Create your value

CRA Project – SSD 프로젝트

A-Teuk

2025.07.18

황웅범, 이민호, 최새롬,
박소정, 홍승표, 이준태

목차

조원 소개 및 역할 담당

황웅범님	SSD Feature Developer 슈퍼 긍정으로 항상 팀의 분위기를 밝게 만들어 주는 리더십 마스터
이민호님	SSD Feature Developer Clean Code의 아버지로 항상 Clean한 코드만 작성하는 LGTM 수집가, Clean Code 마스터
최새롬님	Shell Feature Developer Refactoring 마스터로 Feature 구현 & Refactoring 모두 훌륭한 실력을 가진 코딩 마스터
박소정님	SSD Feature Developer Design Pattern 마스터로 SW의 확장성을 고려해 적합한 Pattern을 적용하는 디자인 마스터
홍승표님	Shell Feature Developer Code Review 마스터로 팀원들에게 항상 새로운 관점을 제시하는 CR 마스터
이준태님	Shell Feature Developer TDD 마스터로 조기 버그 발견과 높은 Code Coverage를 유지하는 Test 마스터

기능 구현 소개

압도적인
커버리지

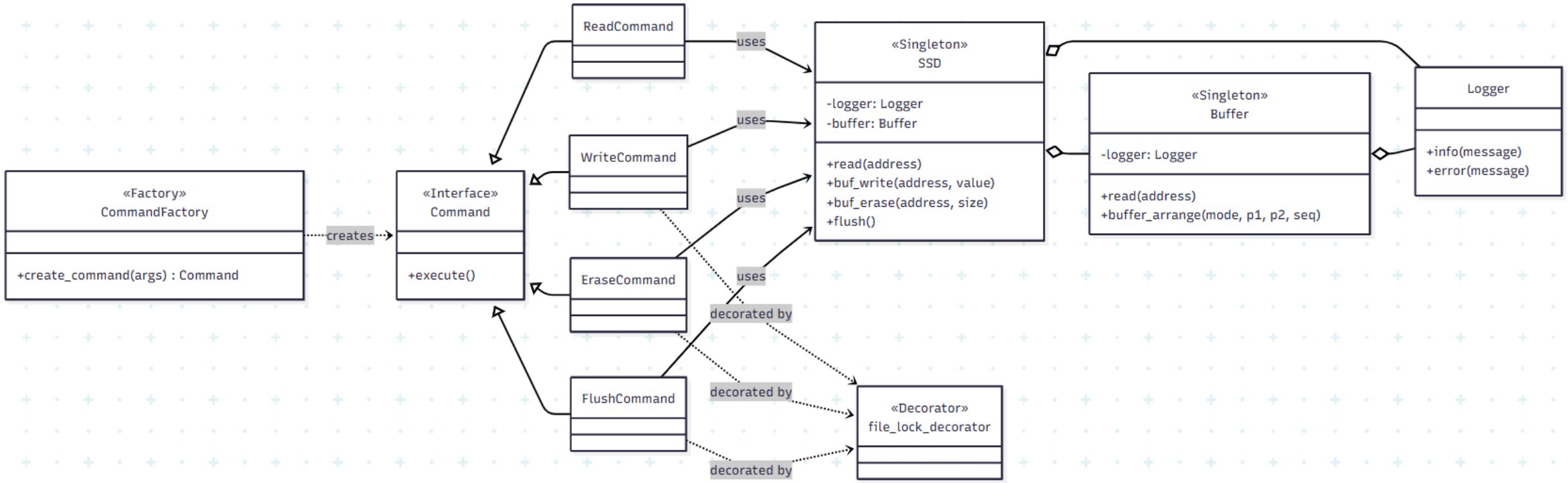
Coverage report: 98%

Files | Functions | Classes

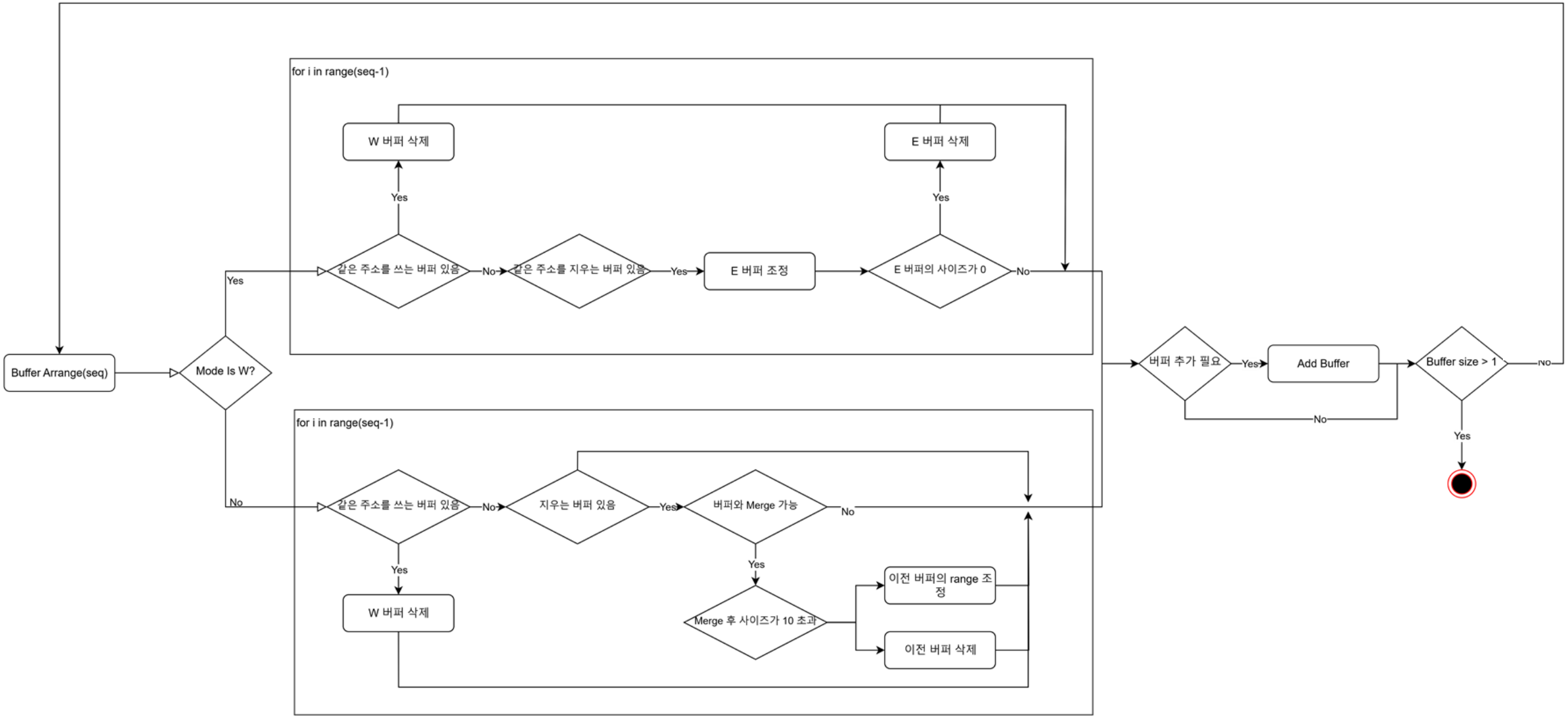
coverage.py v7.9.1, created at 2025-07-18 13:21 +0900

File	*statements* ▼	*missing*	*excluded*	*coverage*
ssd.py	206	8	0	96%
ssd_tool\buffer.py	157	15	0	90%
tests\shell\test_erase_range.py	100	0	0	100%
tests\shell\test_erase.py	100	0	0	100%
shell_tool\shell_logger.py	79	6	0	92%
commands\mixin.py	76	10	0	87%
shell_tool\shell_constants.py	74	0	0	100%
tests\shell\script\test_script1.py	72	0	0	100%
shell.py	67	5	0	93%
commands\base.py	66	1	0	98%
tests\shell\test_shell_rev.py	64	0	0	100%
tests\test_ssd\conftest.py	64	1	0	98%
tests\shell\script\test_script3.py	59	0	0	100%

