

# Device Solution

SSD Project (25' 6. 10 ~ 6. 16)

# 조원소개

## 역할

김영식(팀장) : TestShell, 발표

권성민 : File 관리 기능, Command Buffer, 협업 Tool Setup

권희정 : SSD 관리 기능, Command Buffer

박준경 : SSD 컨트롤러 기능

오시훈 : TestScript, Logger, Command Buffer 알고리즘

이상훈 : TestShell, 통합 Test

추준성 : TestShell 컨트롤러, Runner

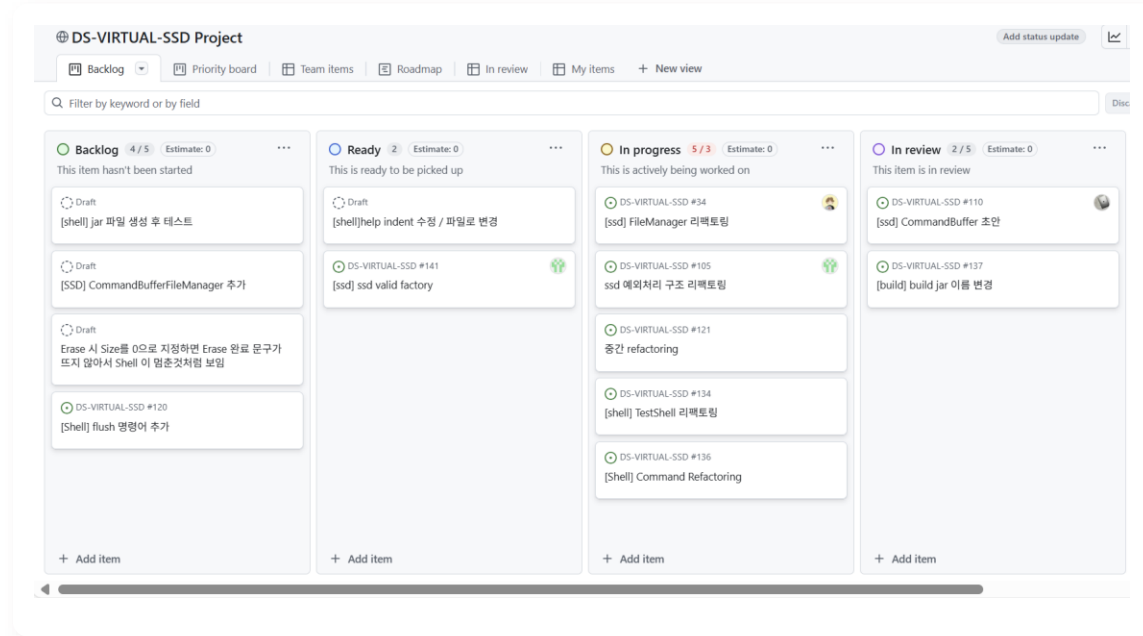
## Clean Code

- 가독성 좋은 코드
- 안전한 코드
- 유지보수성이 좋은 코드

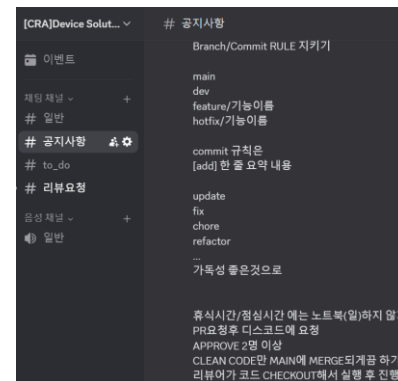


# 협업관리

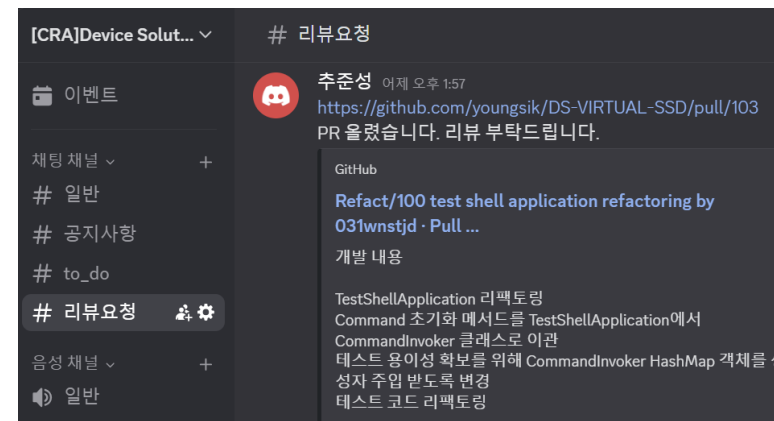
## Task 관리



## Communication



#공지사항 - Grond Rule 공유




#리뷰요청 - PR 시 알림

# 협업관리


## Task Process

**Backlog** 2 / 5 Estimate: 0 ...

This item hasn't been started

 Draft

[SSD] CommandBufferFileManager 추가

 Draft **More actions**

Erase 시 Size를 0으로 지정하면 Erase 완료 문구가 뜨지 않아서 Shell 이 멈춘것처럼 보임


1


### Backlog 이슈 생성

- 기능 및 변경 사항 추가
- 버그 문제
- 코드 리팩토링

**Ready** 1 Estimate: 0 ...

This is ready to be picked up

 DS-VIRTUAL-SSD #141

[ssd] ssd valid factory 


2


### 이슈 할당

- 적극적 이슈 할당
- 충돌 사전 방지

**In progress** 6 / 3 Estimate: 0 ...

This is actively being worked on

 DS-VIRTUAL-SSD #34

[ssd] FileManager 리팩토링 


3


### TDD 개발

- 유닛 테스트는 필수
- 질문이 있을 경우 손!

