Create your value

Agile팀 프로젝트 결과 보고

Code review agent 과정(2024.06.03~06.25)

팀장 박상현 | 팀원 김정원, 서지웅, 이원희, 최한비, 허은지

조원 소개 및 역할

조원 소개 및 역할

박상현 | Shell, Logger, SSD Driver 구현 및 리팩토링

김정원 | Shell 구현 및 리팩터링, 코드리뷰

서지웅 | SSD Driver, Buffer 구현 및 리팩토링, 코드 리뷰

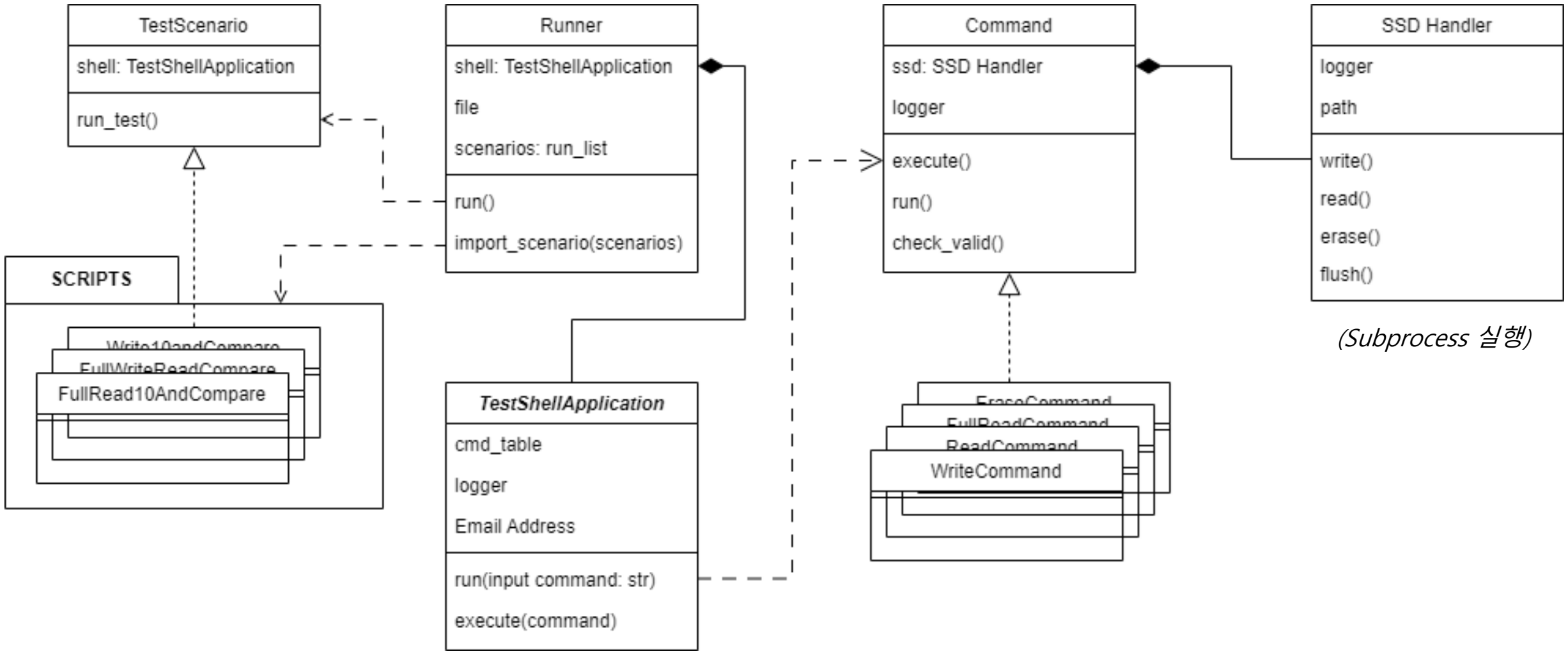
이원희 | Shell 구현 및 리팩터링, 코드리뷰

최한비 | Shell, Runner 구현 및 리팩터링

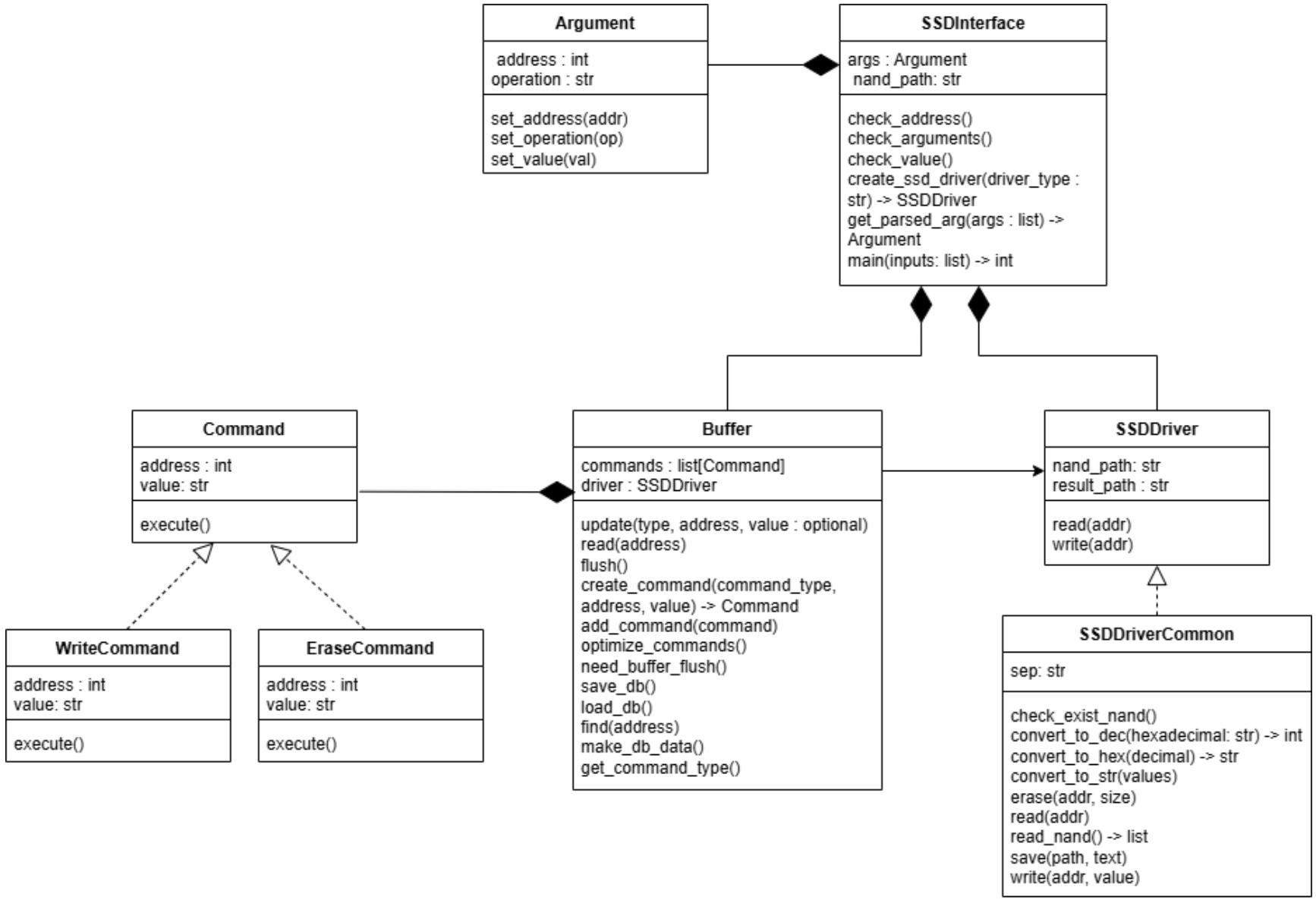
허은지 | SSD Interface, Buffer 구현 및 리팩터링, 코드리뷰

기능 구현 방식

Test Shell


TestScenario
shell: TestShellApplication
run_test()
Runner
shell: TestShellApplication
file
scenarios: run_list
run()
import_scenario(scenarios)
Command
ssd: SSD Handler
logger
execute()
run()
check_valid()
SSD Handler
logger
path
write()
read()
erase()
flush()
SCRIPTS
FullRead10AndCompare
TestShellApplication
cmd_table
logger
Email Address
run(input command: str)
execute(command)
WriteCommand
(Subprocess 실행)

SSD


Argument
address : int
operation : str
set_address(addr)
set_operation(op)
set_value(val)
SSDInterface
args : Argument
nand_path: str
check_address()
check_arguments()
check_value()
create_ssd_driver(driver_type : str) -> SSDDriver
get_parsed_arg(args : list) -> Argument
main(inputs: list) -> int
Command
address : int
value: str
execute()
Buffer