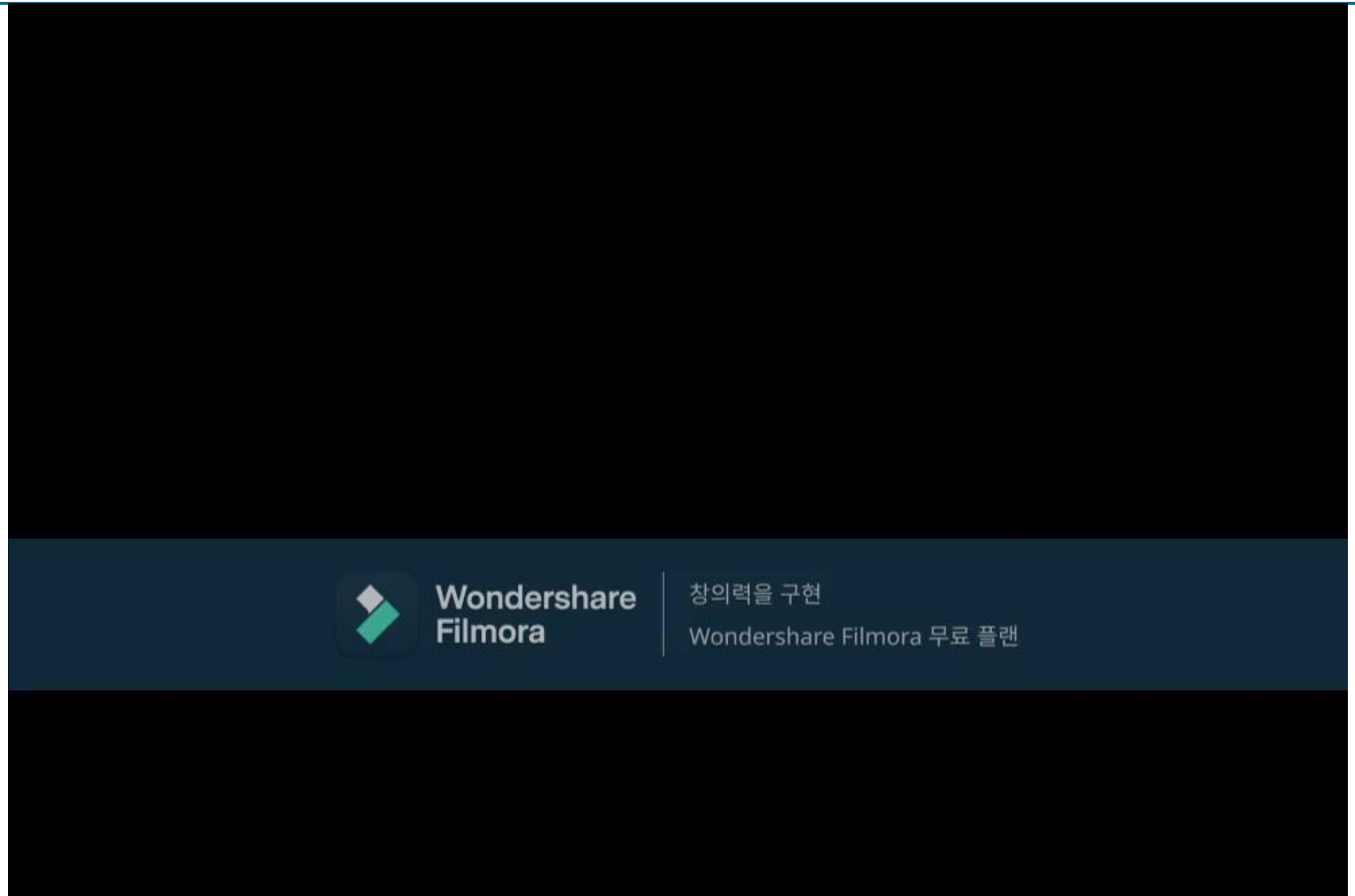
기능 구현 소개 - SSD


«Factory»
CommandFactory
+create_command(args) : Command
creates
«Interface»
Command
+execute()
ReadCommand
WriteCommand
EraseCommand
FlushCommand
uses
uses
uses
uses
«Singleton»
SSD
-logger: Logger
-buffer: Buffer
+read(address)
+buf_write(address, value)
+buf_erase(address, size)
+flush()
«Singleton»
Buffer
-logger: Logger
+read(address)
+buffer_arrange(mode, p1, p2, seq)
Logger
+info(message)
+error(message)
decorated by
decorated by
decorated by
«Decorator»
file_lock_decorator

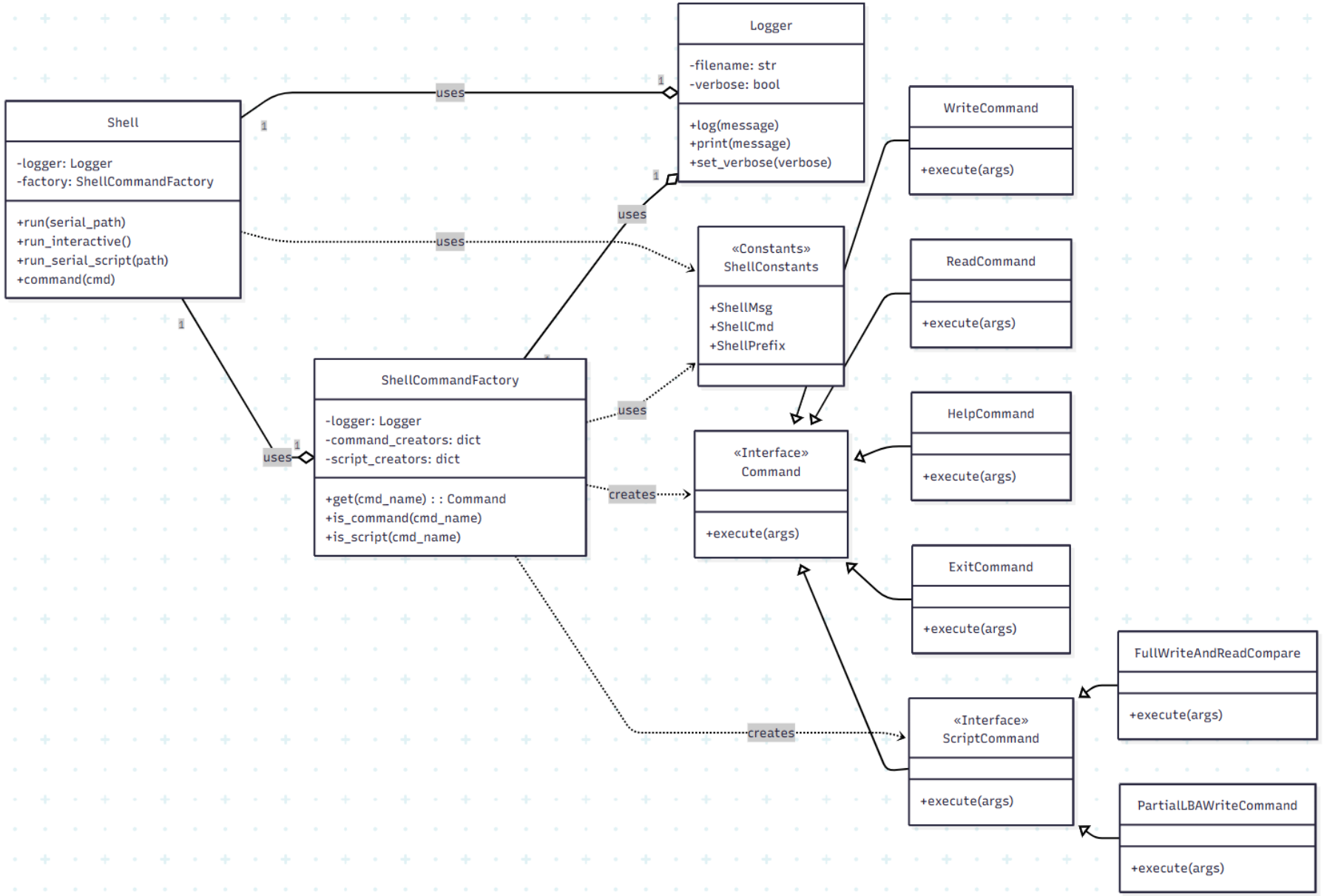
기능 구현 소개 - SSD(Buffer Algorithm)



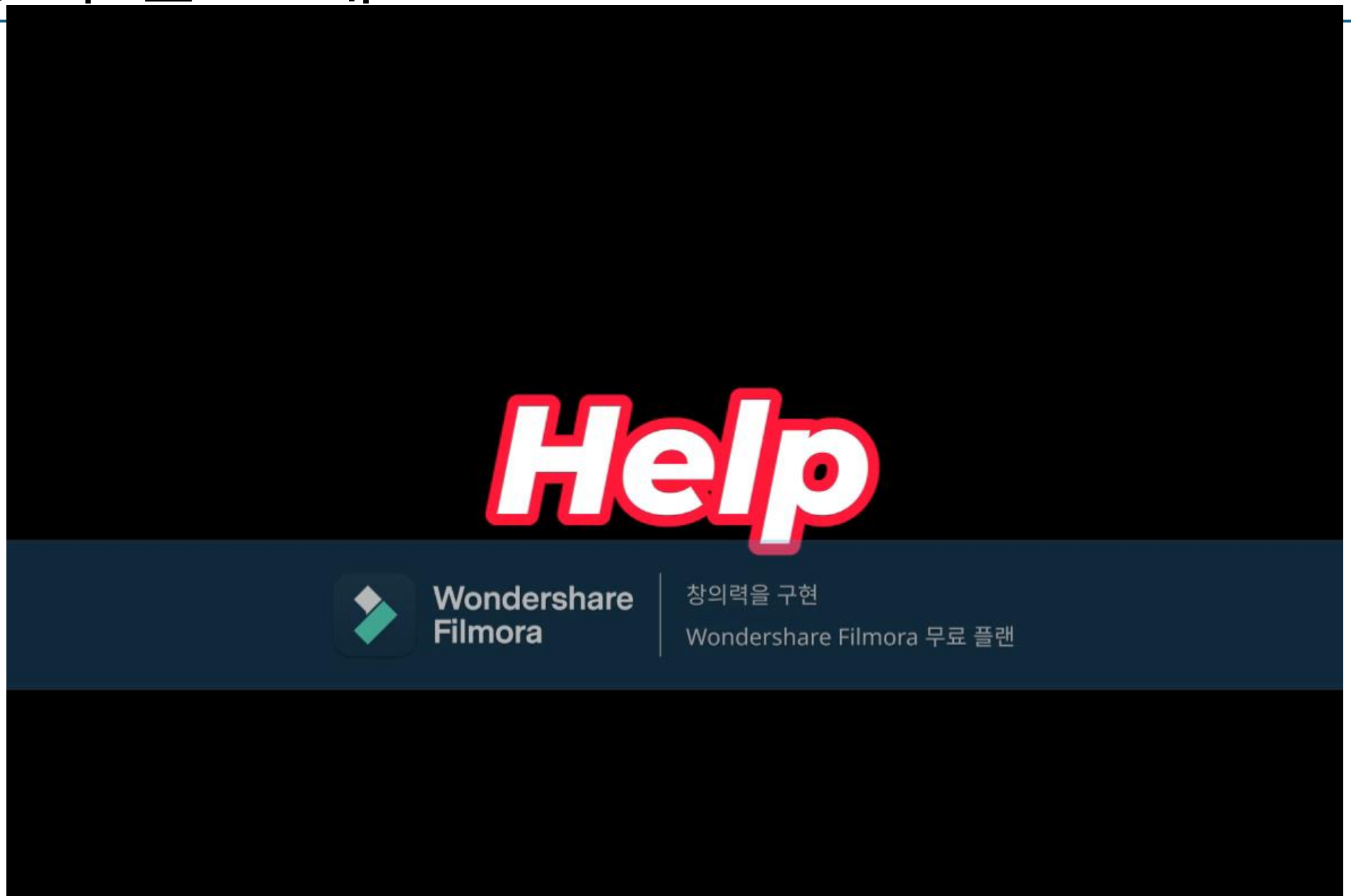
기능 구현 소개 – SSD 기본



기능 구현 소개 - Shell



기능 구현 소개 - Shell



TDD 활용 예시 - SSD Write

RED

```python
def test_ssd_write_pass():  saerom
    input_address = '00'
    input_value = '00000001'
    expected_value = ''
    ssd = SSD()
    ssd.write(input_address, input_value)
                매번 arrange 를 해주는 중
    with open('ssd_output.txt') as f:
        actual_value = f.read()
    assert actual_value == expected_value

    input_address = '02'
    ssd.write(input_address, input_value)
    with open('ssd_output.txt') as f:
        actual_value = f.read()
    assert actual_value == expected_value

    input_address = '99'
    ssd.write(input_address, input_value)
    with open('ssd_output.txt') as f:
        actual_value = f.read()
    assert actual_value == expected_value
```

