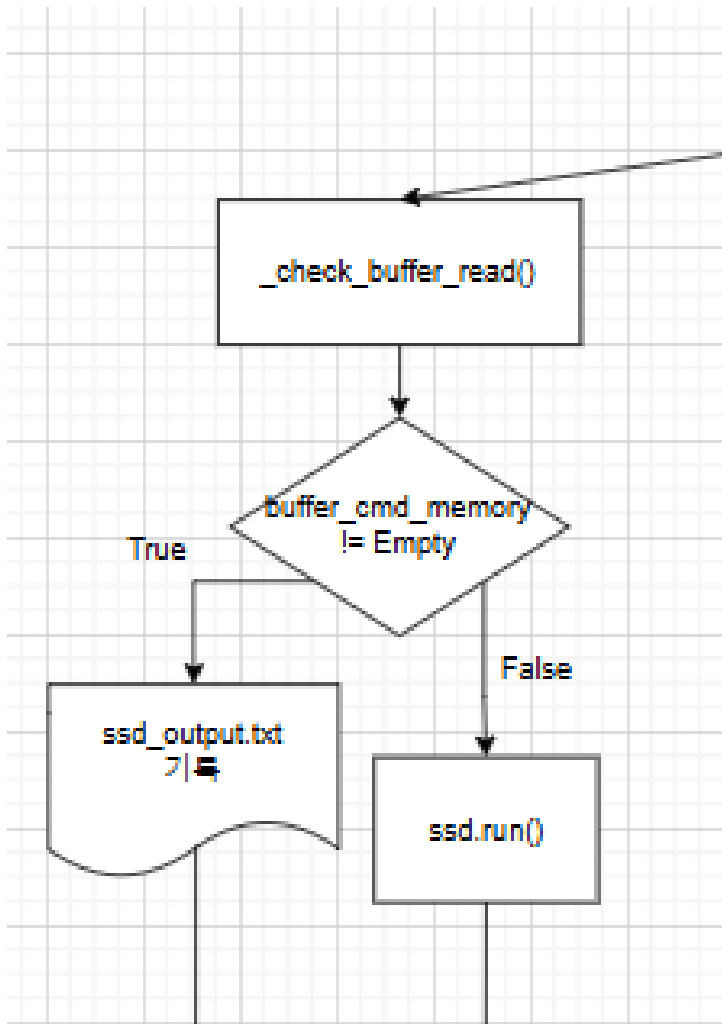
buffer_cmd_memory

LBA (Index)	Command	Value
0	EMPTY	0x00000000
1	EMPTY	0x00000000
2	EMPTY	0x00000000
3	EMPTY	0x00000000
...
98	EMPTY	0x00000000
99	EMPTY	0x00000000

SSD사용을 최소화 하기위해
Buffer의 Command들을 병합하기 용이하도록
buffer_cmd_memory 사용



기능 구현 소개 -Buffer (Read)

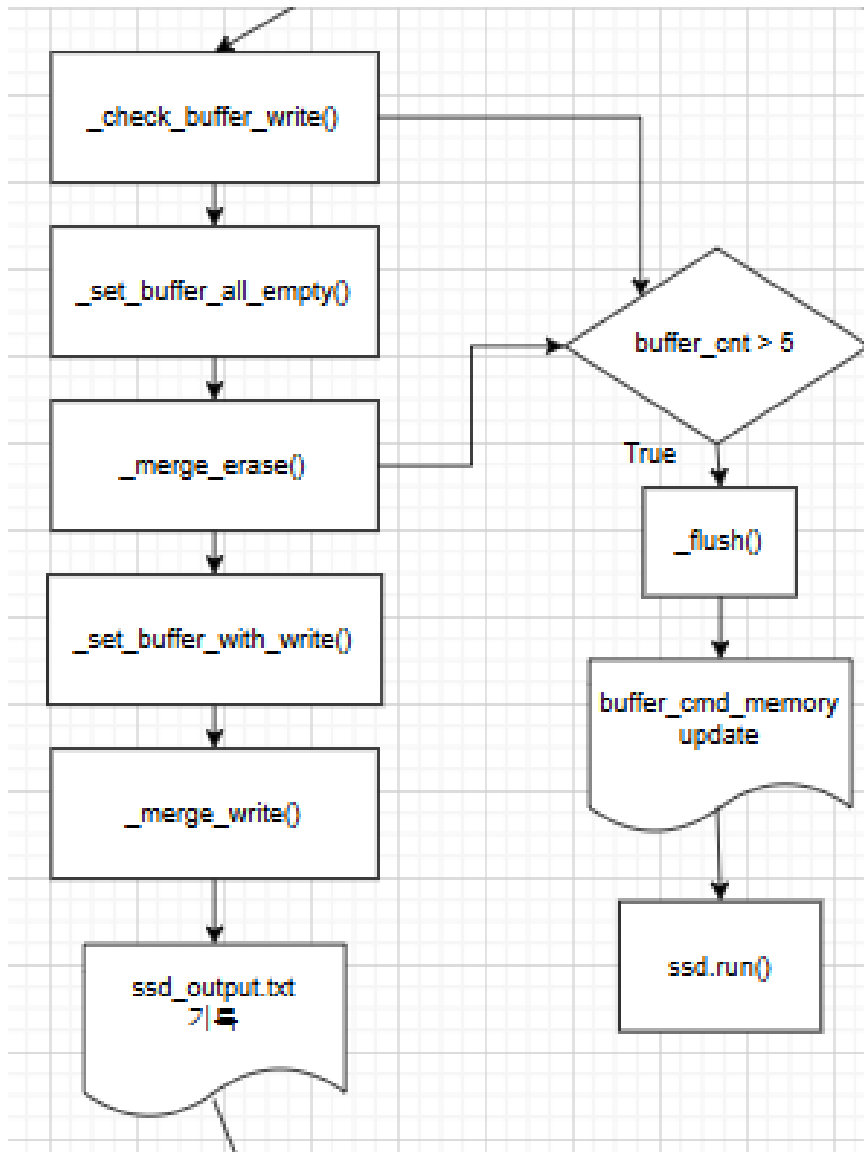


buffer_cmd_memory		
LBA (Index)	Command	Value
0	EMPTY	0x00000000
1	EMPTY	0x00000000
2	WRITE	0x12341234
3	EMPTY	0x00000000
...
98	EMPTY	0x00000000
99	EMPTY	0x00000000