commands : list[Command]
driver : SSDDriver
update(type, address, value : optional)
read(address)
flush()
create_command(command_type, address, value) -> Command
add_command(command)
optimize_commands()
need_buffer_flush()
save_db()
load_db()
find(address)
make_db_data()
get_command_type()
SSDDriver
nand_path: str
result_path : str
read(addr)
write(addr)
WriteCommand
address : int
value: str
execute()
EraseCommand
address : int
value: str
execute()
SSDDriverCommon
sep: str
check_exist_nand()
convert_to_dec(hexadecimal: str) -> int
convert_to_hex(decimal) -> str
convert_to_str(values)
erase(addr, size)
read(addr)
read_nand() -> list
save(path, text)
write(addr, value)

추가미션 - Erase

- 기존 command에 *EraseCommand*, *EraseRangeCommand* 추가

```
class EraseCommand(Command):
    def check_valid(self, *args):
        if len(args) != 2:
            return False
        if not self.is_address_valid(args[0]):
            return False
        if not self.is_valid_size(*args):
            return False
        return True

    def run(self, *args):
        steps =
self.get_steps_with_size(int(args[0]),
int(args[1]))
        for n in range(len(steps) - 1):
            size = steps[n + 1] - steps[n]
            self.ssd.erase(steps[n], size)
```