GREEN

```python
def write(self, line_number, new_content):  18 usages  SojeongPark
    print('START WRITE')
    if not self.validate_address(line_number):
        self.report_error()
        return
    if not self.validate_value(new_content):
        self.report_error()
        return
    line_number = int(line_number)
    if new_content is None:                    복잡하게 모여있는 예외 처리 등
        self.report_error()
        return
    with open(SSD_NAND_FILE_PATH, encoding='utf-8') as f:
        lines = f.readlines()

    lines[line_number] = f'{line_number:02d} {new_content}\n'
    print(lines)
    with open(SSD_NAND_FILE_PATH, 'w', encoding='utf-8') as f:
        f.writelines(lines)

    print(f'Write Success! {line_number:02d}:{new_content}')
    with open(SSD_OUTPUT_FILE_PATH, 'w', encoding='utf-8') as f:
        f.write('')
```

TDD 활용 예시 - SSD Write

REFACTOR

```
def run_direct(ssd_instance):  16 usages  WB
    def runner(*args):  WB
        ssd_instance.execute_test(args)
        ssd_instance.execute_test('F')

    return runner


def run_cli(ssd_instance):  16 usages  WB
    def runner(*args):  WB
        cli_args = [str(arg) for arg in args]
        full_command = f'{COMMAND_PREFIX} {cli_args[0]} {" ".join(cli_args[1:])}'
        subprocess.run(full_command, shell=True, check=True)
        subprocess.run( args: f'{COMMAND_PREFIX} F', shell=True, check=True)

    return runner
```

Ssd 직접 호출과, cli 환경을 통한 실행 환경 준비

Fixture 를 활용하여 arrange 번거로움 해소

```
@pytest.fixture(params=['0', '50', '99'])
def valid_address(request):
    return request.param
```

```
@pytest.mark.parametrize('runner_factory', [run_direct, run_cli])  WB
def test_ssd_write_pass(ssd, runner_factory, valid_address):
    runner = runner_factory(ssd)
    runner('W', valid_address, VALID_VALUE)

    actual_value = read_file_with_lines(SSD_OUTPUT_FILE_PATH)
    assert not actual_value

    actual_value = read_file_with_lines(SSD_NAND_FILE_PATH)
    assert actual_value[int(valid_address)] == f'{int(valid_address):02d} {VALID_VALUE}'
```

TDD 활용 예시 - SSD Write

R E F A C T O R

```python
class CommandFactory:  7 usages  WB +1
    MODES = {
        'R': {'command': ReadCommand, 'args_count': 1},
        'W': {'command': WriteCommand, 'args_count': 2},
        'E': {'command': EraseCommand, 'args_count': 2},
        'F': {'command': FlushCommand, 'args_count': 0},
    }

    @staticmethod  2 usages  WB +1
    def create_command(args):
        if not args:
            raise InvalidInputError('No arguments provided')

        mode = args[0].upper()
        ssd = SSD()

        if mode not in CommandFactory.MODES:
            raise InvalidInputError(
                f'Invalid mode: {mode}. Supported modes are {", ".join(CommandFactory.MODES.keys())}.'
            )

        if CommandFactory.MODES[mode]['args_count'] != len(args) - 1:
            raise InvalidInputError(
                f'{mode} only accepts {CommandFactory.MODES[mode]["args_count"]} arguments'
            )

        command = CommandFactory.MODES[mode]['command']
        return command(ssd, *args[1:])
```

객체 생성을 관리 하기 위해
Simple Factory 패턴으로
Command 를 생성하고자 함

생성 시,
잘못된 명령어 예외 처리

TDD 활용 예시 - SSD Write

REFACTOR

```python
class WriteCommand(Command):  SojeongPark
    def __init__(self, ssd, address, value):  SojeongPark
        self.ssd = ssd
        self.address = address
        self.value = value

    @file_lock_decorator(LOCK_FILE_PATH)  SojeongPark
    def execute(self):
        if not self.ssd.validate_address(self.address) or not self.ssd.validate_value(
            self.value
        ):
            raise InvalidInputError('Address validation failed')
        self.ssd.buf_write(int(self.address), self.value)
```

```python
def _write(self, address, new_content):  1 usage  SojeongPark +1
    with open(SSD_NAND_FILE_PATH, 'r+', encoding='utf-8') as f:
        lines = f.readlines()
        lines[address] = f'{address:02d} {new_content}\n'
        f.seek(0)
        f.writelines(lines)
        f.truncate()

    self.initialize_ssd_output()
    self.logger.info(f'Write complete: {address:02d}: {new_content}')
```

Command Pattern 적용으로
확장성에 용이하게 수정하였고,

해당 command 에서
주소, 값 validity 검증

Ssd 내부 write 간략화.
온전히 해당 기능만을 수행

TDD 활용 예시 - SSD Read

RED

```python
@pytest.mark.parametrize('runner_factory', [run_direct, run_cli])
def test_ssd_read_initial_value_check(ssd, runner_factory, valid_address):
    runner = runner_factory(ssd)
    runner('R', valid_address)

    actual_value = read_file_with_lines(SSD_OUTPUT_FILE_PATH)
    assert actual_value == [INITIAL_VALUE]


@pytest.mark.parametrize('runner_factory', [run_direct, run_cli])
def test_ssd_read_fail_wrong_address(ssd, runner_factory, invalid_address):
    runner = runner_factory(ssd)
    runner('R', invalid_address)

    actual_value = read_file_with_lines(SSD_OUTPUT_FILE_PATH)
    assert actual_value == [ERROR]


@pytest.mark.parametrize('runner_factory', [run_direct, run_cli])
def test_ssd_read_written_value_pass(ssd, runner_factory, valid_address):
    runner = runner_factory(ssd)

    runner('W', valid_address, VALID_VALUE)
    actual_value = read_file_with_lines(SSD_OUTPUT_FILE_PATH)
    assert not actual_value

    runner('R', valid_address)
    actual_value = read_file_with_lines(SSD_OUTPUT_FILE_PATH)
    assert actual_value == [VALID_VALUE]
```

GREEN

```python
def read(self, input_address):
    ret_value = ""
    with open('./ssd_nand.txt', 'r') as f:
        for line in f:
            data = line.strip().split(' ')
            ind = int(data[0])
            value = data[1]

            if int(input_address) == ind:
                ret_value = value

    with open('./ssd_output.txt', 'w') as f:
        f.write(f'{ret_value}')
```

TDD 활용 예시 - SSD Read

R
E
F
A
C
T
O
R

```
def read(self, address):  astralmin +2 *
    with open(SSD_NAND_FILE_PATH) as f:
            lines = f.readlines()
            ret_value = lines[int(address)].strip().split(' ')[1]
    with open(SSD_OUTPUT_FILE_PATH, 'w') as f:
        f.write(f'{ret_value}')
    self.logger.info(f'Read complete: {address:02d}: {ret_value}')
```

- 하드코딩 된 문자열을 상수로 추출 후 적용
- 불필요한 if문 제거
- 1회성 변수 inline 시켜 loc 감소

TDD 활용 예시 - SSD 전반적 개발 과정

Test fail with skeleton code

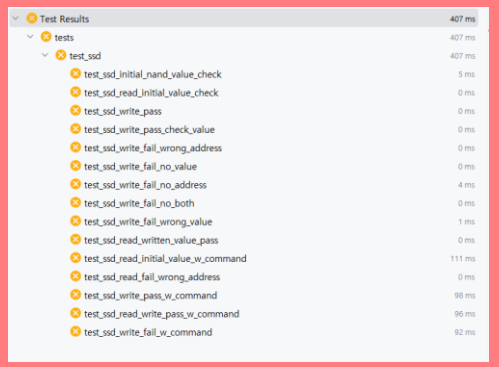
```python
class SSD:  12 usages  SojeongPark *
    def __init__(self):  SojeongPark *
        pass

    def initialize_ssd_nand(self) -> None:  SojeongPark *
        pass

    def initialize_ssd_output(self) -> None:  SojeongPark *
        pass

    def read(self):  SojeongPark
        pass

    def write(self, line_number, new_content):  SojeongPark *
        pass
```