→ READ는 버퍼에 저장 될 필요 없으므로 바로 SSD에서 처리하도록 함

→ 이미 바뀌어 있는 값은 바로 보여주도록 함 (버퍼를 읽는 것이나 변화된 값을 읽는 것이나 동일)

기능 구현 소개 -Buffer (Write)



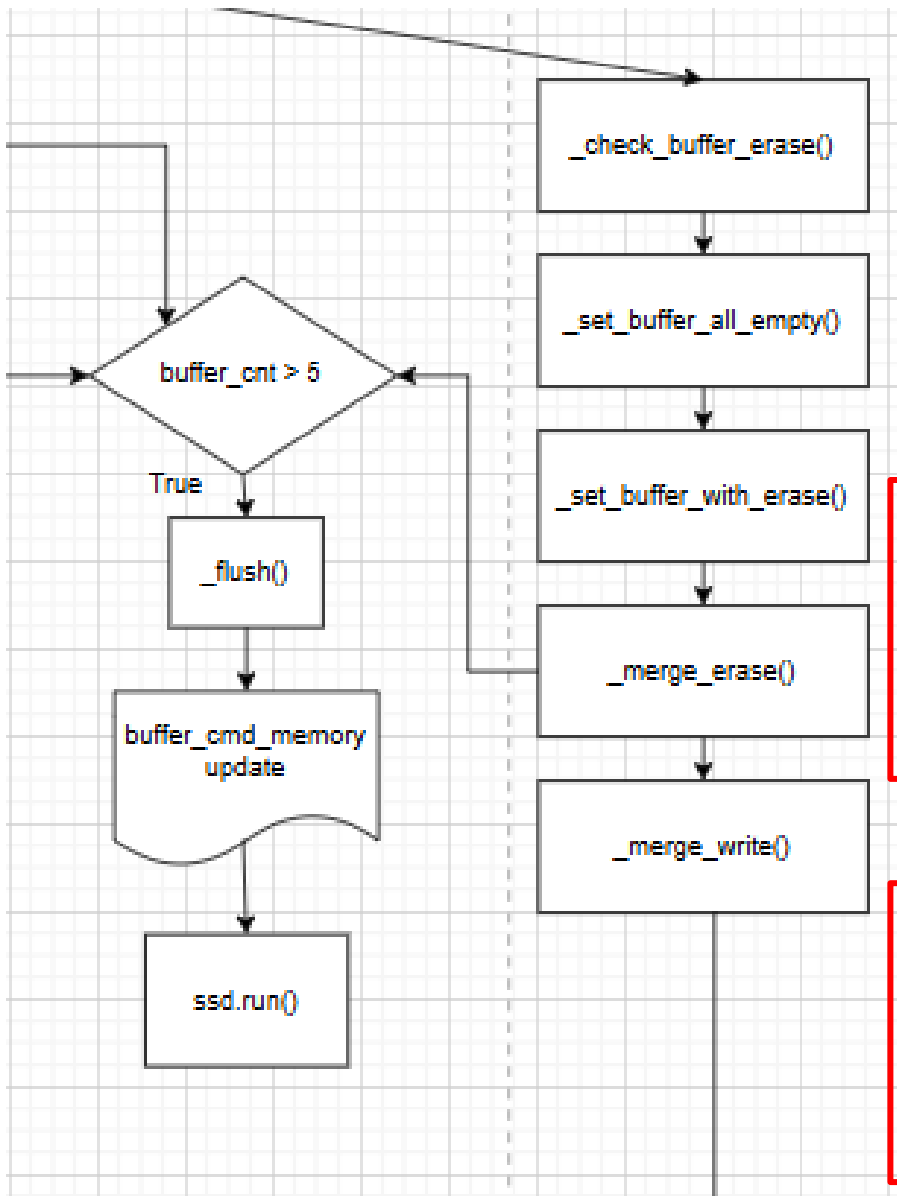
buffer_cmd_memory

LBA (Index)	Command	Value
0	EMPTY	0x00000000
1	EMPTY	0x00000000
2	WRITE	0x12341234
3	EMPTY	0x00000000
...
98	ERASE	0x00000000
99	ERASE	0x00000000

W 2 0x12121212

5개짜리 buffer_list를 비우고
기존에 저장되어 있는 값을 정리해서
buffer_list에 새로 채워 넣음

기능 구현 소개 -Buffer (Erase)



buffer_cmd_memory

LBA (Index)	Command	Value
0	EMPTY	0x00000000
1	EMPTY	0x00000000
2	ERASE	0x00000000
3	ERASE	0x00000000
...
98	ERASE	0x00000000
99	ERASE	0x00000000

E 2 5

5개짜리 buffer_list를 비우고

erase값을 memory에 기록함

저장되어 있는 값을 정리해서
buffer_list에 새로 채워 넣음



TDD 활용 예시



TDD 활용 예시 – Shell.fullread()

RED

```
test_shell.py
22 22 @@ -22,6 +22,12 @@ def test_read_fail(mock):
23 23     with pytest.raises(ValueError, match='ERROR'):
24 24         shell.read(10.1)
25 25 + def test_fullread(capsys):
26 26 +     shell = shell_ftn()
27 27 +     shell.fullread()
28 28 +     captured = capsys.readouterr()
29 29 +     assert captured.out.startswith("[Full Read]")
30 30 +
31 31 def test_write(mock):
32 32     shell = shell_ftn()
33 33     mk = mocker.patch('ssd.write')
34 34
35 41     shell.write(3, '0x000000011')
36 42     mk.call_count == 7
37 43     pass
38 44 +
```

GREEN

```
shell.py
25 25 print("fullwrite")
26 26
27 27 def fullread(self):
28 28     print("fullread")
29 29 + try:
30 30 +     ssd_nand = open("ssd_nand.txt", "r")
31 31 +     ssd_output = open("ssd_output.txt", "w")
32 32 +
33 33 +     print("[Full Read]\n")
34 34 +     for i in range(100):
35 35 +         str = ssd_nand.readline()
36 36 +         words = str.split()
37 37 +         print(f"LBA {words[0]} : {words[1]}\n")
38 38 +
39 39 +     ssd_nand.close()
40 40 +     ssd_output.close()
41 41 + except Exception as e:
42 42 +     raise e
43 43
44 44 def FullWriteAndReadCompare(self):
45 45     print("1_FullWriteAndReadCompare")
```