**In review** 1 / 5 Estimate: 0 ...


This item is in review

 DS-VIRTUAL-SSD #110

[ssd] CommandBuffer 초안 

**Done** 84 Estimate: 0 ...

This has been completed

 DS-VIRTUAL-SSD #134

[shell] TestShell 리팩토링

4

### PR + 리뷰 = Merge

- 리뷰는 정확하게
- 오픈 마인드로 리뷰 수용
- New 이슈 생성 선순환

## 최소 놀람의 원칙

- 알고 보니 팀원과 동일한 문제를 다루고 있네.
- 알고 보니 이미 해결된 문제를 개발하고 있네.
- 알고 보니 아키텍처가 크게 바뀌었어.
- 알고 보니 내 리뷰를 기다리고 있는 PR이 5개야.
- 알고 보니 Ground Rule을 지키지 못했구나.

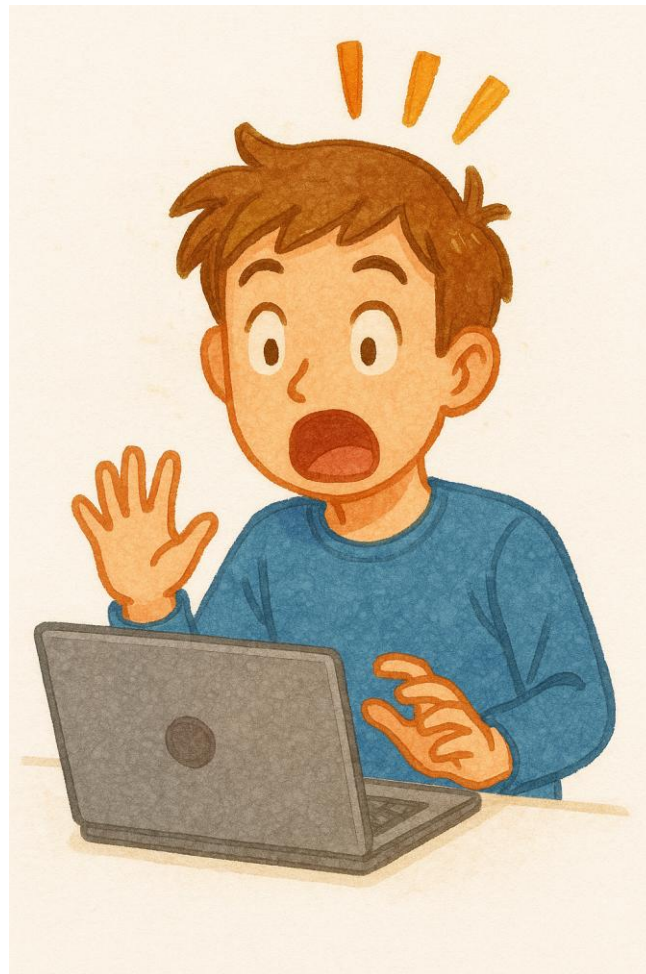


Kanban Board



Discord

협업 툴을 활용한 **놀람** 주의 사전 **예방**