- SSD에서 size 10 단위로 'E' 명령어 실행되게끔 구현

```
def get_steps_with_end(start, end):
    numbers = list(range(start, end, 10))
    if numbers and numbers[-1] != end:
        numbers.append(end)
    elif not numbers:
        numbers.append(end)
    return numbers
```

추가미션 - Logger

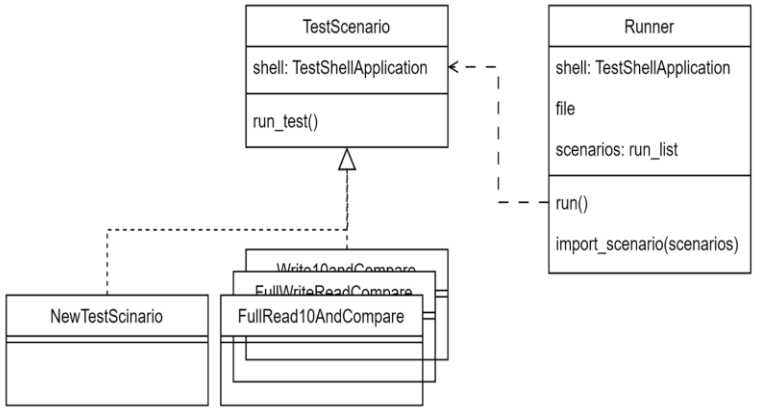
- Singleton pattern 기반의 Logger 기능 추가

```
class Logger(SingletonClass):
    def __init__(self):
        self.root = ROOT_DIR
        self.latest = Path(join(self.root, 'latest.log'))
        self.threshold = 10240  # 10KB = 10 * 1024 bytes

    def log(self, message):
        class_name = inspect.currentframe().f_back.f_locals['self'].__class__.__name__
        method_name = inspect.getframeinfo(inspect.currentframe().f_back).function + "()"
        self.save_latest(f"[{self.get_now()}] {class_name + '.' + method_name:<30}: {message}\n")
        if self.is_oversized_latest():
            self.save_oversized_log()
            self.covert_log_to_zip()
```

추가미션 - Runner & 시나리오 추가 효율화

- **run_list.txt** 명령 입력 시, txt 내 시나리오 자동 실행
  - 각 시나리오는 공통 interface를 갖는 개별 클래스로 구현
  - 지정된 위치에서 시나리오 모듈 검색 후 module load
    → ***재빌드 없이*** *사전 정의한 시나리오 테스트 가능*




```
def import_scenario(self, scenario_name):
    try:
        module_path = os.path.join(ROOT_DIR, "SHELL",
"SCRIPTS", f"{scenario_name}.py")
        spec = spec_from_file_location(scenario_name,
module_path)
        module = module_from_spec(spec)
        spec.loader.exec_module(module)

        return getattr(module,
scenario_name)(self.shell)

    except FileNotFoundError:
        self.print_fail()
```

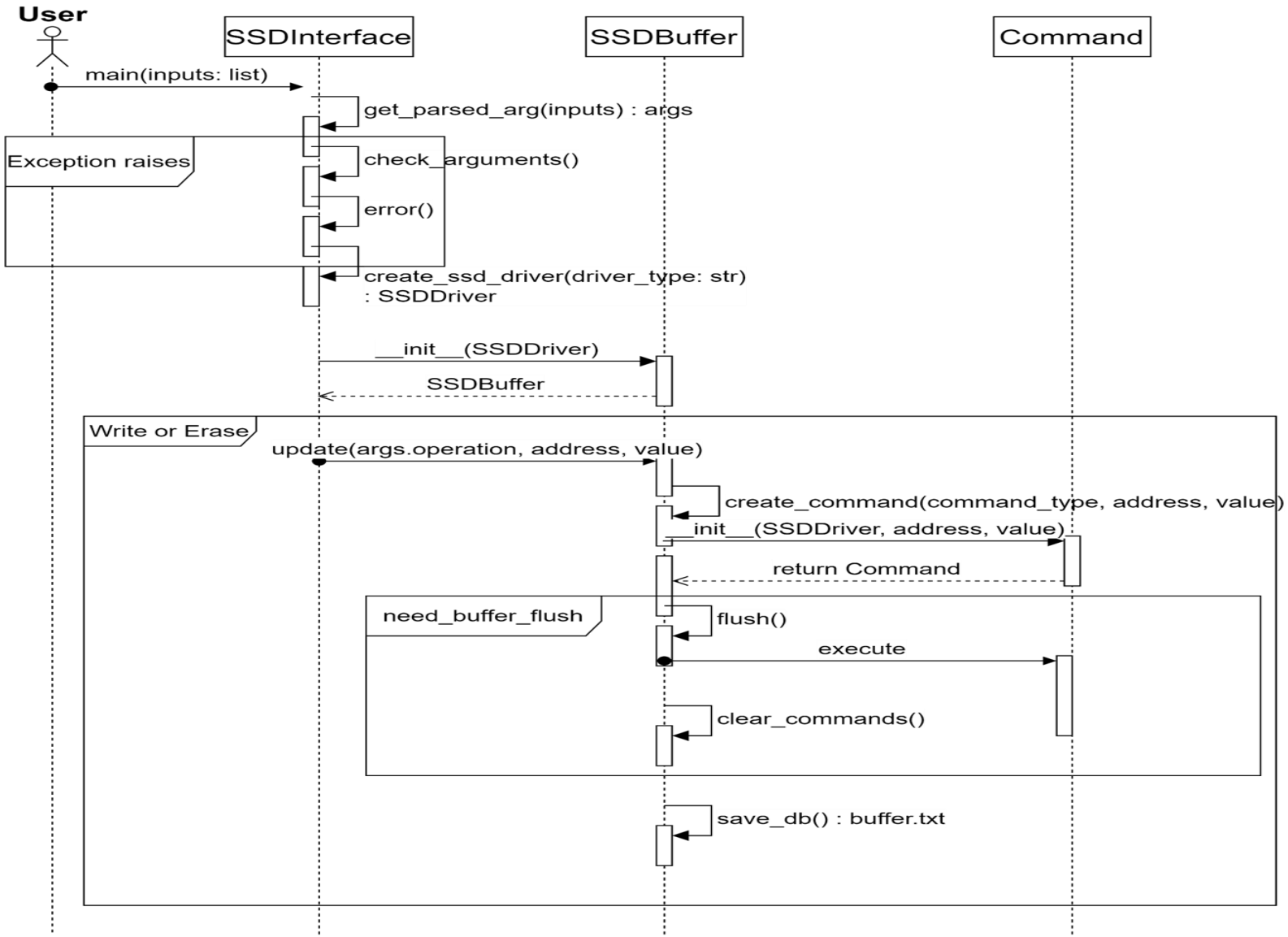
```
class FullRead10AndCompare(TestScenario):
    def run_test(self):
        f = io.StringIO()
        with redirect_stdout(f):
            self.shell.run("fullread")
        output1 = f.getvalue()
        with redirect_stdout(f):
            for i in range(2):
                self.shell.run("fullread")
            output2 = f.getvalue()

        if output1 * 3 == output2:
            return True
        return False
```