Test pass after Implementation

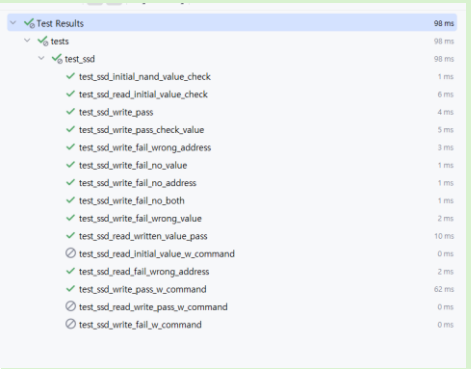
```python
def read(self):  SojeongPark
    pass

def write(self, line_number, new_content):  SojeongPark
    if not self.validate_address(line_number):
        self.report_error()
        return
    if not self.validate_value(new_content):
        self.report_error()
        return
    line_number = int(line_number)

    with open(SSD_NAND_FILE_PATH, encoding='utf-8') as f:
        lines = f.readlines()

    lines[line_number] = f'{line_number:02d} {new_content}\n'
    with open(SSD_NAND_FILE_PATH, 'w', encoding='utf-8') as f:
        f.writelines(lines)

    with open(SSD_OUTPUT_FILE_PATH, 'w', encoding='utf-8') as f:
        f.write('')
```



Refactoring

```
* 503c868 - refactor: Use Exception error for Error case (2 days ago) <SojeongPark>
* 4bb3ce8 - refactor: Use Absolute path (2 days ago) <SojeongPark>
* cdff305 - fix: Report error when input is invalid (3 days ago) <SojeongPark>
* dcd0b71 - Fix: Fix for command input (3 days ago) <SojeongPark>
* 72b9b23 - refactor: Remove dulplicated code (3 days ago) <SojeongPark>
* cc4f791 - test: Use fmt skip for useless class creator (3 days ago) <SojeongPark>
* 473d8d9 - Feat: SSD- write use 0x format (3 days ago) <SojeongPark>
* add8ebf - Feat: Implement of class SSD (3 days ago) <SojeongPark>
* c35824d - Feat: SSD-Write (3 days ago) <SojeongPark>
```

TDD 활용 예시 - Shell full_write

R
E
D

```python
def test_shell_fullwrite(capsys: pytest.CaptureFixture, mocker: MockerFixture):
    ssd = mocker.Mock()
    mock_process = mocker.Mock()
    mock_process.returncode = 0

    mock_run = mocker.patch('subprocess.run', return_value=mock_process)
    mocker.patch('builtins.input', side_effect=['fullwrite 0xFFFFFFFF', 'exit'])

    shell = Shell(ssd)
    shell.run()

    captured = capsys.readouterr()
    output = captured.out

    assert '[Full Write] Done' in output
    mock_run.assert_called_with(
        [
            'python',
            'C:\\Users\\User\\PycharmProjects\\pythonProject31\\ssd.py',
            'W',
            '99',
            '0xFFFFFFFF',
        ],
        check=True,
    )
```

Ssd 직접 주입

긴 test 입력 TC 내부 나열 작성

```python
def test_shell_fullwrite_invalid_input(
    capsys: pytest.CaptureFixture, mocker: MockerFixture
):
    ssd = mocker.Mock()

    mocker.patch('builtins.input', side_effect=['fullwrite 0 0', 'exit'])

    shell = Shell(ssd)
    shell.run()

    captured = capsys.readouterr()
    output = captured.out

    assert '[Read]  ERROR' in output
```

직접 문자열로 TC 작성

TDD 활용 예시 - Shell full_write

G
R
E
E
N

```
class FullWriteCommand(Command):
    def __init__(self):
        self._logger = Logger(Pre.READ)
        self._lba = None

    def parse(self, args: list[str]) -> list[str]:
        if len(args) != 1:
            raise ValueError(Msg.WRITE_HELP)
        data = args[0]
        if not self._check_data(data):
            raise ValueError('Data must be a hex string like 0x0129ABCF')
        return ['W', self._lba, data]

    def execute(self, args, ssd=None) -> bool:
        try:
            for index in LBA_RANGE:
                self._lba = f'{index:02d}'
                ssd_args = self.parse(args)
                return_code = subprocess.run(RUN_SSD + ssd_args, check=True)
                if return_code.returncode != 0:
                    self._logger.error(ShellMsg.ERROR)
                with open(SSD_OUTPUT_FILE) as f:
                    result = self.parse_result(f.read().strip())
                self._logger.info(result)
        except ValueError:
            self._logger.error(ShellMsg.ERROR)
        return True
```

기능 추가를 위한 최소한의 상속 구조만 따르며 기능 작성

상위 기능과 무관하게 단순 overwrite하여 기능 작성

중복 코드, 역할 고려하지 않고
기능 구현만을 위해 직접 subprocess 호출

TDD 활용 예시 - Shell full_write

REFACTOR

Full_write

```python
from commands.script import ScriptCommand
from shell_constants import LBA_RANGE
from shell_constants import ShellMsg as Msg
from shell_constants import ShellPrefix as Pre
from shell_logger import Logger


class FullWriteCommand(ScriptCommand):
    def __init__(self, logger: Logger, prefix=Pre.FULLWRITE):
        super().__init__(logger, prefix)
        self._lba = None

    def execute(self, args=None) -> bool:
        try:
            for index in LBA_RANGE:
                self.write(index, args[0])
                self._logger.print_and_log(self._prefix, Msg.DONE)
        except ValueError:
            self._logger.print_and_log(self._prefix, Msg.ERROR)
        return True
```

공통 기능 캡슐화하여 상위로 추출, 상속

필요 추가 기능(logger)은 외부 주입

동작 method 추출 하여
역할 및 수준에 따라 분리

```python
@pytest.fixture
def case_list():
    test_cases = []
    for index in LBA_RANGE:
        cmd_args = RUN_SSD + [
            'W',
            str(index),
            TEST_DATA,
        ]
        test_cases.append(call(cmd_args, check=True))

    return test_cases


def test_shell_fullwrite(
    capsys: pytest.CaptureFixture, mocker: MockerFixture, case_list
):
    mock_process = mocker.Mock()
    mock_process.returncode = 0

    mock_run = mocker.patch('subprocess.run', return_value=mock_process)
    mocker.patch(
        'builtins.input', side_effect=[f'{Cmd.FULLWRITE} {TEST_DATA}', Cmd.EXIT]
    )

    shell = Shell()
    shell.run()

    captured = capsys.readouterr()
    output = captured.out

    assert Pre.FULLWRITE + ' ' + Msg.DONE in output
    for case in case_list:
        assert case in mock_run.call_args_list
```

Fixture 사용하여 case 활용

패키지 내 상수/변수
통일하여 선언 및 정리

TC와 test case 분리 외부화

Mocking 활용 예시

SSD flush 테스트
- SSD flush 가 호출되었는지 아닌지 테스트 하는 과정에서,
- 버퍼에 5개가 찼을 때,
- Flush 함수가 호출이 되었는지 검증하는 부분에서 mock 활용

```
@pytest.mark.parametrize('runner_factory', [run_direct_wo_flush])
def test_ssd_auto_flush_operation_with_direct(ssd, runner_factory):
    runner = runner_factory(ssd)
    with patch('ssd.SSD.flush') as mock_flush:
        for i in range(5):
            runner('W', str(i), VALID_VALUE)

        mock_flush.assert_not_called()
        runner('R', '0')
        mock_flush.assert_called_once()

        mock_flush.reset_mock()
        actual_value = read_buffer()
        assert 'empty' not in actual_value
```

Mocking 활용 예시 - shell(SUT) Erase Test

```python
def test_erase_call_args_unaligned_list_valid_order(  junt-lee
    capsys: pytest.CaptureFixture, mocker: MockerFixture, mock_run
):
    mocker.patch( target: 'builtins.input', side_effect=[f'{Cmd.ERASE} 0 98 ', Cmd.EXIT])
    test_args_list = [('E', str(i), '10') for i in range(0, 90, 10)] + [
        ('E', '90', '8')
    ]
    split_case = case_list(test_args_list)

    shell = Shell()
    shell.run()

    mock_run.assert_has_calls(split_case, any_order=False)
```

- 테스트 설명

Shell이 size를 10단위로 쪼개서 SSD를 알맞은 Param으로 호출하는지 검증.

```
Shell input: "erase 0 98"
-> SSD Call: "E 0 10" "E 10 10" ... "E 90 8"
```

```python
@pytest.fixture  junt-lee
def mock_run(mocker):
    mock_process = mocker.Mock()
    mock_process.returncode = 0

    # subprocess.run을 포함한 patch 설정
    mock_run = mocker.patch('subprocess.run', return_value=mock_process)
    mocker.patch('builtins.open', mocker.mock_open(read_data=''))

    return mock_run
```

* Mock 활용

Shell이 의존하는 SSD객체를 Mock으로 설정
SSD를 실행하는 subprocess 함수를 patch 설정.
Return code, file open의 retur값을 stubbing.
Shell에 원하는 input을 넣어주기 위해
Builtins.input을 patch, side_effect 활용.