REFACTOR

```
shell.py
50 54 def fullread(self):
51 55     try:
52 56         ssd_nand = open("ssd_nand.txt", "r")
53 56         ssd_output = open("ssd_output.txt", "w")
54 57
55 58         print("[Full Read]")
56 58         for i in range(100):
57 58             str = ssd_nand.readline()
58 58             words = str.split()
59 58             print(f"LBA {words[0]} : {words[1]}\n")
60 59 +         for idx in range(100):
61 60 +             self.ssd.read_ssd(idx)
62 61 +             ssd_output = open("ssd_output.txt", "r")
63 62 +             str = ssd_output.readline()
64 63 +             print(f"LBA {str.split()[0]} : {str.split()[1]}\n")
65 64
66 65         ssd_nand.close()
67 66         ssd_output.close()
```

TDD 활용 예시 – command._check_input_validity

RED

```
9 def test_ssd_read_error_minus_index():
10     ssd = SSD()
11     with pytest.raises(ValueError, match = "ERROR"):
12         ssd.read(-1)
13     with pytest.raises(ValueError, match = "ERROR"):
14         ssd.read(-10)
15
16 def test_ssd_read_error_index_above_99():
17     ssd = SSD()
18     with pytest.raises(ValueError, match = "ERROR"):
19         ssd.read(100)
20     with pytest.raises(ValueError, match="ERROR"):
21         ssd.read(1000)
22     with pytest.raises(ValueError, match = "ERROR"):
23         ssd.read(10000)
24
25 def test_ssd_read_error_not_digit():
26     ssd = SSD()
27     with pytest.raises(ValueError, match = "ERROR"):
28         ssd.read("abc")
29
30
31 def test_ssd_write_error_minus_index():
32     ssd = SSD()
33     with pytest.raises(ValueError, match = "ERROR"):
34         ssd.write(-1)
35     with pytest.raises(ValueError, match = "ERROR"):
36         ssd.write(-10)
```

GREEN

```
8 def write(self, lba, value):
9     if type(lba) is not int:
10         raise ValueError("ERROR")
11     if lba < 0:
12         raise ValueError("ERROR")
13     if lba >= 100:
14         raise ValueError("ERROR")
15
```

```
8 def write(self, lba, value):
9     if type(lba) is not int:
10         raise ValueError("ERROR")
11     if type(value) is not int:
12         raise ValueError("ERROR")
13     if lba < 0:
14         raise ValueError("ERROR")
15     if lba >= 100:
16         raise ValueError("ERROR")
17     if value < 0:
18         raise ValueError("ERROR")
19     if value >= 0x100000000:
20         raise ValueError("ERROR")
```

REFACTOR

```
9 - if type(lba) is not int:
10 -     raise ValueError("ERROR")
11 + if type(value) is not int:
12 -     raise ValueError("ERROR")
13 - if lba < 0:
14 -     raise ValueError("ERROR")
15 - if lba >= 100:
16 -     raise ValueError("ERROR")
17 - if value < 0:
18 -     raise ValueError("ERROR")
19 - if value >= 0x100000000:
```

```
9 + if not self.check_input_validity(lba, value):
27 + def check_input_validity(self, lba, value):
28 +     if type(lba) is not int:
29 +         return False
30 +     if type(value) is not int:
31 +         return False
32 +     if not 0 <= lba < 100:
33 +         return False
34 +     if not 0 <= value <= 0xFFFFFFFF:
35 +         return False
36 +     return True
```

TDD 활용 예시 - Write()

RED

```
def test_write_success(shell, mocker):
    mocker.patch.object(ssd.SSD, 'write_ssd')
    mocker.patch.object(ssd.SSDOutput, 'read', return_value='')
    mock_print = mocker.patch('builtins.print')

    result = shell.write( num: 3, value: '0x00000000')

    mock_print.assert_called_once()
    assert result is True
```

YoungJun Cho

```
def test_write_fail(shell, mocker):
    mocker.patch.object(ssd.SSD, 'write_ssd')
    mocker.patch.object(ssd.SSDOutput, 'read', return_value='ERROR')
    mock_print = mocker.patch('builtins.print')
    result = shell.write( num: 10, value: '0xABCDEF12')

    mock_print.assert_not_called()
    assert result is False
```

테스트 기반 리팩토링

GREEN

```
def write(self, num: int, value: str) -> None:
    my_ssd = SSD()
    my_ssd.write_ssd(num, int(value,16))
    my_ssdoutput = SSDOutput()
    if my_ssdoutput.read() == '':
        print('[Write] Done')
    return True
```

```
class WriteCommand(Command):
    def __init__(self, shell, idx: int, value: int):
```

```
def write(self, idx, value):
    if idx<0 or idx>99:
        raise ValueError("ERROR")
    if len(value)>8:
        raise ValueError("ERROR")
    print(idx, value)
```

```
def write(self, num: int, value: int) -> None:
    if self.ssd.write_ssd(num, value):
```

REFACTOR

```
def write(self, num: int, value: str) -> None:
    self.ssd.write_ssd(num, int(value,16))
    if self.ssd_output.read() == '':
        print('[Write] Done')
    return True
    return False
```

네이밍 변경 및 내부 메소드화

```
class ShellWriteCommand(Command):
    def __init__(self, shell, idx: int, value: int):
```

클래스 추출

jihun75.kim

```
def __init__(self, shell, idx: int, value: int):
    super().__init__(shell)
    self.idx = idx
    self.value = value
```

5 usages jihun75.kim

```
def execute(self) -> bool:
    self.shell._send_command('W', self.idx, self.value)
    if self.shell.ssd_output.read() == '':
        print('[Write] Done')
        self.shell.logger.print(f"{self.execute.__qualname__}()", "DONE")
        return True
    self.shell.logger.print(f"{self.execute.__qualname__}()", "FAIL")
    return False
```