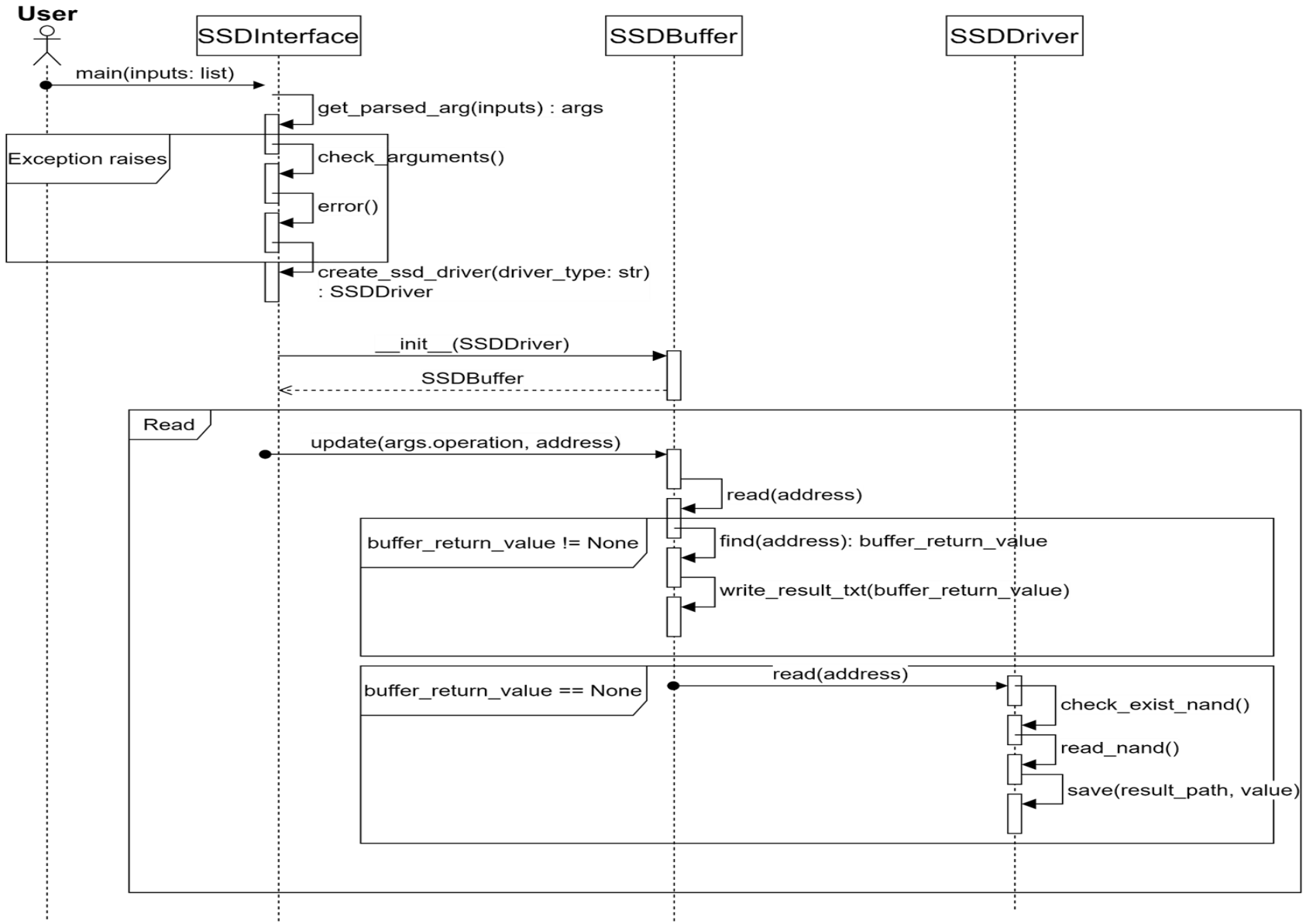
추가미션 - Command buffer

- **요구사항**
  - **Flush**
    - Write와 Erase 명령어들을 Buffer에 최대 10개 저장한 후 10개 초과 시에 한꺼번에 명령어를 수행한다.
  - **Fast Read**
    - Read를 수행할 때 Command Buffer부터 확인하여 NAND를 읽지 않고도 값을 리턴할 수 있다.

추가미션 - Command buffer (Flush)



추가미션 - Command buffer (Fast Read)



추가미션 - Command buffer 최적화

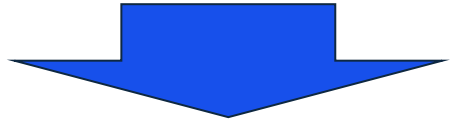
- **요구사항**
  - Command Buffer에 저장되어 있는 Command 개수가 최소가 되도록 최적화

  - 최적화 알고리즘 설명
    - command를 모두 수행하고 난 이후 각 address에 최종 수행된 결과를 통해 압축
    - 명령이 연속되어있다면, Erase 명령을 길이 10 이하로 묶어 저장