Subprocess의 호출 param순서를 검증하기 위해
Assert_has_calls 활용

리팩토링 전후 비교 – 2_PartialLBAWrite

리팩토링 이전

```
class PartialLBAWriteCommand(Command):
    def __init__(self):
        self._logger = Logger(Pre.READ)
        self._lba = None
        self._read = ReadCommand()
        self._write = WriteCommand()
        self._random_value = random.randint(0x00000000, 0xFFFFFFFF)

    def parse(self, args: list[str]) -> list[str]: ...

    def execute(self, args, ssd=None) -> bool:
        try:
            sample_index = ['04', '00', '03', '01', '02']
```

나열된 동작 코드

```
            for _ in range(30):
                hex_string = f'0x{self._random_value:08X}'
                for index in sample_index:
                    self._lba = index
                    ssd_args = ['W', self._lba, hex_string]
                    return_code = subprocess.run(RUN_SSD + ssd_args, check=True)
                    if return_code.returncode != 0:
                        self._logger.error(ShellMsg.ERROR)

                for index in sample_index:
                    self._lba = index
                    ssd_args = ['R', self._lba]
                    return_code = subprocess.run(RUN_SSD + ssd_args, check=True)
                    if return_code.returncode != 0:
                        self._logger.error(ShellMsg.ERROR)
                    with open(SSD_OUTPUT_FILE) as f:
                        read_value = f.read().strip()
                        if read_value != hex_string:
                            print('[2_PartialLBAWrite] Fail')
                            return True
                print('[2_PartialLBAWrite] Pass')
```

하드코딩된 상수/문자

리팩토링 전후 비교 - 2_PartialLBAWrite

리팩토링 과정

```
self._lba = index
ssd_args = ['R', self._lba]
return_code = subprocess.run(RUN_SSD + ssd_args, check=True)
if return_code.returncode != 0:
    self._logger.error(ShellMsg.ERROR)
with open(SSD_OUTPUT_FILE) as f:
    read_value = f.read().strip()
```

```
ssd_args = [index]
read_value = self._read.execute(ssd_args)
if read_value != hex_string:
```

중복된 기능 하위 객체 생성하여 메소드 활용

```
self._lba = index
ssd_args = ['W', self._lba, hex_string]
return_code = subprocess.run(RUN_SSD + ssd_args, check=True)
if return_code.returncode != 0:
    self._logger.error(ShellMsg.ERROR)
self._execute_write(index, hex_string)
```

```
def _execute_write(self, lba, current_value):
    cmd = f'{lba} {current_value}'
    self._write.execute(cmd.split())
```

나열된 동작 메소드 추출

```
            print('[2_PartialLBAWrite] Fail')
            self._logger.info(Pre.SCRIPT2 + ShellMsg.FAIL)
            return True
print('[2_PartialLBAWrite] Pass')
self._logger.info(Pre.SCRIPT2 + ShellMsg.PASS)
```

상수/문자열 추출 및 의미 부여

리팩토링 전후 비교 - 2_PartialLBAWrite

리팩토링 이후
- 공통 기능 가지는 Class 생성

```
class ScriptCommand(Command):
    def __init__(self, logger: Logger, prefix=None):
        super().__init__(logger, prefix)
        self._read_cmd: ReadCommand | None = None
        self._write_cmd: WriteCommand | None = None

    @property
    def read_cmd(self):
        if self._read_cmd is None:
            self._read_cmd = ReadCommand(self._logger, prefix=None)
        return self._read_cmd

    @property
    def write_cmd(self):
        if self._write_cmd is None:
            self._write_cmd = WriteCommand(self._logger, prefix=None)
        return self._write_cmd
```

자주 사용되는 공통 기능 생성자
Property로 대체

```
def read(self, lba) -> str:
    if not isinstance(lba, str):
        lba = str(lba)
    self.read_cmd.execute([lba])
    return self.read_cmd.result

def write(self, lba, value):
    if not isinstance(lba, str):
        lba = str(lba)
    if not isinstance(value, str):
        value = str(value)
    self.write_cmd.execute([lba, value])

def read_with_verify(self, lba, expected) -> bool:
    read_value = self.read(lba)
    return read_value == expected

def parse(self, args: list[str]) -> list[str]:
    return []

def parse_result(self, result) -> str:
    return ''
```

공통 기능 상위화

리팩토링 전후 비교 - 2_PartialLBAWrite

리팩토링 이후
- 정의한 ScriptCommand 를 상속 받아 사용

출력 메시지,
파라미터 정의 분리

반복/공통 기능 추출
및 상위화

```
class PartialLBAWriteCommand(ScriptCommand):
    def __init__(self, logger: Logger, prefix=Pre.SCRIPT_2):
        super().__init__(logger, prefix)
        self._lba = None
        self._random_value = random.randint(0x00000000, 0xFFFFFFFF)

    def execute(self, args=None) -> bool:
        try:
            sample_index = ['4', '0', '3', '1', '2']
            self._logger.print_blank_line()
            self._logger.print_and_log(self._prefix)
            for _ in range(30):
                hex_string = f'0x{self._random_value:08X}'
                for index in sample_index:
                    self.write(index, hex_string)

                for index in sample_index:
                    success = self.read_with_verify(index, hex_string)
                    if not success:
                        self._logger.print_and_log(self._prefix, ShellMsg.FAIL)
                        return True
                self._logger.print_and_log(self._prefix, ShellMsg.PASS)
        except ValueError:
            self._logger.print_and_log(self._prefix, ShellMsg.ERROR)
        return True
```

리팩토링 전후 비교 - SSD Buffer

- Refactoring으로 동일한 역할을 하는 메소드를 함수로 추상화
- 하드코딩 된 문자열을 상수로 변경

```
if mode == CMD_WRITE:
    for i in range(seq, -1, -1):
        buf = self._buffer_list[i]
        if buf[0] == CMD_WRITE and buf[1] == param1:
            self._remove_buffer(i)
            self._sort_buffer()
            seq -= 1
            break
        if buf[0] == CMD_ERASE:
            erased = self._reduce_erase_buffer(i, param1)
            if erased:
                self._sort_buffer()
                seq -= 1
                break
elif mode == CMD_ERASE:
    for i in range(seq, -1, -1):
        buf = self._buffer_list[i]
        if buf[0] == CMD_WRITE and param1 <= buf[1] < param1 + param2:
            self.logger.info(f'Write is useless now. Remove buffer {i}')
            self._remove_buffer(i)
            self._sort_buffer()
            seq -= 1
```

```
def _remove_and_sort_buffer(self, i, seq):
    self._remove_buffer(i)
    self._sort_buffer()
    seq -= 1
    return seq
```

```
for i in range(seq, -1, -1):
    buf = self._buffer_list[i]
    if mode == CMD_WRITE:
        if buf[0] == CMD_WRITE and buf[1] == param1:
            seq = self._remove_and_sort_buffer(i, seq)
            break
        if buf[0] == CMD_ERASE and self._reduce_erase_buffer(i, param1):
            seq = self._remove_and_sort_buffer(i, seq)
            break
    elif mode == CMD_ERASE:
        if buf[0] == CMD_WRITE and param1 <= buf[1] < param1 + param2:
            self.logger.info(f'Write is useless now. Remove buffer {i}')
            seq = self._remove_and_sort_buffer(i, seq)
```

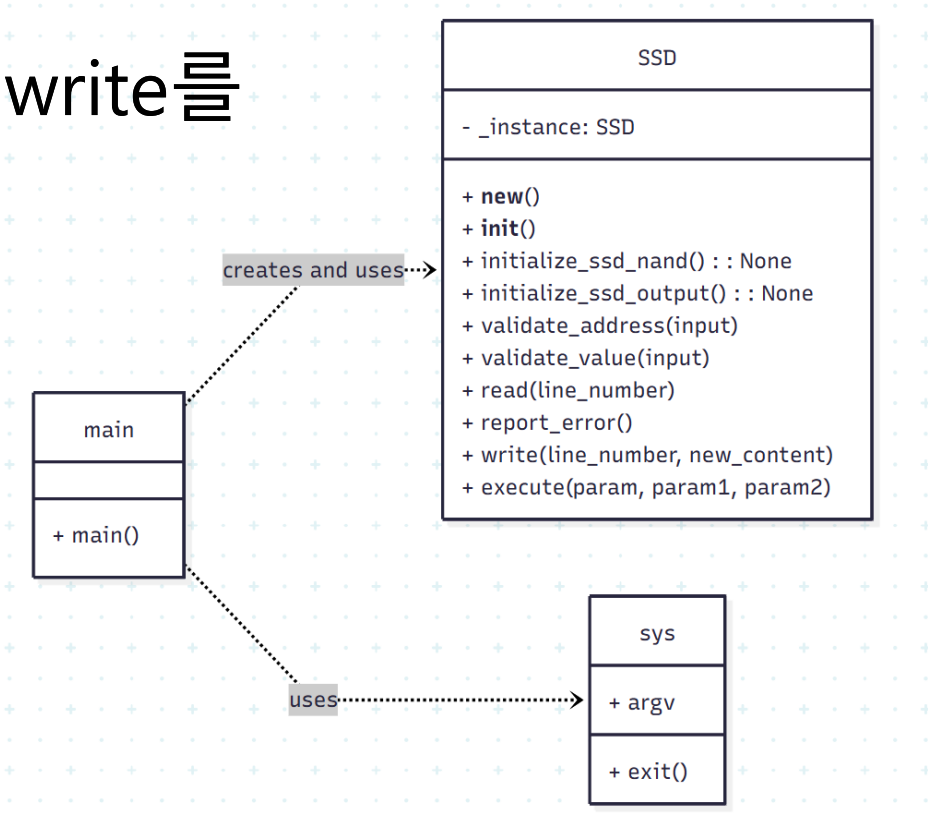
공통 동작에 대해
메서드 추출

```
6     MAX_BUFFER_SIZE = 5
7     ERASE_MAX_RANGE = 10
8     CMD_WRITE = 'W'
9     CMD_ERASE = 'E'
10    EMPTY = 'empty'
11    BUFFER_DIR_NAME = 'buffer'
12    BUFFER_DIR = (
13        f'{os.path.dirname(os.path.dirname(os.path.abspath(__file__)))}/{BUFFER_DIR_NAME}'
14    )
```