# 협업관리

June 8, 2025 – June 15, 2025

Period: 1 week ▾

## Overview

86 Active pull requests

109 Active issues

86

Merged pull requests

0

Open pull requests

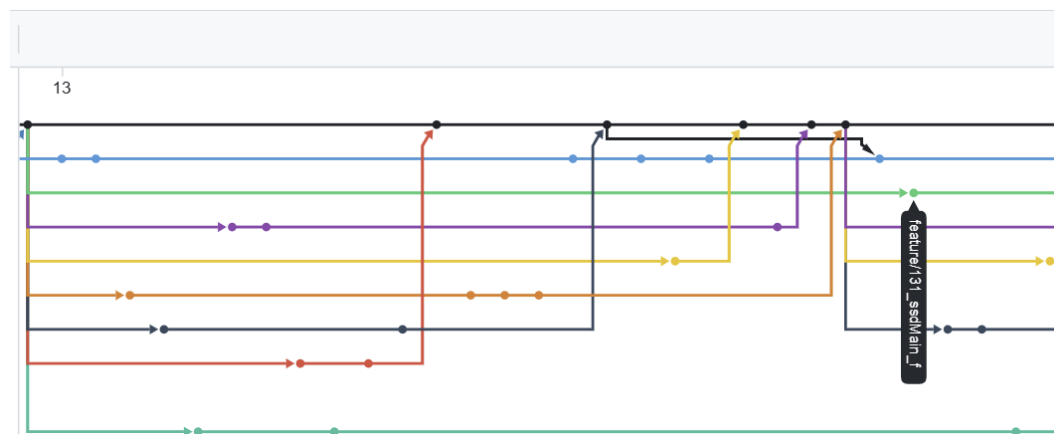
109

Closed issues

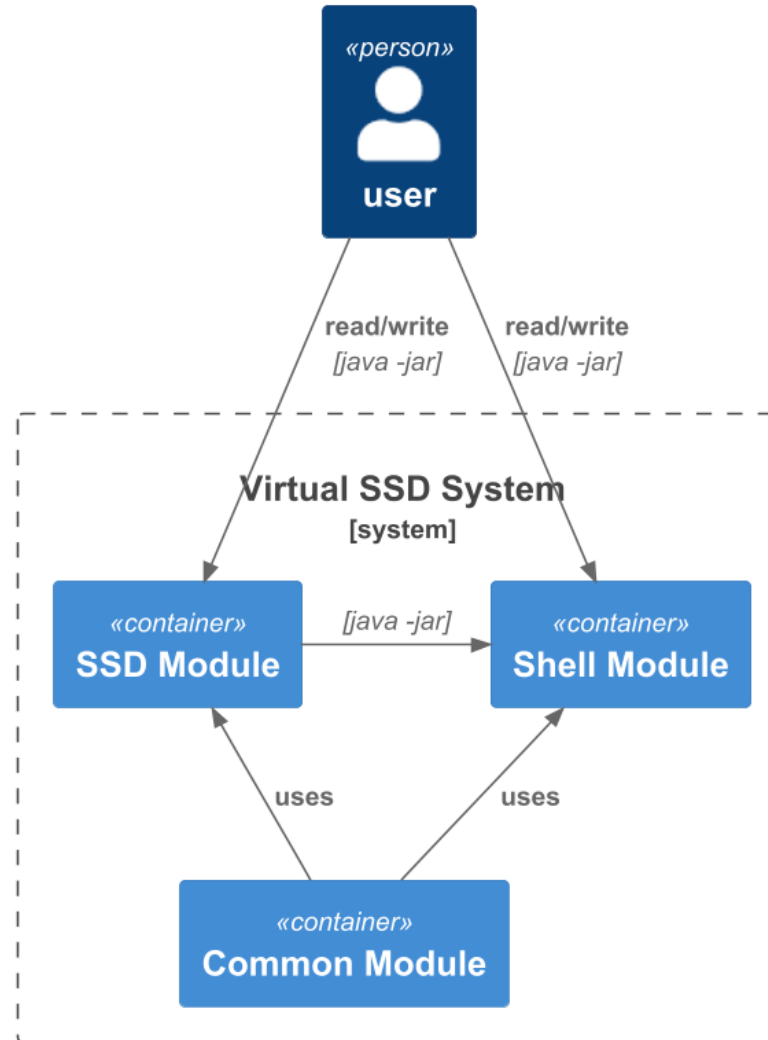
0

New issues

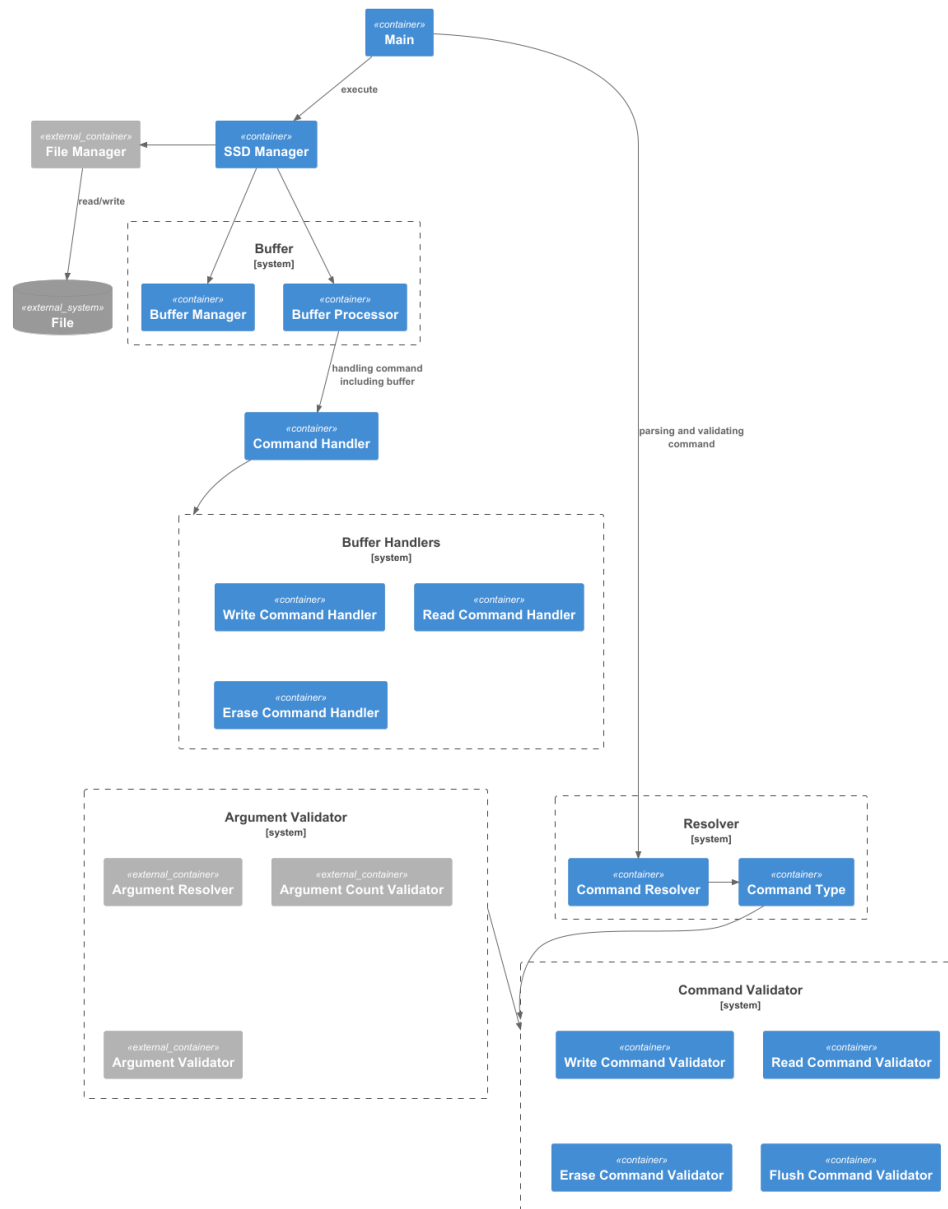
Excluding merges, 10 authors have pushed **385 commits** to main and **385 commits** to all branches. On main, **0 files** have changed and there have been **0 additions** and **0 deletions**.