TDD 활용 예시 – PartialLBWrite()

RED

```
def test_PartialLBWrite_pass(shell, mocker):
    mocker.patch('random.randint', return_value=12345678)
    mock_write_ssd = mocker.patch.object(shell.ssd, 'write_ssd')
    mock_print = mocker.patch('builtins.print')
    shell.PartialLBWrite()

    assert mock_write_ssd.call_count == 150
    mock_print.assert_called_with('PASS')

    YoungJun Cho

def test_PartialLBWrite_fail(shell, mocker):
    mocker.patch('random.randint', return_value=12345678)
    mocker.patch.object(shell.ssd, 'write_ssd')
    mocker.patch.object(shell.ssd_nand, 'readline', side_effect=['
    mock_print = mocker.patch('builtins.print')
    result = shell.PartialLBWrite()

    mock_print.assert_called_with('FAIL')
    assert result is None
```

테스트 기반 리팩토링

GREEN

```
def PartialLBWrite(self):
    for i in range(30):
        r1 = random.randint(0, 0xFFFFFFFF)
        self.write(4, r1)
        self.write(0, r1)
        self.write(3, r1)
        self.write(1, r1)
        self.write(2, r1)
```

네이밍 변경, 반복문 사용, 메서드 사용으로
리팩토링

REFACTOR

```
def PartialLBWrite(self):
    partialLBA_index_list = [4, 0, 3, 1, 2]
    for _ in range(30):
        for write_index in range(5):
            random_write_value = random.randint(0, 0xFFFFFFFF)
            self.write(partialLBA_index_list[write_index], random_write_value)
        check_read_value = self.read(0)
```

팩토리 패턴 적용으로 클래스 추출

```
3 usages YoungJun Cho +3
def PartialLBWrite(self):
    partialLBA_index_list = [4, 0, 3, 1, 2]
    for _ in range(30):
        random_write_value = random.randint(a: 0, b: 0xFFFFFFFF)
        for x in range(5):
            self.ssd.write_ssd(partialLBA_index_list[x], random_write_value)
        check_ref = self.ssd_nand.readline(0).split()[1]
        for x in range(1, 5):
            if check_ref != self.ssd_nand.readline(x).split()[1]:
                print('FAIL')
                self.logger.print(header: f"{self.read.__qualname__}()", message: "FAIL")
                return
        print("PASS")
    self.logger.print(header: f"{self.read.__qualname__}()", message: "PASS")
```

```
class PartialLBWriteCommand(Command):
    YoungJun Cho
    def __init__(self, shell):
        super().__init__(shell)
    3 usages YoungJun Cho
    def execute(self):
        partialLBA_index_list = [4, 0, 3, 1, 2]
        for _ in range(30):
            random_write_value = random.randint(a: 0, b: 0xFFFFFFFF)
            for x in range(5):
                self.shell.send_command('W', partialLBA_index_list[x], random_write_value)
            check_ref = self.shell.ssd_nand.readline(0).split()[1]
            for x in range(1, 5):
                if check_ref != self.shell.ssd_nand.readline(x).split()[1]:
                    print('FAIL')
                    self.shell.logger.print(f"{self.execute.__qualname__}()", "FAIL")
                    return
            print("PASS")
        self.shell.logger.print(f"{self.execute.__qualname__}()", "PASS")
```

TDD 활용 예시 – Test Coverage 결과

Coverage report: 99%

Files

Functions

Classes

coverage.py v7.9.2, created at 2025-07-18 12:46 +0900

File ▲	statements	missing	excluded	coverage
<u>buffer.py</u>	140	0	0	100%
shell.py	261	6	0	98%
ssd_commands.py	113	0	0	100%
ssd_interface.py	17	2	0	88%
ssd_texts.py	41	0	0	100%
ssd.py	34	2	0	94%
test_buffer.py	181	0	0	100%
test_shell.py	191	0	0	100%
test_ssd.py	175	0	0	100%
Total	1153	10	0	99%

coverage.py v7.9.2, created at 2025-07-18 12:46 +0900

- ✓ TDD를 통하여 Code Coverage 99% 달성
(Abstract method 선언부는 missing으로 체크 됨)



Mocking 활용 예시



Mocking 활용 예시 – test_shell.py

✓ pytest fixture 사용 : 깔끔하고 재사용 가능

✓ 중요 method들 mocker 적용

```
10 class Test_shell:
11     @pytest.fixture
12     def setup_shell(self, mocker):
13         self.shell = Shell()
14         self.get_response = mocker.patch.object(self.shell, 'get_response')
15         self.get_response_value = mocker.patch.object(self.shell, 'get_response_value')
16         self.mock_print = mocker.patch('builtins.print')
17         self.rand_num = mocker.patch('random.randint', return_value=12345678)
18         self.shell.send_command = mocker.Mock()
```

```
20 @pytest.fixture
21 def setup_ssdinterface(self, mocker):
22     self.shell = Shell()
23     self.shell.ssd_interface = mocker.Mock()
25 @pytest.mark.parametrize('command', ['E', 'W', 'R'])
```

```
26 def test_send_command(self, setup_ssdinterface, command):
27     self.shell.ssd_interface.run.return_value = 'OK'
28     result = self.shell.send_command(command, lba: 10, value: 10)
29
30     self.shell.ssd_interface.run.assert_called_once()
31     assert result == 'OK'
```

Act

Assert

```
33 def test_response(self, setup_ssdinterface):
34     self.shell.ssd_interface.get_response.return_value = 'RESPONSE'
35     assert self.shell.get_response() == 'RESPONSE'
36     self.shell.ssd_interface.get_response.assert_called_once()
```