    - Erase 이후 write 명령 수행

추가미션 - Command buffer 최적화

명령/주소	0	1	2	3	4	5	6
E 1 1		E					
E 3 1				E			
E 5 1						E	
W 2			W				
W 4					W		

최종 결과		E	W	E	W	E	



명령/주소	0	1	2	3	4	5	6
E 1 5		E	E	E	E	E	
W 2			W				
W 4					W		

기존 명령
E 1 1 / E 3 1 / E 5 1 / W 2 / W 4 (5개)

최적화 이후 명령
E 1 5 / W 2 / W 4 (3개)

추가미션 - Command buffer 최적화

edge case : Erase를 중간에 끊어야 최적이 되는 경우

input command : E 0 10 / W 10 ... W 16 / E 13 3 / E 16 7 (10개)

최종 결과
EEEEEEEEEEWWWEEEEEEEEEE

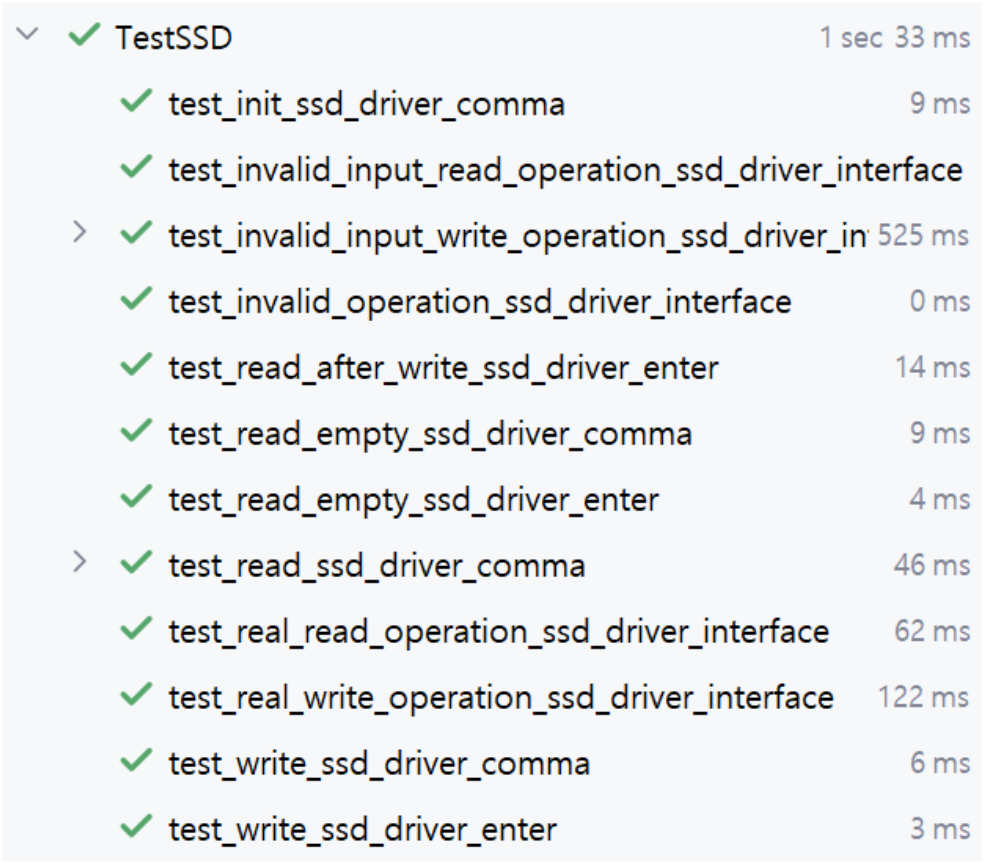
예외처리를 수행하지 않은 알고리즘 수행 결과
E 0 24 / W 10 ... W 12 (6개)