# 기능구현

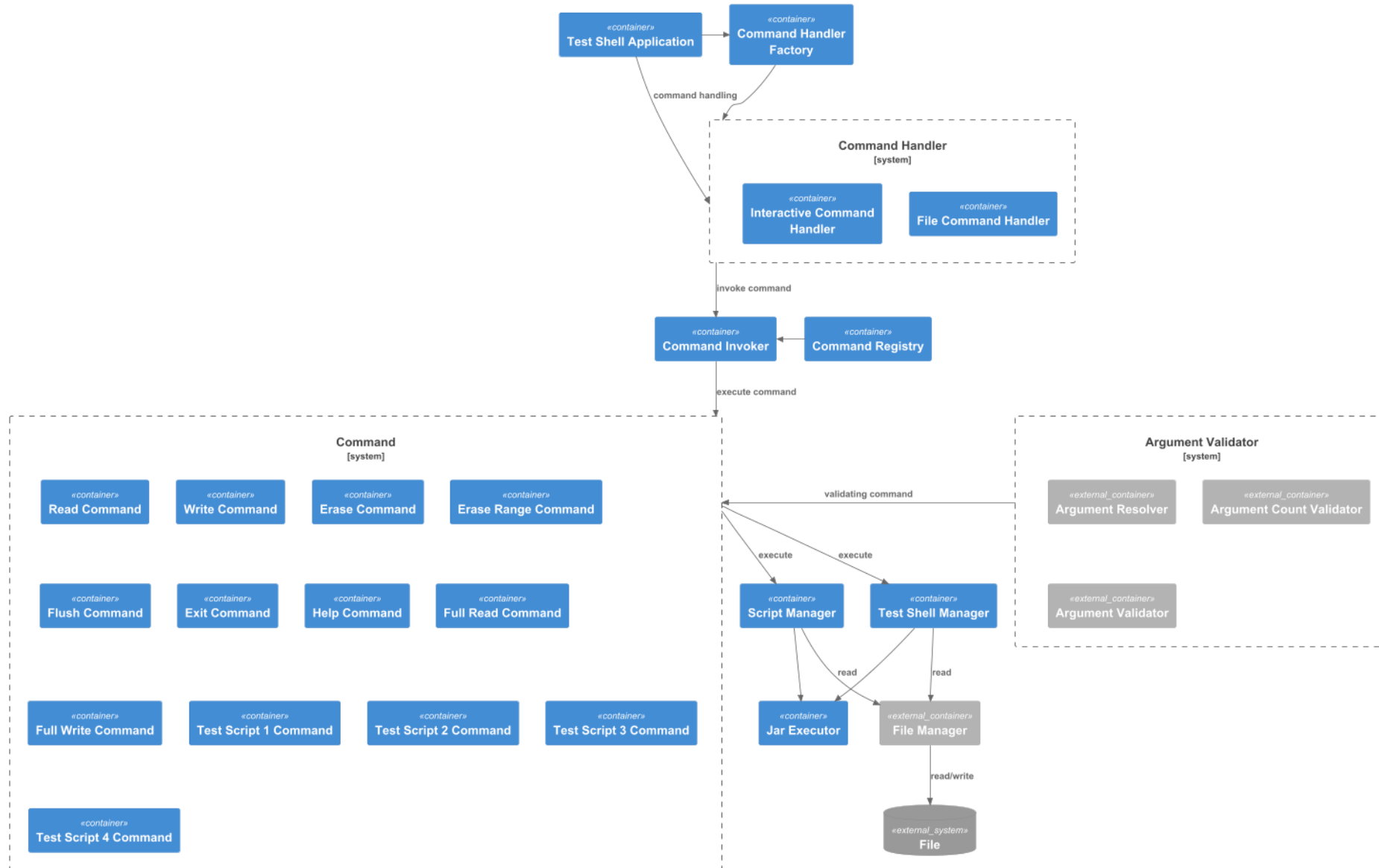


# 기능구현 [SSD Module]





# 기능구현 [Shell Module]



# TDD 활용예제

## Feature/26 test script #43

Merged dolgogae merged 23 commits into main

Conversation 4 Commits 23 CI

Commits on Jun 11, 2025

### [test] ScriptManager test

dolgogae committed 2 days ago

### [feat] ScriptManager

dolgogae committed 2 days ago

### [refactor] ScriptManager

dolgogae committed 2 days ago

### [test] TestScript1Command test

dolgogae committed 2 days ago

### [feat] TestScript1Command

dolgogae committed 2 days ago

### [feat] move logic to ScriptManager

dolgogae committed 2 days ago

### [test] TestScript2Command test

dolgogae committed 2 days ago

### [feat] TestScript2Command

dolgogae committed 2 days ago

### [refactor] ScriptManager

dolgogae committed 2 days ago

## RED

```
public class ScriptManagerTest {
    @Mock
    SsdApplication ssdApplication;

    @Test
    void createSscScriptManager(){
        ScriptManager scriptManager = new ScriptManager(ssdApplication);

        assertNotNull(scriptManager);
    }

    @Test
    void callSsdApplicationReadTest(){
        String expected = "0xAAAABBBB";
        ScriptManager scriptManager = new ScriptManager(ssdApplication);
        when(ssdApplication.execute("R 1")).thenReturn(expected);

        String result = scriptManager.read(1);

        verify(ssdApplication).execute("R 1");
        assertEquals(expected, result);
    }
}
```

Test Script 개발 전, Read  
기능의 입출력과 예상 값을  
바탕으로 **실패**하는 테스트를  
**먼저** 작성하는 단계

## GREEN

```
public class ScriptManager {
    private final SsdApplication ssdApplication;

    public ScriptManager(SsdApplication ssdApplication) {
        this.ssdApplication = ssdApplication;
    }

    public String read(Integer lba){
        String command = "R " + lba;
        return ssdApplication.execute(command);
    }
}
```

테스트를 **통과**시키는 것이  
목표이며, 복잡한 구조 없이  
**최소한**의 코드로 **동작**하게  
만드는 단계

## REFACTOR

```
public class ScriptManager {
    private static final String READ_PREFIX_COMMAND = "R ";
    private static final String WRITE_PREFIX_COMMAND = "W ";
    private final SsdApplication ssdApplication;

    public ScriptManager(SsdApplication ssdApplication) {
        this.ssdApplication = ssdApplication;
    }

    public String read(Integer lba){
        String command = "R " + lba;
        String command = READ_PREFIX_COMMAND + lba;
        return ssdApplication.execute(command);
    }
}
```