Act

Assert

✓ send_command mock 처리

✓ return_value를 활용한 stub 처리

Mocking 활용 예시 – test_ssd.py

테스트 목표: LBA가 유효 범위(0-99)를 초과했을 때 SSD가 예외를 던지고, "ERROR"를 출력
파일에 기록하는지 검증

테스트 파라미터로 잘못된 LBA 준비,
SSD 객체는 fixture에서 생성됨

SSD에 잘못된 명령어 실행 시 예외
발생되는지 확인

- 예외 메시지가
ERROR_MESSAGE인지 검증
- ssd_output.txt 파일에 기록된
내용도 ERROR_MESSAGE 인지
검증

Arrange

Act

Assert

```
@pytest.fixture
def ssd():
    return SSD(True)

@pytest.mark.parametrize("lba, value", [(100, 0x00000000), (1000, 0x00000000), (10000, 0x00000000)])
def test_ssd_write_error_index_above_99(ssd, lba, value):
    with pytest.raises(ValueError, match=ERROR_MESSAGE):
        ssd.run([None, 'W', lba, dec_to_hex(value)])
    output_txt = SSDOutput()
    assert output_txt.read() == ERROR_MESSAGE
```

Mocking 활용 예시 – test_ssd.py

테스트 목표: F(flush) 명령이 buffer 내부에 쌓인 명령을 올바르게 실행하여 NAND에 기록되는지 검증

SSD, SSDNand 객체 준비
Buffer.run()을 stubbing하여 flush할
명령을 강제로 설정

flush 명령 실행

flush 결과가 실제 NAND에
기록되었는지 검증

Arrange

Act

Assert

```
@patch('buffer.Buffer.run')
def test_flush_success(mock_buffer_run):
    ssd = SSD()
    ssd_nand = SSDNand()
    mock_buffer_run.return_value = [[None, 'W', 1, '0x1234ABCD']]

    ssd.run([None, 'F'])

    assert ssd_nand.read()[1] == "01 0x1234ABCD\n"
```

Mocking 활용 예시 – test_buffer.py

테스트 목표: 버퍼에 쌓인 명령들이 flush() 호출 시 모두 정상 처리되는지 확인하고 명령 처리 후 버퍼 상태가 초기화되는지 검증

```
@pytest.fixture
def buffer(mockers: Mockers):
    mocker.patch("os.makedirs")
    mocker.patch("os.path.exists", return_value=True)
    mocker.patch("os.listdir", return_value=[])
    mocker.patch("builtins.open", mocker.mock_open())
    mocker.patch("buffer.SSDOutput", return_value=mocker.Mock(write=mocker.Mock()))
    buffer = Buffer()
    buffer._buf_lst = ['1_empty', '2_empty', '3_empty', '4_empty', '5_empty']
    buffer._buffer_cnt = 0
    return buffer
```

5개의 명령이 들어있는 버퍼를 미리 설정하고 flush에서 호출될 함수를 mock 처리

flush(5)를 호출함으로써 5개의 명령을 처리하고 버퍼를 비워야 함.

함수가 5번 호출되었는지, 버퍼 리스트가 초기화되었는지, 명령 개수가 0으로 줄었는지를 검증

Arrange

Act

Assert

```
def test_flush_calls_remove_buffer_and_put_run_command(buffer: Buffer, mocker: Mockers):
    buffer._buf_lst = [
        "1_W_1_0x00000001",
        "2_E_2_3",
        "3_W_3_0x00000003",
        "4_E_4_1",
        "5_W_5_0x00000005"
    ]
    buffer._buffer_cnt = 5

    mock_remove = mocker.patch.object(buffer, '_remove_buffer_and_put_run_command')

    buffer._flush(5)

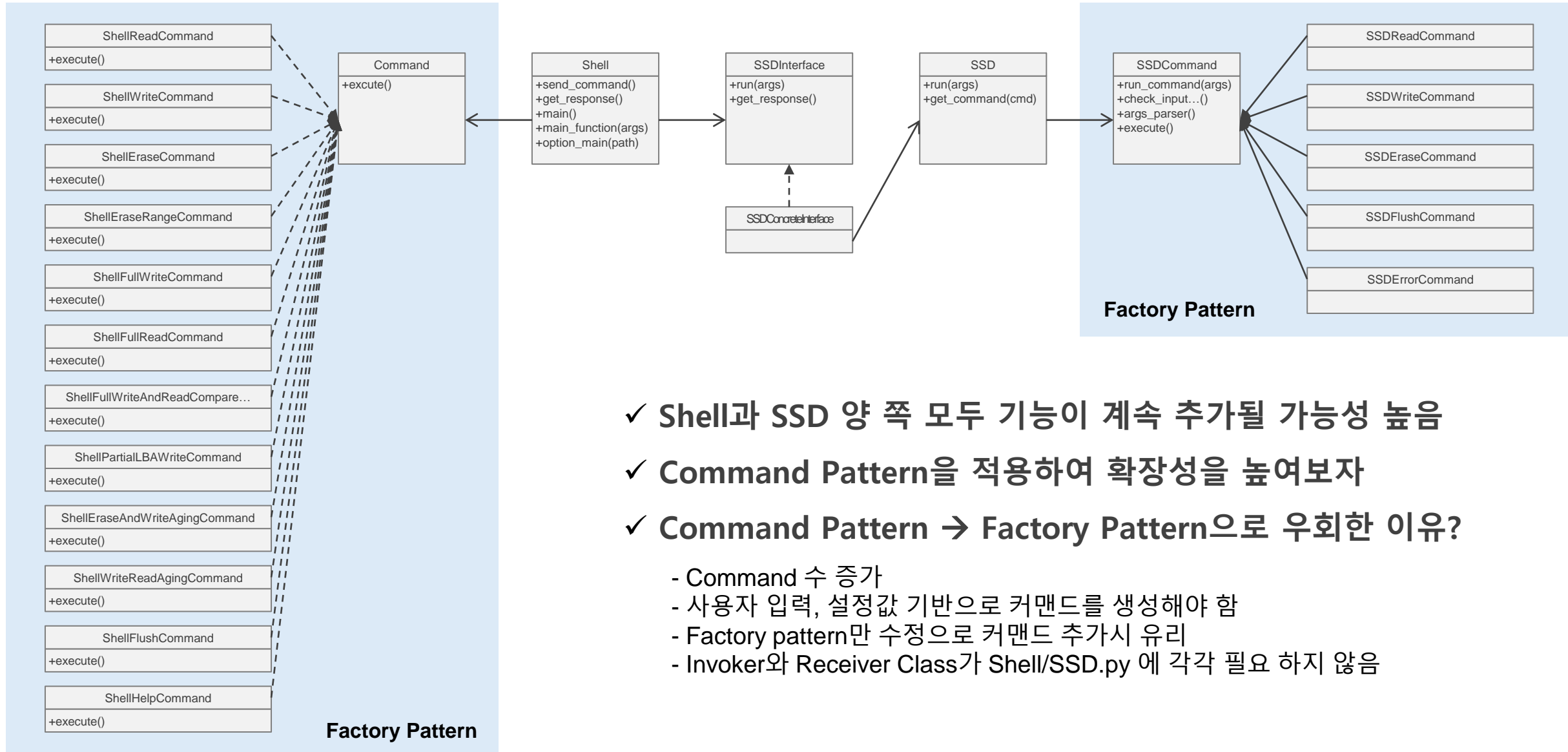
    assert mock_remove.call_count == 5
    assert buffer._buf_lst == ['1_empty', '2_empty', '3_empty', '4_empty', '5_empty']
    assert buffer._buffer_cnt == 0
```