하드코딩 된 문자열 상수화

리팩토링 전후 비교 - SSD 디자인 패턴 적용 1

디자인 패턴 적용 이전
- Main 에서 바로 SSD의 read, write를 호출하는 상황

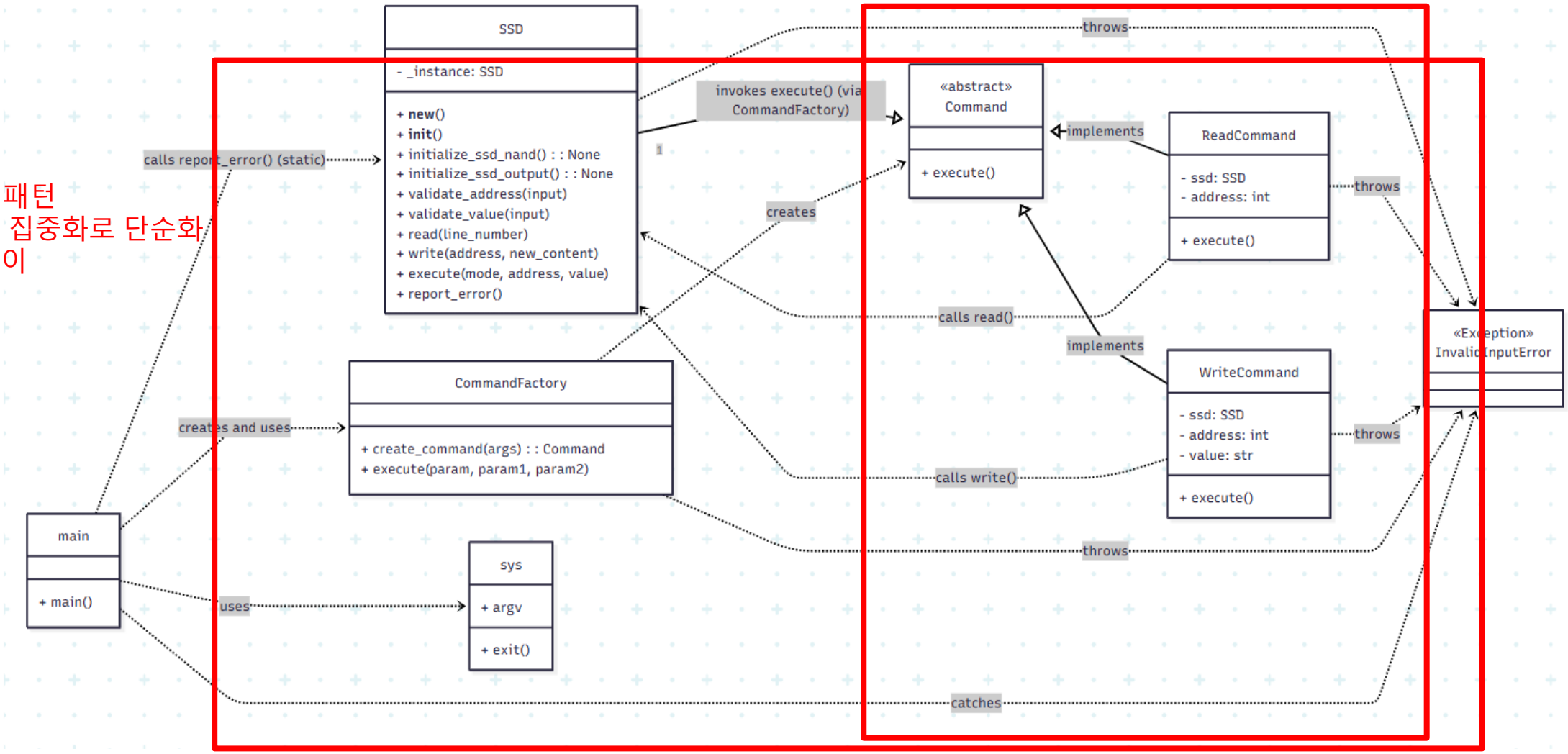


리팩토링 전후 비교 - SSD 디자인 패턴 적용 1

디자인 패턴 적용 이후

Command 패턴
- 커맨드를 캡슐화하여 요청 발행 객체와 요청 수행 객체 분리
확장성 추가

Simple Factory 패턴
- 생성 로직 집중화로 단순화
- 변경에 용이



리팩토링 전후 비교 - command 패턴

```python
def main():  1 usage  SojeongPark *
    args = sys.argv[1:]
    try:
        if len(args) < 2:
            raise InvalidInputError()

        mode = args[0].upper()

        if mode == 'W':
            if len(args) != 3:
                raise InvalidInputError()
            address = args[1]
            value = args[2]

        elif mode == 'R':
            if len(args) != 2:
                raise InvalidInputError()
            address = args[1]
        else:
            raise InvalidInputError()

        ssd = SSD()
        if mode == 'W':
            ssd.write(address, value)
        elif mode == 'R':
            ssd.read(address)
    except InvalidInputError:
        SSD.report_error()
        sys.exit(1)
```

ssd.py 실행 시, main에서
input으로 실행 method 판단

```python
class CommandFactory:  7 usages  WB +1
    MODES = {
        'R': {'command': ReadCommand, 'args_count': 1},
        'W': {'command': WriteCommand, 'args_count': 2},
        'E': {'command': EraseCommand, 'args_count': 2},
        'F': {'command': FlushCommand, 'args_count': 0},
    }

    @staticmethod  2 usages  WB +1
    def create_command(args):...
```

ssd.py 실행 시, main에서 Command Factory를
호출하여 알맞은 Command instance 생성

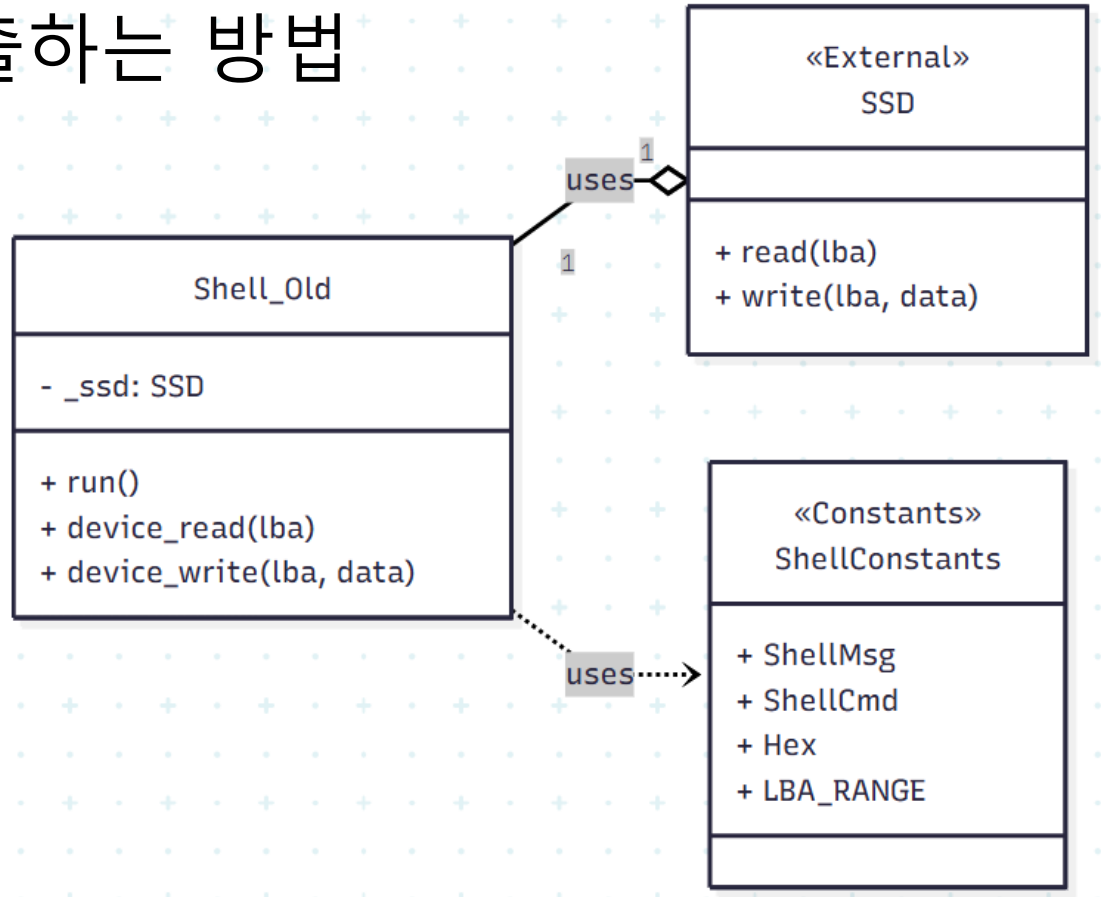
```python
class EraseCommand(Command):  1 usage  WB +1
    def __init__(self, ssd, address, size):  WB
        self.ssd = ssd
        self.address = address
        self.size = size

    @file_lock_decorator(LOCK_FILE_PATH)  WB +1
    def execute(self):
        if not self.ssd.validate_address(self.address) or not self.ssd.validate_size(
            self.address, self.size
        ):
            raise InvalidInputError('Address validation failed')
        self.ssd.buf_erase(int(self.address), int(self.size))
```