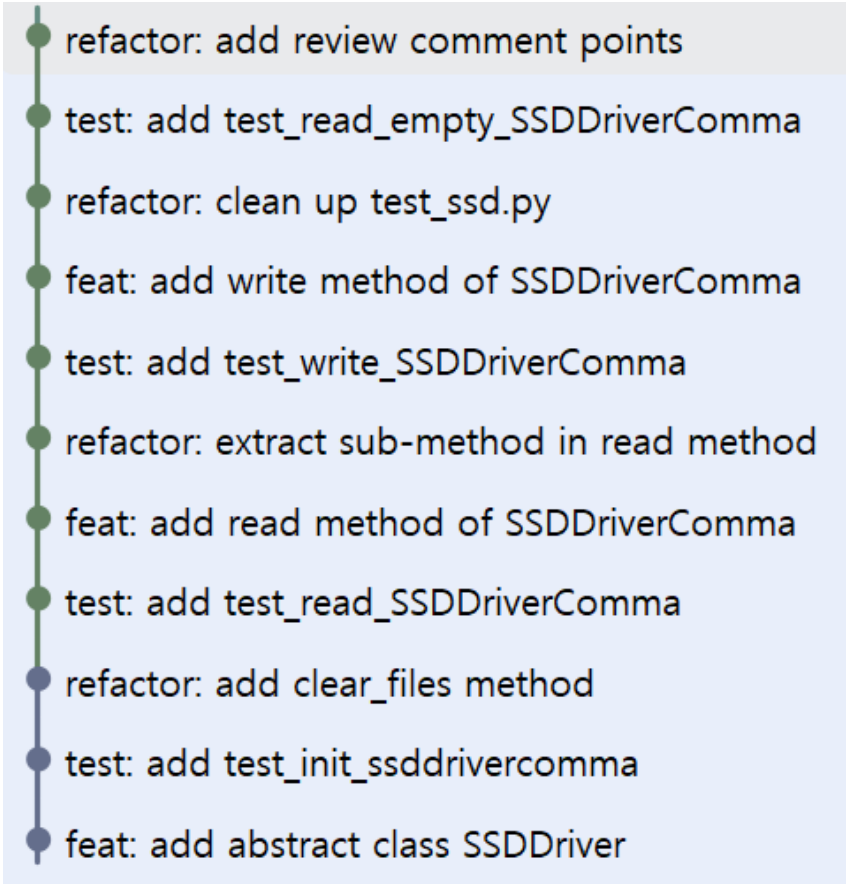
예외처리를 수행한 알고리즘 수행 결과
E 0 10 / W 10 ... W 12 / E 13 10 (5개)

TDD 및 Mocking 활용 예시

TDD & Mocking

- SSD 모듈, Test Shell 개발시 TDD cycle을 통해 개발





TDD & Mocking

- 초기 Test shell 개발 시 SSD 모듈과 Test Shell을 동시 개발
  - SSD 모듈을 Mock으로 대체해 unittest 수행
  - Test shell의 command 코너 케이스 검증하며 개발

```
class TestTestShellApplication(TestCase):
    def setUp(self):
        super().setUp()
        self.mk_ssd = Mock()
        self.shell = TestShellApplication(self.mk_ssd)

    def test_verify_write_invalid_address(self):
        self.assertEqual(False, self.shell.run("write 100 OxAAAABBBB"))
        self.assertEqual(False, self.shell.run("write -1 OxAAAABBBB"))
        self.assertEqual(False, self.shell.run("write 0x11 OxAAAABBBB"))

    def test_verify_write_invalid_data(self):
        self.assertEqual(False, self.shell.run("write 3 OxAAAABBBB"))
        self.assertEqual(False, self.shell.run("write 3 0xAABBBB"))
        self.assertEqual(False, self.shell.run("write 3 0xAAAABBBBCC"))
```

TDD & Mocking

- 초기 SSD Interface 개발 시 SSDDriver와 동시 개발
  - SSDDriver 모듈을 Mock으로 대체해 unittest 수행

```
@patch.object(SSDApplication, "create_ssd_driver")
def test_main_write(self, mk_driver_factory):
    mk = self.create_mock_ssd_driver()
    mk_driver_factory.return_value = mk
    ret = self.app.main(["W", "0", "0x12345678"])
    self.assertEqual(ret, True)
    self.assertEqual(mk.write.call_count, 1)

def create_mock_ssd_driver(self):
    mk = Mock(spec=SSDDriver)
    mk.read.side_effect = "driver : read"
    mk.write.side_effect = "driver : write"
    return mk
```