디자인 패턴 활용 예시



디자인 패턴 활용 예시 – Factory Pattern



- ✓ Shell과 SSD 양 쪽 모두 기능이 계속 추가될 가능성 높음
- ✓ Command Pattern을 적용하여 확장성을 높여보자
- ✓ Command Pattern → Factory Pattern으로 우회한 이유?
 - Command 수 증가
 - 사용자 입력, 설정값 기반으로 커맨드를 생성해야 함
 - Factory pattern만 수정으로 커맨드 추가시 유리
 - Invoker와 Receiver Class가 Shell/SSD.py 에 각각 필요 하지 않음

디자인 패턴 활용 예시 – Factory Pattern

기존 shell.py 구조

Shell() 클래스 내부
def write()
def read()
...

```
self.command_dictionary(args)[(args[0], len(args))]()  
#self.command_dictionary(args)[(args[0], len(args))]()  
shellCommand : Command = self.command_dictionary(args)[(args[0], len(args))]()  
shellCommand.execute()
```

Shell() 클래스

Command(ABC)
클래스

ReadCommand(shell) 클래스

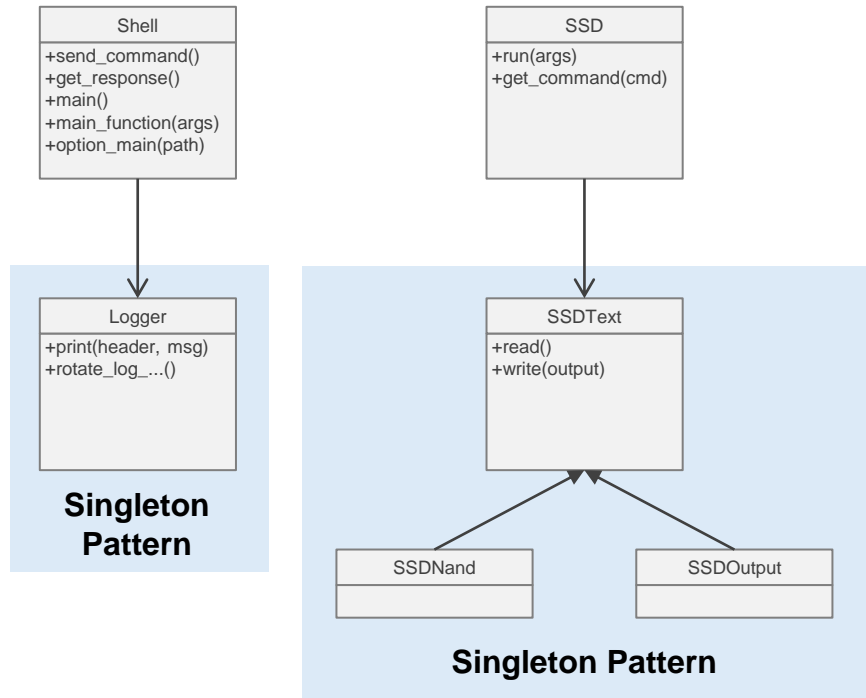
WriteCommand(shell) 클래스

```
class Shell():  
    seunguk-park +1  
    def __init__(self):  
        self.ssd = SSD()  
        self.ssd_output = SSDOutput()  
        self.ssd_nand = SSDNand()  
        self.logger = Logger()  
  
25 usages (8 dynamic)  jjaamm.lee +3  
    def read(self, idx: int) -> None:  
        self.ssd.read_ssd(idx)  
        result = self.ssd_output.read()  
        value = result.split()[1]  
        print(f'[Read] LBA {idx}: {value}')        self.logger.print( header: f"{self.read..."
```

```
51 + class Command(ABC):  
52 +     def __init__(self, shell):  
53 +         self.shell = shell
```

```
class ReadCommand(Command):  
    jihun75.kim  
    def __init__(self, shell, idx):  
        super().__init__(shell)  
        self.idx = idx  
  
2 usages  jihun75.kim  
    def execute(self) -> None:  
        self.shell._send_command('R', self.idx)  
        result = self.shell.ssd_output.read()  
        value = result.split()[1]  
        print(f'[Read] LBA {self.idx}: {value}')        self.shell.logger.print(f"{self.execute..."
```