테스트의 **신뢰성**을 해치지  
않으면서 직관적이지 않은  
하드코딩 값을 상수로 바꾸는 등  
코드를 **정리**하는 단계

# Mocking 활용

파일 시스템 처리를 담당하는 FileManager를 Mocking으로 대체하여 외부 의존성과 미구현 모듈에 대응함

## SUT 준비

```
@Mock 5 usages
FileManager fileManager;

@BeforeEach @ dean +1
void setUp() {
    System.setOut(new PrintStream(outContent));
    testShellManager = new TestShellManager(jarExecutor, fileManager);
}
```

## STUB

```
void testFullreadOutput() {
    List<String> fakeData = new ArrayList<>();
    fakeData.add("0xFFFFFFFF");
    fakeData.add("0x12345678");
    for (int i = 0; i < 100; i++) {
        fakeData.add("0x00000000");
    }
    when(fileManager.getResultFromOutputFile()).thenReturn(fakeData.get(0))
        .thenReturn(fakeData.get(1))
        .thenReturn(fakeData.get(2));
}
```

# Mocking 활용

테스트 수행

```
testShellManager.fullread();
```

Behavior  
Verification

```
verify(fileManager, times(wantedNumberOfInvocations: 100)).getResultFromOutputFile();
```

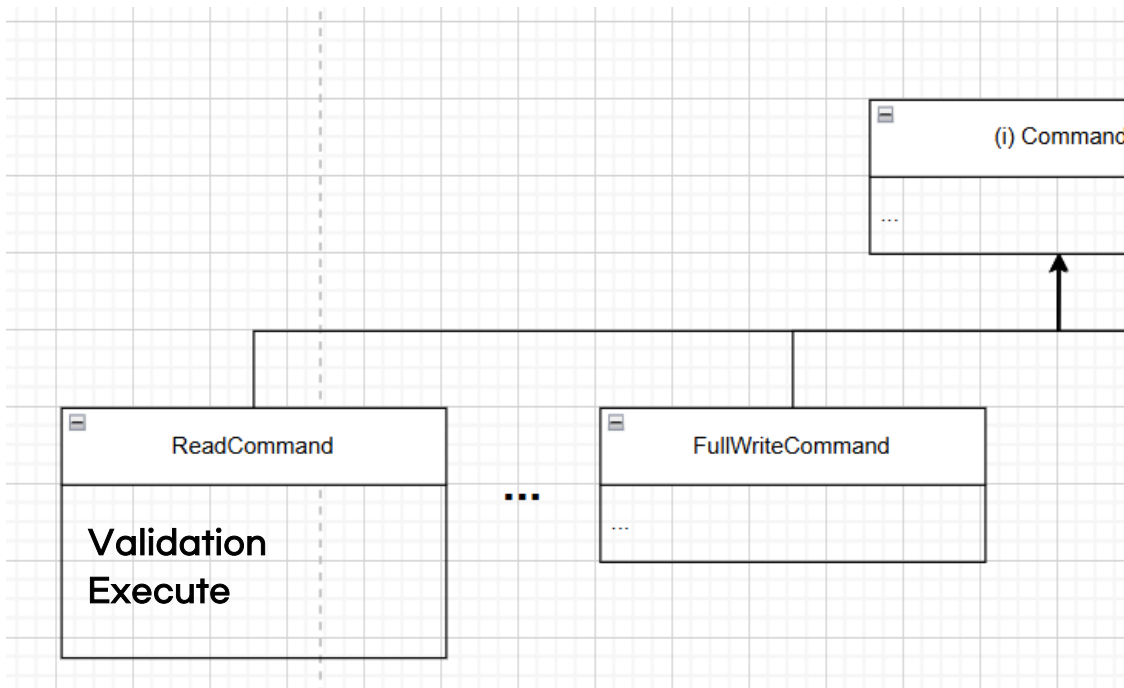
State  
Verification

```
String[] outputLines = outContent.toString().trim().split(regex: "\n");
assertThat(outputLines).hasSize(expected: 100);
assertThat(outputLines[0].replace(target: "\n", replacement: "").replace(target: "\r", replacement: "").trim())
    .isEqualTo(expected: "[Full Read] LBA 00 0xFFFFFFFF");
assertThat(outputLines[1].replace(target: "\n", replacement: "").replace(target: "\r", replacement: "").trim())
    .isEqualTo(expected: "[Full Read] LBA 01 0x12345678");
for (int i = 2; i < 100; i++) {
    String expected = String.format("[Full Read] LBA %02d 0x00000000", i);
    assertThat(outputLines[i].replace(target: "\n", replacement: "").replace(target: "\r", replacement: "").trim())
        .isEqualTo(expected);
}
```