TDD & Mocking

- SSD 모듈 개발 이후, SSD 모듈과 Test Shell을 연계하여 통합 unittest 작성
  - print 함수를 Mocking하여 Console 출력 및 동작 확인

```
@patch('sys.stdout', new_callable=StringIO)
def test_valid_fullwrite(self, mock_stdout):
    # arrange
    sample_points = [0, 49, 99]
    tc = [["fullwrite", "0xAAAABBBB"],
          ["fullwrite", "0x00000001"]]
    for n in range(len(tc)):
        with self.subTest(f"subtest {n}"):
            # action
            self.run_testcase(tc[n])
            # assert
            for p in sample_points:
                self.check_right_data(p, \
                            tc[n][1], mock_stdout)
            self.clear_test_files()
```

```
@patch('sys.stdout', new_callable=StringIO)
def test_invalid_fullwrite_address(self, mock_stdout):
    # arrange
    sample_points = [0, 49, 99]
    tc = [["fullwrite", "50", "OxAAAABBBB"],
          ["fullwrite", "0xAABBFFTT"]]

    for n in range(len(tc)):
        with self.subTest(f"subtest {n}"):
            # action
            self.run_testcase(tc[n])
            # assert
            self.check_stdout("INVALID COMMAND", mock_stdout)
            for p in sample_points:
                self.check_right_data(p,\
                            self.convert_to_hex(0),mock_stdout)
            self.clear_test_files()
```

TDD & Mocking

- Buffer 개발 시, Buffer Erase Command 동작 검증을 위해 Mocking 응용

```
@staticmethod
def erase(address: int, value: int):
    if value > 10:
        raise Exception
    return

def test_erase_over_ten(self):
    self.ssd_driver = Mock()
    self.ssd_driver.erase.side_effect = self.erase
    erase_command = EraseCommand(self.ssd_driver, 1, 20)
    with self.assertRaises(Exception):
        erase_command.execute()
        self.fail()
    self.assertEqual(self.ssd_driver.erase.call_count, 2)
```

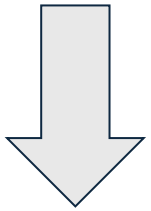
driver에 전달된 value가 10이 넘는지 확인

리팩터링 Before & After

Test shell : Command pattern 적용

Before

- Test shell 클래스 내 메서드로 모든 command 구현
- 입력 command에 대한 유효성 검사를 Test shell 클래스 내에서 수행



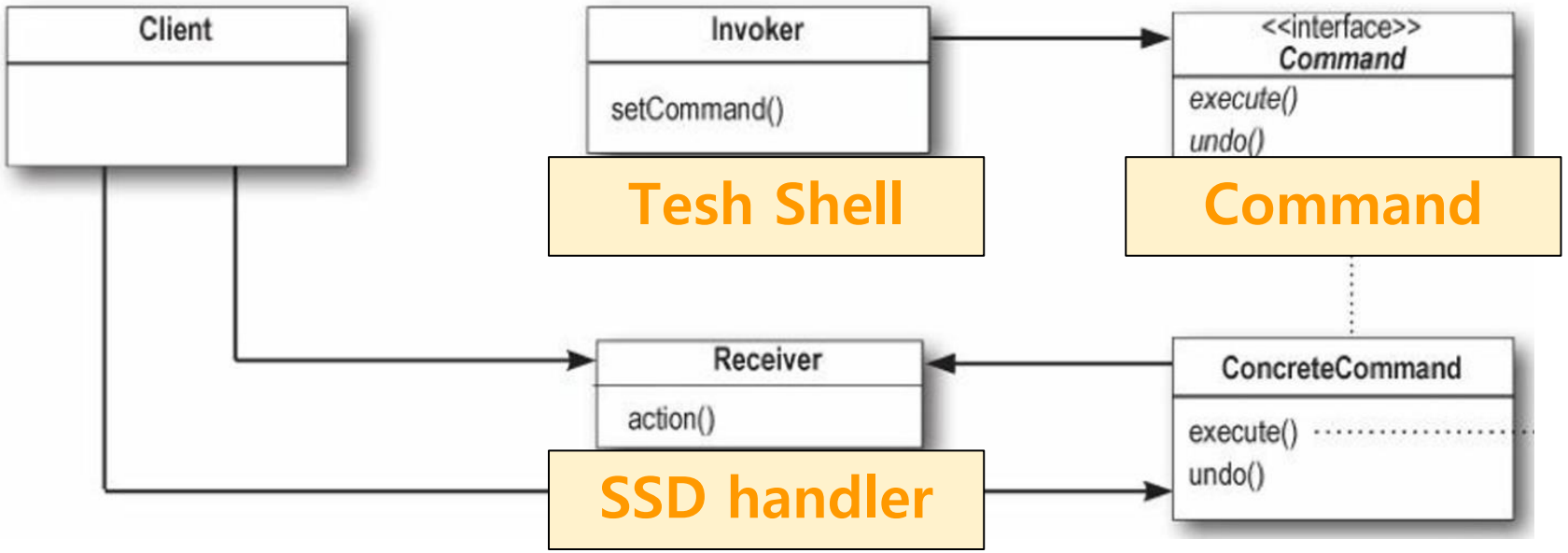
(-) 중복 코드 다수 발생
(-) command 추가 시 코드 수정 복잡