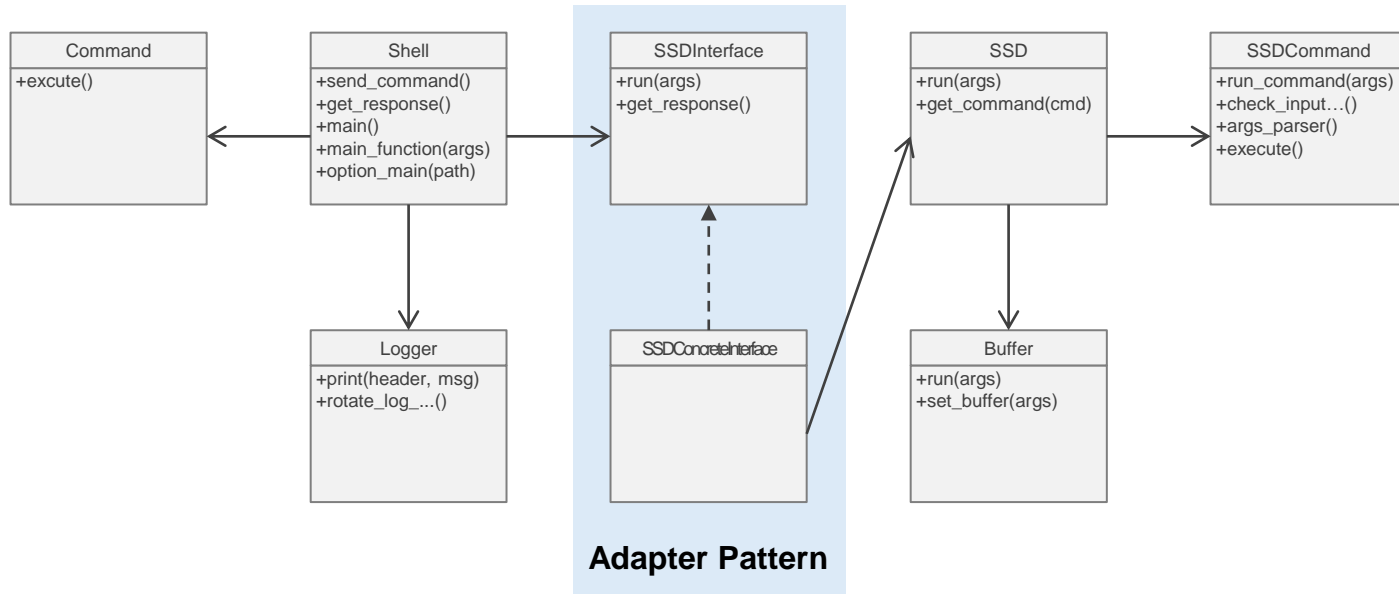

디자인 패턴 활용 예시 – Singleton Pattern



```
class Logger:
+   _instance = None
+   LOG_FILE = 'latest.log'
+   MAX_SIZE = 10 * 1024 # 10KB
+   def __new__(cls, *args, **kwargs):
+       if cls._instance is None:
+           cls._instance = super().__new__(cls)
+           cls._instance._initialized = False
+       return cls._instance
+
+   def __init__(self):
+       pass
+       if self._initialized:
+           return
+       self._initialized = True
+       if os.path.exists(self.LOG_FILE):
+           os.remove(self.LOG_FILE)
```

- ✓ Log File, Text File 모두 하나의 파일을 여러 군데에서 접근
- ✓ Singleton Pattern을 적용

디자인 패턴 활용 예시 – Adapter Pattern



- ✓ 여러 SSD 제품을 하나의 Test Shell 프로그램으로 테스트할 수 있도록
- ✓ Shell-SSD간 변경에 의한 영향도를 최소화
- ✓ SSD Concrete Interface 가 변경 되거나, 새로운 Concrete Interface가 추가 되더라도 Shell의 수정사항은 발생하지 않도록 Adapter pattern 활용



리팩토링을 통한 클린코드 전후 결과 비교



리팩토링을 통한 클린코드 전후 결과 비교

➤ 변수명의 구체화, 통일화 (idx vs lba / output vs value vs result 등)

```
115 shell.py

92 93 class ShellEraseCommand(Command):
93 - def __init__(self, shell, lba: int, size: int):
94 + def __init__(self, shell, st_lba: int, erase_size: int):
94 95     super().__init__(shell)
95 - self.lba = lba
96 - self.size = size
96 + self.st_lba = st_lba
97 + self.erase_size = erase_size

97 98
98 99 def execute(self):
99 - if (0 > self.lba or self.lba > 99) or (1 > self.size or self.size > 100) or (self.lba + self.size > 100):
100 + if (0 > self.st_lba or self.st_lba > 99) or (1 > self.erase_size or self.erase_size > 100) or (self.st_lba + self.erase_size > 100):
100 101     self.shell.logger.print(f"{self.execute.__qualname__}()", f"FAIL")
101 102     raise Exception()
102 103
103 104     offset = 0
104 - while self.size > 0:
105 -     erase_size = min(self.size, 10)
106 -     self.shell.send_command('E', self.lba + offset, erase_size)
105 + while self.erase_size > 0:
106 +     erase_size = min(self.erase_size, 10)
107 +     self.shell.send_command('E', self.st_lba + offset, erase_size)
107 108     offset += 10
108 - self.size -= erase_size
109 + self.erase_size -= erase_size
110 +
109 111     caller_frame = inspect.stack()[4]
110 112     caller_name = caller_frame.code_context
```

✓ idx vs lba 로 변수 통일

✓ output vs value vs result 로 변수 통일

✓ size → erase_size 변수 구체화

리팩토링을 통한 클린코드 전후 결과 비교

➤ 변수명의 구체화, 통일화 (idx vs lba / output vs value vs result 등)

```
115 shell.py
234 230 class ShellWriteReadAgingCommand(Command):
235 231     def __init__(self, shell):
236 232         super().__init__(shell)
237 233
238 234     def execute(self):
239 -         value = random.randint(0, 0xFFFFFFFF)
240 -         for i in range(200):
241 -             self.shell.send_command('W', 0, value)
242 -             self.shell.send_command('W', 99, value)
235 +         random_value = random.randint(0, 0xFFFFFFFF)
236 +         for _ in range(200):
237 +             self.shell.send_command('W', 0, random_value)
238 +             self.shell.send_command('W', 99, random_value)
243 239
244 240         self.shell.send_command('R', 0)
245 -         check_ref = self.shell.get_response_value()
241 +         ref_value = self.shell.get_response_value()
246 242
247 243         self.shell.send_command('R', 99)
248 -         check_comp = self.shell.get_response_value()
244 +         comp_value = self.shell.get_response_value()
249 245
250 -         if check_ref != check_comp:
246 +         if ref_value != comp_value:
251 247             print('FAIL')
252 248             self.shell.logger.print(f"{self.execute.__qualname__}()", "FAIL")
253 249             return
250 +
254 251         print('PASS')
255 252         self.shell.logger.print(f"{self.execute.__qualname__}()", "PASS")
```