# 리팩토링 [1]

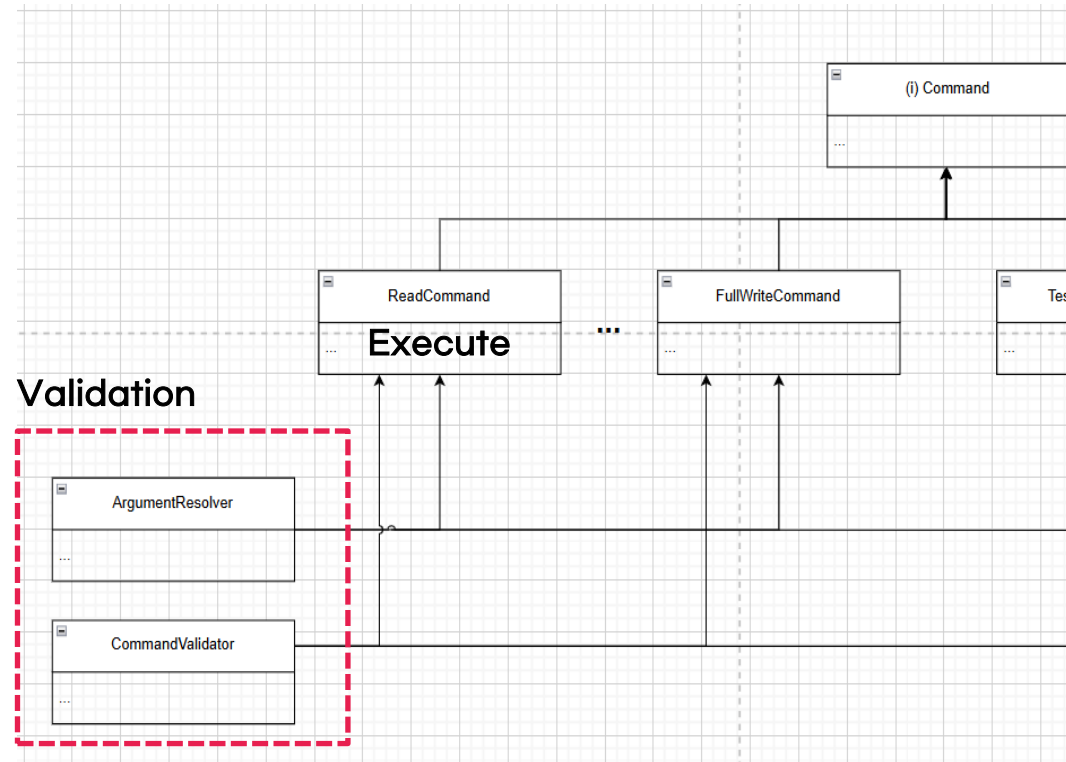
- Smell - Long Method (불필요한 복잡성)
- 하나의 책임만 수행 - 단일 책임 원칙(Single Responsibility Principle, SRP) 적용

## Before



Command 모듈은 Validation과 Execute 기능을 모두 포함

## After



Command 모듈은 Execute 기능만 전담하도록 설정하고,  
Validation 기능은 별도의 패키지 로 분리

# 리팩토링 [1]

## Test Shell - EraseRange Validation

```
5 private final TestShellManager testShellManager;
6
7 public EraseRangeCommand(TestShellManager testShellManager) {
8     @@ -10,28 +17,17 @@ public EraseRangeCommand(TestShellManager testShellManager) {
9
10 Command의 파라미터 유효 수 체크
11 @Override
12 public void execute(String[] cmdArgs) {
13     if(cmdArgs.length != 3) {
14         throw new RuntimeException("INVALID COMMAND PARAMETER");
15     }
16
17     int beginLBA;
18     int endLBA;
19
20     try{
21         beginLBA = Integer.parseInt(cmdArgs[1]);
22         endLBA = Integer.parseInt(cmdArgs[2]);
23     } catch(NumberFormatException e) {
24         throw new RuntimeException("INVALID COMMAND PARAMETER");
25     }
26
27     if(!isLBAValid(beginLBA) || !isLBAValid(endLBA)) {
28         throw new RuntimeException("INVALID COMMAND PARAMETER");
29     }
30
31     testShellManager.eraseRange(beginLBA, endLBA);
32 }
33
34 private boolean isLBAValid(int LBA) {
35     return LBA >= 0 && LBA < 100;
```

Argument 가능 값 (0~99) 체크

```
8 +
9 + private static final int BEGIN_LBA_INDEX = 1;
10 + private static final int END_LBA_INDEX = 2;
11 +
12 private final TestShellManager testShellManager;
13
14 public EraseRangeCommand(TestShellManager testShellManager) {
```

CommandValidator 수행

```
17
18 @Override
19 public void execute(String[] cmdArgs) {
20     CommandValidator.validateThreeArgs(cmdArgs);
21     testShellManager.eraseRange(
22         extractValidatedBeginLba(cmdArgs),
23         extractValidatedEndLba(cmdArgs));
24 }
```

ArgumentResolver 수행

```
25
26 private Integer extractValidatedBeginLba(String[] cmdArgs) {
27     return
28     ArgumentResolver.resolveLba(cmdArgs[BEGIN_LBA_INDEX]);
29 }
30 private Integer extractValidatedEndLba(String[] cmdArgs) {
31     return ArgumentResolver.resolveLba(cmdArgs[END_LBA_INDEX]);
```

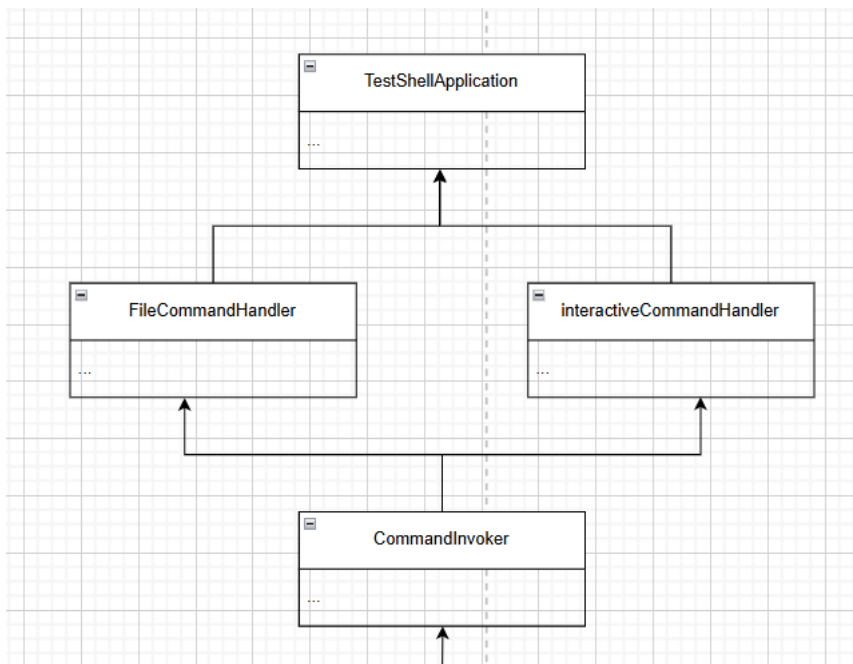
# 리팩토링 [2]