Test shell : Command pattern 적용

After

- Command interface와 상속 command 클래스들로 구성된 파일을 별도 구성, Test shell에서 입력된 명령에 해당하는 command를 통해 ssd 명령 실행

(+) command 추가 및 수정이 간편함
(+) command 별 구현 가독성 높임



Test shell : Command pattern 적용

Before

```
def go_execution(self):
    if self.execution == WRITE_CODE:
        return self.write()
    elif self.execution == READ_CODE:
        return self.read()
    elif self.execution == FULLWRITE_CODE:
        return self.fullwrite()
    elif self.execution == FULLREAD_CODE:
        return self.fullread()
    elif self.execution == HELP_CODE:
        return self.help()
    elif self.execution == TEST_APP_2:
        return self.test_app_2()
```

After

```
class Command(ABC):
    def __init__(self):
        self.ssd = SSDHandler()
        self.logger = Logger()

    def execute(self, *args):
        if not self.check_valid(*args):
            print('INVALID COMMAND')
            return
        self.run(*args)

    @abstractmethod
    def run(self, *args):
        pass

    @abstractmethod
    def check_valid(self, *args):
        pass
```

Test shell : Command pattern 적용

Before

```
def is_valid_command(self, input_command_elements: list):
    if len(input_command_elements) > 3 or len(input_command_elements) <= 0:
        return False

    if input_command_elements[0] == WRITE_CODE:
        if len(input_command_elements) != 3:
            return False
        if not self.is_valid_address(input_command_elements[1]):
            return False
        if not self.is_valid_data_format(input_command_elements[2]):
            return False
        return True
    elif input_command_elements[0] == READ_CODE:
        if len(input_command_elements) != 2:
            return False
        if not self.is_valid_address(input_command_elements[1]):
            return False
        return True
    elif input_command_elements[0] == FULLWRITE_CODE:
        if len(input_command_elements) != 2:
            return False
        if not self.is_valid_data_format(input_command_elements[1]):
            return False
        return True
    elif input_command_elements[0] == FULLREAD_CODE:
        if len(input_command_elements) != 1:
            return False
        return True
    (...)
    return False
```

After

```
class WriteCommand(Command):
    def __init__(self):
        super().__init__()

    def check_valid(self, *args):
        if len(args) != 2:
            return False
        if not self.is_address_valid(args[0]):
            return False
        if not self.is_data_valid(args[1]):
            return False
        return True

    def run(self, *args):
        self.ssd.write(*args)
```

SSD Buffer : Command pattern 적용

```
class Command(ABC):
    def __init__(self, driver: SSDDriver, address: int, value: Union[str, int]):
        self.ssd_driver = driver
        self.logger = Logger()
        self.address = address
        self.value = value

    @abstractmethod
    def execute(self):
        pass


class WriteCommand(Command):
    def execute(self):
        try:
            self.ssd_driver.write(self.address, self.value)
            self.logger.log(f'success!')
        except Exception:
            self.logger.log(f'fail! Error 내용 : {str(Exception)}')
```

Command pattern 적용으로 Client는 내부 동작 구조를 알 필요가 없게 함

```
def flush(self):
    for command in self.commands:
        command.execute()
```

SSD Buffer : Factory pattern 적용

```
def create_command(self, command_type: str, address: int, value: str):
    if command_type == CMD_WRITE:
        return WriteCommand(self.driver, address, value)
    elif command_type == CMD_ERASE:
        return EraseCommand(self.driver, address, value)
    self.logger.log(f"invalid command type {command_type}")
```

Client단 사용 예시

```
command = self.create_command(command_type, address, value)
```

코드 동작 시연

시연 순서

- **Runner :** run_list.txt
- **special command:** testapp1, testapp2
- **normal command**
  - read 3
  - write 3 0xAAAABBBB | read 3
  - fullwrite 0x1234CDEF
  - erase 33 20 | read 52
  - erase_range 10 20 | read 19
  - help
  - exit
- **오류 command**
  - (none)
  - write 1
  - read abc
  - erase 95 10
  - erase_range 99 95
- **Bufffer 작동 확인**
- **log파일 확인**

소감

2주간 동에듀로 출퇴근한 모든 분들 고생 많으셨습니다.
CRA는 현업부터 시작이라고 하던데, 모두모두 화이팅입니다!

3주 동안 다들 열심히 공부하시느라 고생하셨습니다. 😊
앞으로 이번에 배운 지식을 활용해서 이쁜 코드 작성하겠습니다~

훌륭한 팀원분들께 많이 배웠습니다. 다들 고생하셨고 현업 과제도 화이팅입니다!

코드리뷰 문화를 경험할 수 있었고, 각 분야에서 활약중인 S/W 전문가들과 같이 프로젝트 수행해보는 값진 경험이었습니다.

그동안 고생 많으셨습니다.
코드리뷰 문화를 체험하니 얼른 저희 팀에 전파해야겠다는 생각이 듭니다..