✓ lba를 의미하는 loop: lba으로 사용

✓ 단순 횟수를 의미하는 loop는 index 사용 여부에 따라 i or _ 사용

✓ 비교 변수는 ref_value, comp_value로 통일

✓ value → random_value 변수 구체화

리팩토링을 통한 클린코드 전후 결과 비교

```
- class ReadCommand(Command):
```

```
+ class ShellReadCommand(Command):
```

```
    def __init__(self, shell, idx):
```

✓SSD의 클래스명과 구분을 위한 naming변경

```
for lba in range(100):
```

```
    try:
```

```
        self.shell.send_command('R', lba)
```

```
        value = self.shell._get_response_value()
```

```
        value = self.shell.get_response_value()
```

```
    except Exception as e:
```

```
        hex_str = f"0x{rand_num:08X}"
```

```
        self.shell.send_command('W', start_idx + x, rand_num)
```

```
        self.shell.send_command('R', start_idx + x)
```

```
        if self.shell._get_response_value() != hex_str:
```

```
            if self.shell.get_response_value() != hex_str:
```

✓메서드를 외부에서 사용하게 되어 naming 변경

리팩토링을 통한 클린코드 전후 결과 비교

➤ 함수 추출, 함수 기능 간소화 (한 개의 함수에서는 가능한 한 가지 기능만 담당하도록)

1 file changed +35 -34 lines changed

buffer.py

```
100 - flushed = False
101 - while len(self.buf_lst) > 5:
102 -     flushed = True
103 -     for i in range(5):
104 -         _, flushed_cmd, flushed_lba, flushed_value = self.buf_lst[i].split("_")
105 -         flushed_lba = int(flushed_lba)
106 -         if flushed_cmd == "W":
107 -             self.command_memory[int(flushed_lba)] = EMPTY
108 -         if flushed_cmd == "E":
109 -             flushed_value = int(flushed_value)
110 -             for flush_erase_idx in range(flushed_lba,
111 flushed_lba+flushed_value):
112                 self.command_memory[int(flush_erase_idx)] = EMPTY
113                 self._run_command.append([None, flushed_cmd, flushed_lba,
114 flushed_value])
115 -
116 - self.buf_lst = self.buf_lst[5:]
```

```
100 + self.flush(5)
101 +
102 + def flush(self, cnt):
103 +     for i in range(cnt):
104 +         self.remove_buffer_and_put_run_command(i)
105 +         self.buf_lst = self.buf_lst[cnt:]
106 +         self.rename_buffer_to_start_1()
107 +         while len(self.buf_lst) < cnt:
```

- ✓ 기존: 버퍼를 탐색하는 함수에 flush기능이 모두 들어가 있음
- ✓ 변경: flush기능을 Extract Method하여 기존 함수에서 추출
- ✓ 기존: flush기능은 버퍼가 5개 초과하는 경우만 작동하여 flush기능만 요청받는 경우 처리할 수 없었음
- ✓ 변경: flush기능이 특정 조건을 만족하지 않아도 작동하도록 cnt를 인자로 하는 함수로 추출

리팩토링을 통한 클린코드 전후 결과 비교

➤ 함수 추출, 함수 기능 간소화 (한 개의 함수에서는 가능한 한 가지 기능만 담당하도록)

buffer.py

+130 -95

40

41

42

43

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

```
def run(self, sys_argv):  
    self.set_buffer(sys_argv)  
    cmd = sys_argv[1]  
    if cmd == "W":  
        value = int(sys_argv[3], 16)  
        self.command_memory[lba] = WRITE  
        self.value_memory[lba] = value  
        self.output_txt.write("")  
    if cmd == "E":  
        size = int(sys_argv[3])  
        for erase_lba in range(lba, lba+size):  
            self.command_memory[erase_lba] = ERASE  
            self.value_memory[erase_lba] = EMPTY_VALUE  
    #update buffer  
    prev_command = EMPTY  
    start_lba = -1  
    erase_size = 0  
    for memory_lba, saved_command in enumerate(self.command_memory):  
        if saved_command == WRITE:  
            self.buf_lst.append(f"  
{len(self.buf_lst)+1}_W_{memory_lba}_0x{self.value_memory[memory_lba]:08X}")  
        if prev_command == ERASE:  
            self.buf_lst.append(f"  
{len(self.buf_lst)+1}_E_{start_lba}_{erase_size}")  
        prev_command = WRITE  
    continue
```

71

72

73

74

75

76

77

78

79

80

81


82

83


```
def run(self, sys_argv):  
    self._run_command = []  
    cmd = sys_argv[1]  
    if cmd == 'R':  
        self._check_buffer_read(sys_argv)  
    elif cmd == 'W':  
        self._check_buffer_write(sys_argv)  
    elif cmd == 'E':  
        self._check_buffer_erase(sys_argv)  
    else:  
        self._flush(self._buffer_cnt)
```

✓ 기존: 어떤 명령어가 오더라도 set_buffer라는 함수에서 모든 처리를 함

✓ 변경: 명령어에 따라 buffer가 작동하는 법이 다른 만큼 함수를 분리하여 처리하도록 함




소감




소감...

조영준	클린코드에 대해 많이 배울 수 있었고, 현업에서 잘 활용할 수 있어 보입니다.
민동학	협업에 도움이 되는 많은 것들을 배울 수 있었고, 리팩토링 및 git 사용에 대해 정말 많이 배워갈 수 있었습니다.
박승욱	여태 git의 기능을 절반도 활용하지 못하고 있었다는 걸 깨달았습니다. 현업에 돌아가면 써먹을 수 있는 실용적인 내용을 많이 배운 귀중한 시간이었습니다.
이재원	unittest는 처음 사용해보는데 매우 유용한 기능이라고 생각이 되어 앞으로 개발할 때 tdd로 개발하면 좋을 것 같다는 생각을 했습니다. 훌륭한 팀원들을 만나 많이 배우고 재미있게 프로젝트 할 수 있었습니다.
최일묵	Unit test를 만들면서 개발을 진행하는 것이 귀찮게 느껴질 수 있으나, 프로젝트의 형상 관리를 위하여 Unit test를 꾸준히 생성/진행하는 것이 중요함을 느꼈습니다.
한재원	다양한 리팩토링의 방법부터 프로젝트를 통한 실전 실습까지 하는 귀중한 경험을 했습니다.



Q&A



감사합니다.