## ■ Factory Method Pattern 적용 : CommandHandlerFactory

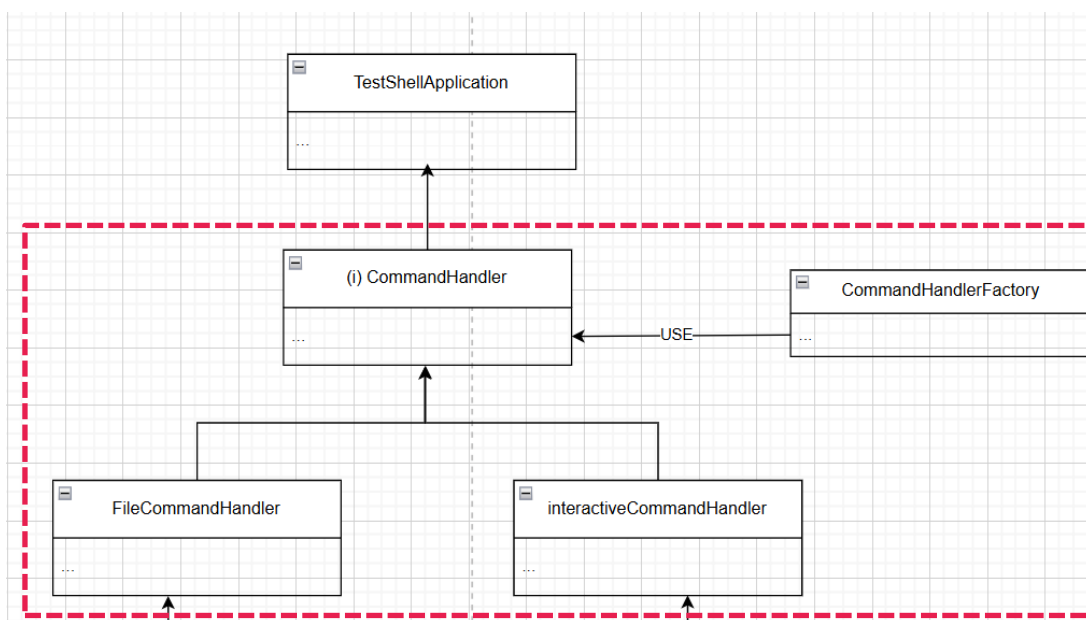
OCP (Open and Closed Principle) - 확장될 가능성이 있는 Command Handler는 interface화 하여 유지보수 용이하도록 변경

DIP (Dependency Inversion Principle) - TestShellApp이 Interface에 의존하도록 변경 (강한 의존성 낮춤)

### Before



### After



# 리팩토링 [2]

```
11 public class TestShellApplication {
12 + public static void main(String[] args) {
13 -     CommandInvoker commandInvoker = new
14 -     CommandInvoker(new HashMap<>());
15 -     commandInvoker.initAllCommands();
16 -
17 -     try {
18 -         if (args.length != 0 && args[0] !=
19 -         null) {
20 -             FileCommandHandler
21 -             fileCommandHandler = new
22 -             FileCommandHandler(commandInvoker);
23 -             fileCommandHandler.handle(args[0]);
24 -         } else {
25 -             InteractiveCommandHandler
26 -             interactiveCommandHandler = new
27 -             InteractiveCommandHandler(commandInvoker);
28 -             interactiveCommandHandler.handle();
29 -         }
30 -     } catch (Exception e) {
31 -     }
32 - }
```

```
8 public class TestShellApplication {
9 public static void main(String[] args) {
10 +     CommandHandler handler =
11 +     CommandHandlerFactory.getCommandHandler(args);
12 +     handler.handle(args);
13 }
```



객체 생성 기능 -> Factory로 위임

Command Handler 가 추가되더라도 Test Shell Application 없도록 함

```
1 + package com.samsung.handler;
2 +
3 + import com.samsung.command.CommandInvoker;
4 +
5 + import java.util.HashMap;
6 +
7 + public class CommandHandlerFactory {
8 +
9 +     public static CommandHandler
10 +     getCommandHandler(String[] args) {
11 +         CommandInvoker invoker = new
12 +         CommandInvoker(new HashMap<>());
13 +
14 +         if (hasArguments(args)) return new
15 +         FileCommandHandler(invoker);
16 +         return new
17 +         InteractiveCommandHandler(invoker);
18 +     }
19 +
20 +     private static boolean hasArguments(String[]
21 +     mainArgs) {
22 +         return mainArgs.length != 0;
23 +     }
24 + }
```



# 리팩토링 [3]

- 캡슐화 :  
객체의 내부 구현을 외부로부터 숨기고, 필요한 부분만 공개  
구현의 변경에 영향을 받지 않고 사용 (유지보수)

```
32     public void read(int index) {  
33         String head = "[Read] LBA";  
34         String location = String.format("%02d",  
35             index);  
35 -         String value;  
36 -  
37 -         fileManager.readFile(index);  
38 -         value =  
38 -         fileManager.getHashMap().getOrDefault(index,  
38 -         BLANK_DATA);  
39  
40 -         String output = head + " " + location + "  
40 -         " + value;  
41 -         System.out.println(output);  
42     }  
43
```



```
34     public void read(int index) {  
35         String head = "[Read] LBA";  
36         String location = String.format("%02d",  
37             index);  
37 +         String value =  
37 +         fileManager.getValueFromFile(index);  
38  
38  
39 +         System.out.println(head + " " + location  
39 +         + " " + value);  
40  
40     }  
41
```