각 Command instance는 execute() method를 사용하여
ssd에서 실제로 수행해야 하는 method를 호출

리팩토링 전후 비교 - Shell 디자인 패턴 적용 1

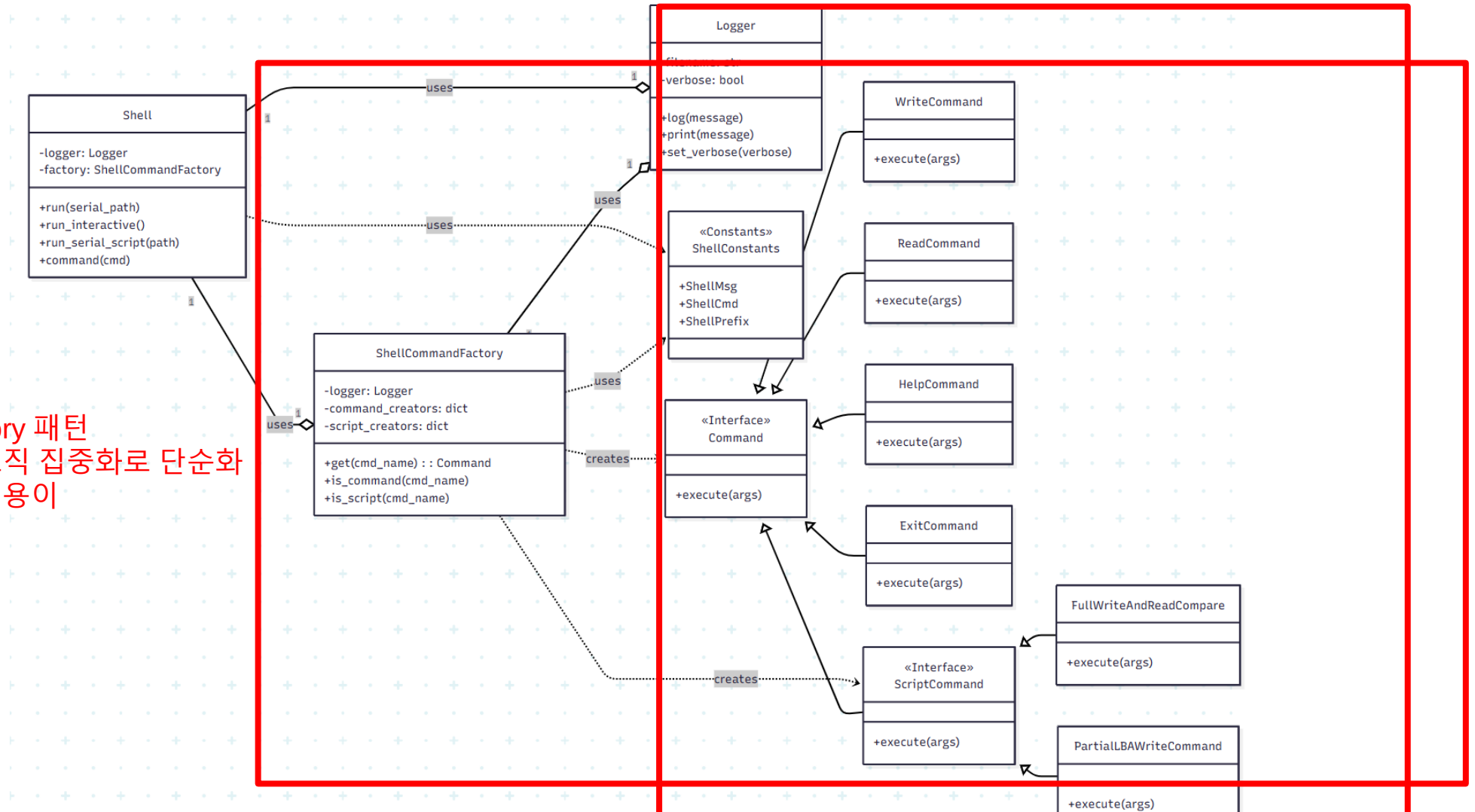
디자인 패턴 적용 이전
- SSD와 마찬가지로 직접 모두 호출하는 방법



리팩토링 전후 비교 - Shell 디자인 패턴 적용 1

디자인 패턴 적용 이후

Command 패턴
- 커맨드를 캡슐화하여 요청 발행 객체와 요청 수행 객체 분리
확장성 추가

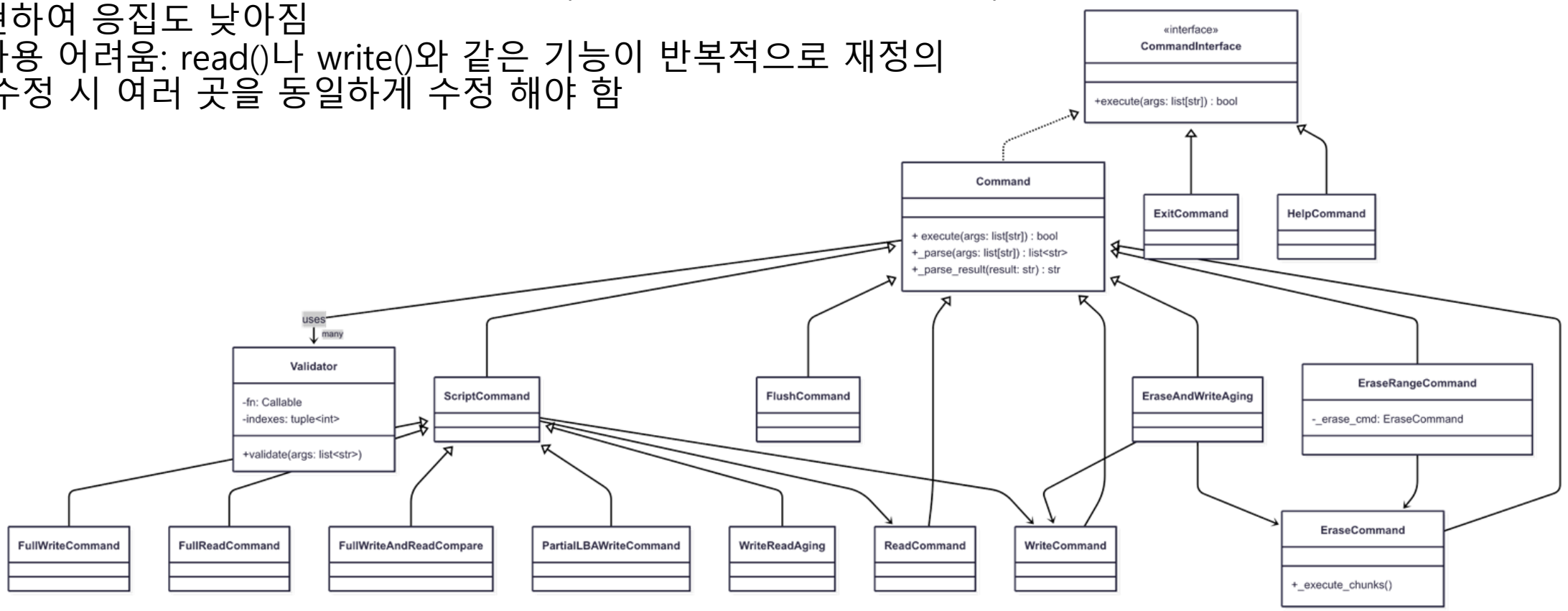


Simple Factory 패턴
- 생성 로직 집중화로 단순화
- 변경에 용이

리팩토링 전후 비교 - Shell 디자인 패턴 적용 2

디자인 패턴 적용 이전

- 중복 발생: 모든 커맨드가 필요 시마다 필요한 Command 인스턴스를 직접 만들고 실행 흐름도 직접 관리
- 관심사의 침해: 원래 책임과 무관한 로직(예: 읽기 동작, 에러 처리 등)를 각 Command에서 구현하여 응집도 낮아짐
- 재사용 어려움: read()나 write()와 같은 기능이 반복적으로 재정의
  → 수정 시 여러 곳을 동일하게 수정 해야 함

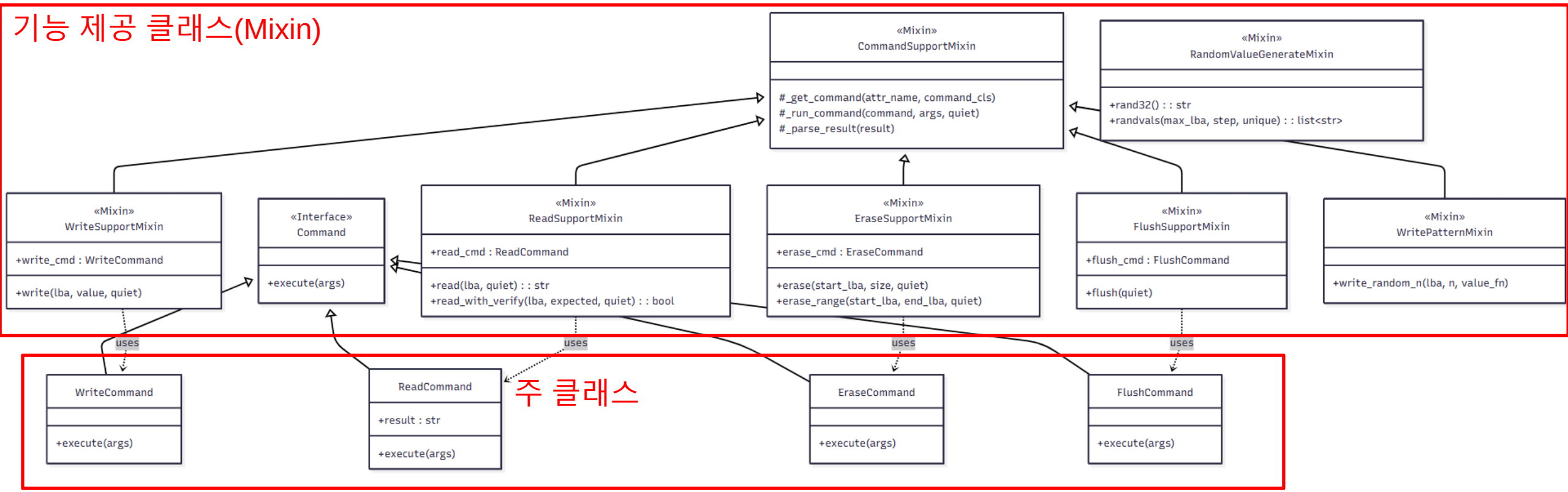


리팩토링 전후 비교 - Shell 디자인 패턴 적용 2

Command에 Mixin 패턴(SRP + 다중 상속) 적용

단일 책임 원칙(SRP)을 따라 **하나의 기능 단위**로 나뉘어 구현하여 **필요한 클래스에서만 선택적으로 상속**받아 사용

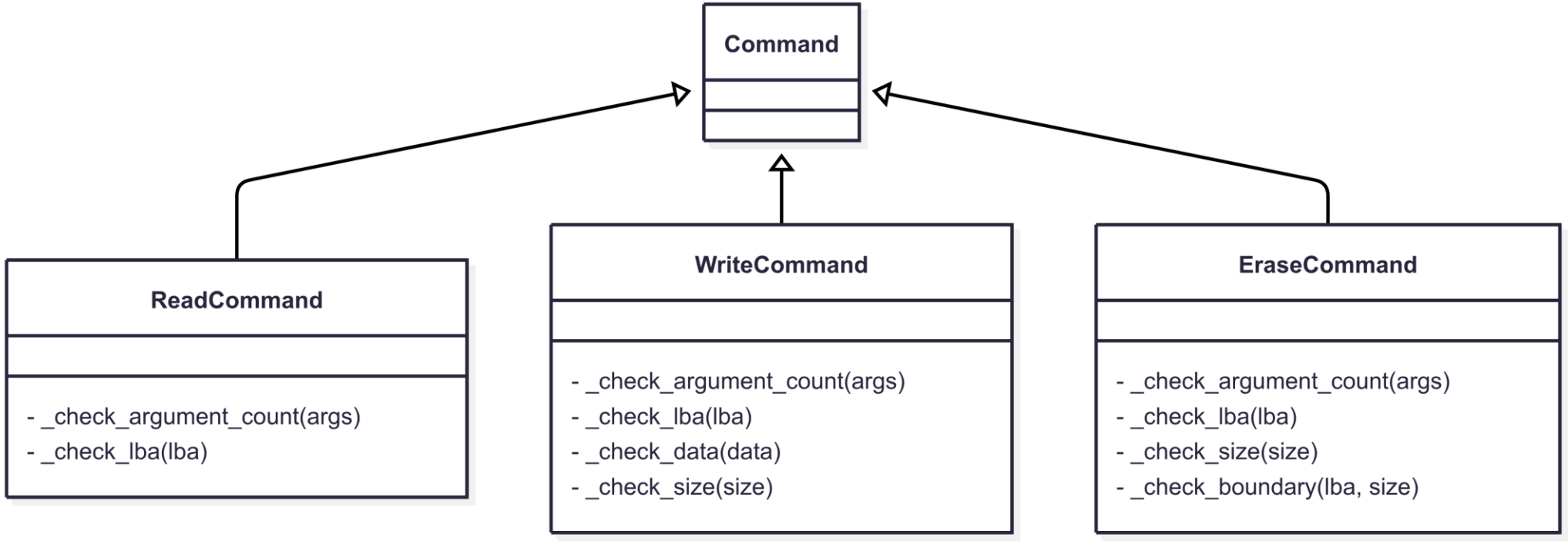
- 중복 코드 제거 및 코드 재사용성 증가
- 작은 기능 단위로 테스트 가능하며 전체 기능에 영향을 주지 않고 기능 단위로 변경/교체 가능
- 기능 수평 확장에 용이



리팩토링 전후 비교 - Shell 디자인 패턴 적용 3

디자인 패턴 적용 이전

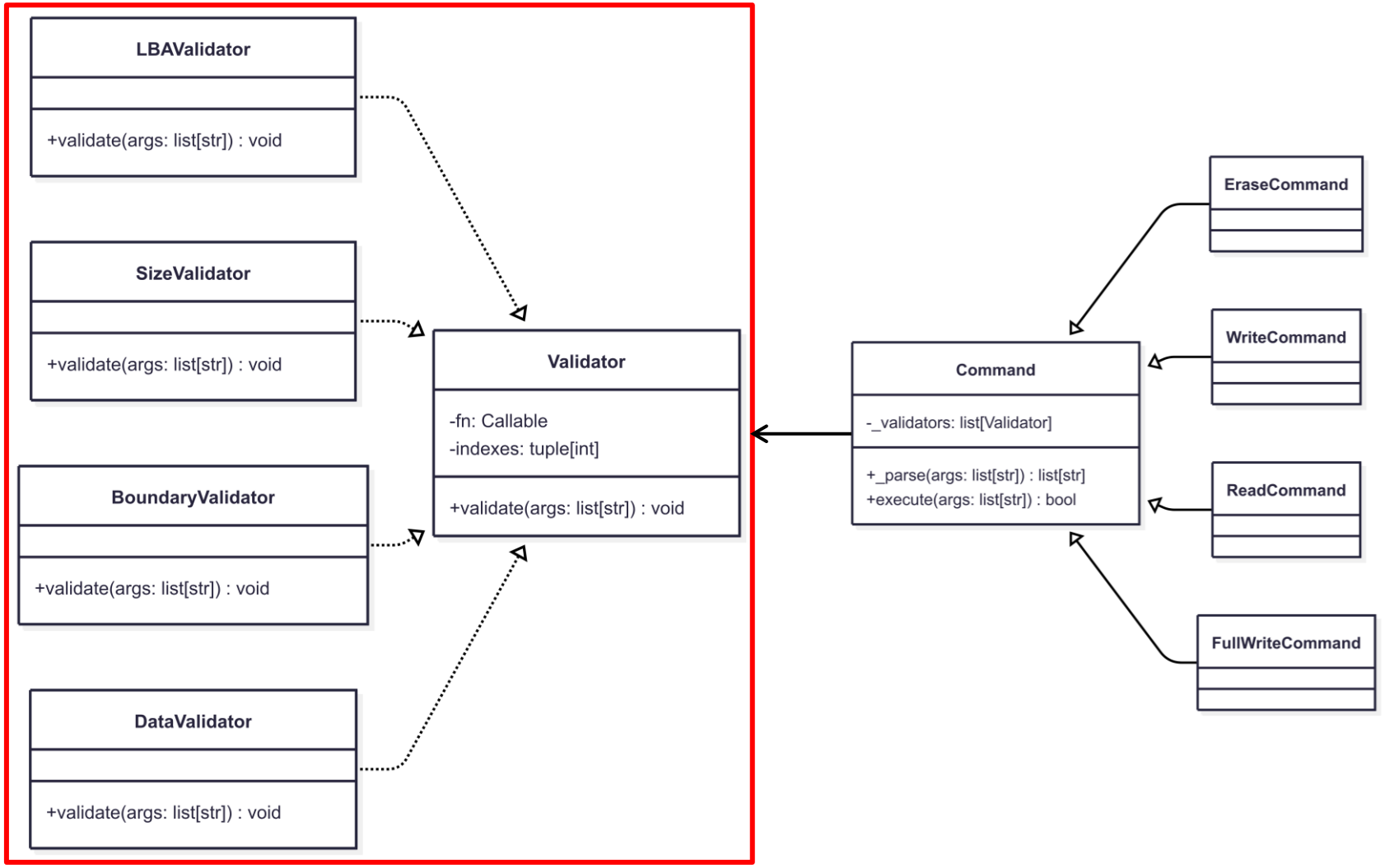
- 동일한 유효성 검사 로직이 여러 클래스에 걸쳐 중복 구현
- 검증 순서 및 처리 방식의 일관성 부족: Command마다 검증 구현 방식이 달라서, 검사 순서, 예외 메시지, 로그 출력 방식이 제각각
- 재사용성 낮음: 다른 클래스에서 재사용하거나 결합 어려움



리팩토링 전후 비교 - Shell 디자인 패턴 적용 3

디자인 패턴 적용 이후

전략 패턴 적용
-Validator를 통해 공통 검증 전략 재사용
-유연한 검증 로직 교체/확장 가능
-통합된 로직으로 사용자 메시지 일관화
-Command 간 설계 통일, 유지보수 용이
-Validator 단위 테스트 가능



소감

황웅범님	A 특공대 팀을 만나 너무 좋았습니다. 다들 너무 멋지시고 열정적으로 임하시는 모습이 존경스러웠습니다. 모르는 것들도 많이 배울 수 있었던 경험이었고, 이런 프로젝트를 빠른 시간내에 유기적으로 핑퐁하며 진행했던 경험이 별로 없었는데 값진 경험이었습니다.
이민호님	TDD를 처음 접했을 때는 “아무것도 없는 상태에서 테스트 코드부터 작성한다고? 이게 가능할까?” 라는 의문이 들었는데, 수업을 통해 직접 연습해보니 초기 요구사항을 명확히 해주고 확보된 TC들을 통해 결과적으로 디버깅이나 리팩토링 시간이 줄어드는 것을 경험했습니다. 향 후 실무에서도 큰 도움이 될 것 같습니다.
최새롬님	오랜만에 다양한 형태의 코딩을 진행하여 즐거운 시간이었습니다. 다양한 경험과 능력을 가진 분들과 함께 프로젝트를 진행하며 기존에 생각하지 못했던 많은 것들을 배울 수 있었습니다. 코드 리뷰와 협업을 빠른 속도로 유기적으로 진행하는 과정에서 협업에서 의사소통이 얼마나 중요한지, 그리고 코드리뷰가 어떤 역할을 할 수 있는지를 직접적으로 경험 할 수 있었습니다. 현업에서도 배운 것들을 잘 활용할 수 있도록 노력하겠습니다.
박소정님	Code Review Agent를 다녀온 후 양질의 리뷰를 달아주시는 파트분들을 보며 저도 교육에 입과하게 되었습니다. 팀에서 TDD 를 권장하다보니 세미나와 TDD News letter등을 통해 TDD의 이론, 필요성을 알고는 있었지만 실무에 접목시키기에는 어려움을 겪고 있었던 탓에 매우 의미있는 시간이었습니다. 실무에 돌아간 후 교육에서 배운 내용들을 잘 적용해보고 싶습니다 . :)
홍승표님	줄글 코드나 파일로 하는 버전 관리가 익숙한 저는 리팩토링이나 TDD 같은 과정들이 번거롭기만 한 과정이라고 생각했었지만, 프로젝트에서 교육 내용을 적용해보며 클린 코드와 TC가 요구사항 분석과 확장, 개발 속도에 얼마나 큰 영향을 주는지 직접 느낄 수 있었습니다. 현업에서도 다양한 코드에 직접 적용해보고 싶습니다!
이준태님	SSD 프로젝트를 진행하면서 TDD, 지속적인 Refactoring, CleanCode작성이 중요하다는 것을 느꼈습니다. 협업시에 규칙을 지키면서 하니 Code Review도 수월했고, 기능 확장 시 큰 구조 변경 없이 가능했습니다. 무엇보다 능력 있는 팀원들을 만나서 많이 배울 수 있는 기회였습니다. 감사합니다 :)