# 리팩토링 [4]

- **모듈 분리** : 단일 책임 원칙(Single Responsibility Principle, SRP) 적용  
CommandInvoker가 여러 역할을 과도하게 수행-> 명령어 선언 및 할당 책임을 분리  
CommandRegistry를 새로 추가하여 책임을 명확히 나눔

```
10 public class CommandInvoker {
11
12     private final Map<String, Command> commandMap;
13
14     public CommandInvoker(Map<String, Command> commandMap) {
15         this.commandMap = commandMap;
16         initShellCommand();
17         initScriptCommand();
18     }
19
20     public void execute(String[] cmdArgs) {
21         Command command = getCommand(cmdArgs[0]);
22
23         command.execute(cmdArgs);
24     }
25
26     private Command getCommand(String commandName) {
27         Command command = commandMap.get(commandName);
28
29         if (command == null) {
30             throw new RuntimeException("INVALID COMMAND");
31         }
32
33         return command;
34     }
35
36     private void initShellCommand() {
37         TestShellManager testShellManager = new TestShellManager(new
38             JarExecutor(), FileManager.getInstance());
39         register("write", new WriteCommand(testShellManager));
40         register("read", new ReadCommand(testShellManager));
41         register("erase", new EraseCommand(testShellManager));
42         register("erase_range", new
43             EraseRangeCommand(testShellManager));
44         register("help", new HelpCommand(testShellManager));
45         register("fullwrite", new
46             FullWriteCommand(testShellManager));
47         register("fullread", new FullReadCommand(testShellManager));
48         register("flush", new FlushCommand(testShellManager));
49     }
50
51     private void register(String commandName, Command command) {
52         commandMap.put(commandName, command);
53     }
54
55     private void initScriptCommand() {
56         ScriptManager scriptManager = new
57             ScriptManager(FileManager.getInstance(), new JarExecutor(),
58                 RandomHex.getInstance());
59
60         String[] script1CommandNames = {"1_FullWriteAndReadCompare",
61             "1_"};
62         String[] script2CommandNames = {"2_PartialBAWrite", "2_"};
63         String[] script3CommandNames = {"3_WriteReadAging", "3_"};
64         String[] script4CommandNames = {"4_EraseAndWriteAging",
65             "4_"};
66
67         register(script1CommandNames, new
68             TestScript1Command(scriptManager));
69         register(script2CommandNames, new
70             TestScript2Command(scriptManager));
71         register(script3CommandNames, new
72             TestScript3Command(scriptManager));
73         register(script4CommandNames, new
74             TestScript4Command(scriptManager));
75     }
76 }
```


```
3 public class CommandInvoker {
4
5     private static final int COMMAND_NAME_INDEX = 0;
6
7     private final CommandRegistry registry;
8
9     public CommandInvoker(CommandRegistry registry) {
10         this.registry = registry;
11     }
12
13     public void execute(String[] cmdArgs) {
14         Command command =
15             registry.getCommand(cmdArgs[COMMAND_NAME_INDEX]);
16         validateCommandNotNull(command);
17
18         command.execute(cmdArgs);
19     }
20
21     private void validateCommandNotNull(Command command) {
22         if (command == null) {
23             throw new RuntimeException("INVALID COMMAND");
24         }
25     }
26 }
```

```
10 public class CommandRegistry {
11
12     private final Map<String, Command> commandMap;
13
14     public CommandRegistry(Map<String, Command>
15         commandMap) {
16         this.commandMap = commandMap;
17         initShellCommand();
18         initScriptCommand();
19     }
20
21     public Command getCommand(String name) {
22         return commandMap.get(name);
23     }
24
25     private void initShellCommand() {
26         TestShellManager testShellManager = new
27             TestShellManager(new JarExecutor(),
28                 FileManager.getInstance());
29         register("write", new
30             WriteCommand(testShellManager));
31         register("read", new
32             ReadCommand(testShellManager));
33         register("erase", new
34             EraseCommand(testShellManager));
35         register("erase_range", new
36             EraseRangeCommand(testShellManager));
37     }
38
39     private void initScriptCommand() {
40         ScriptManager scriptManager = new
41             ScriptManager(FileManager.getInstance(), new JarExecutor(),
42                 RandomHex.getInstance());
43
44         String[] script1CommandNames = {"1_FullWriteAndReadCompare",
45             "1_"};
46         String[] script2CommandNames = {"2_PartialBAWrite", "2_"};
47         String[] script3CommandNames = {"3_WriteReadAging", "3_"};
48         String[] script4CommandNames = {"4_EraseAndWriteAging",
49             "4_"};
50
51         register(script1CommandNames, new
52             TestScript1Command(scriptManager));
53         register(script2CommandNames, new
54             TestScript2Command(scriptManager));
55         register(script3CommandNames, new
56             TestScript3Command(scriptManager));
57         register(script4CommandNames, new
58             TestScript4Command(scriptManager));
59     }
60 }
```

# 리팩토링 [5]

- Enum 사용 - 단일 책임 원칙(Single Responsibility Principle, SRP) 적용  
CmdValidChecker는 ERASE가 들어와도 **변하지 않음**  
변경은 Enum CommandType에서 이루어지며, 확장이 용이함

```
6 public class CmdValidChecker {
7     public CmdData
8     cmdValidCheckAndParsing(String[] cmdParam) {
9         if (cmdParam.length == 0) {
10             return new
11             CmdData(SSDConstant.COMMAND_ERROR, -1,
12             SSDConstant.COMMAND_ERROR);
13         }
14         CommandValidator validator;
15         switch (cmdParam[0]) {
16             case SSDConstant.COMMAND_READ:
17                 validator = new ReadCommandValidator(); break;
18             case SSDConstant.COMMAND_WRITE:
19                 validator = new WriteCommandValidator(); break;
20             case SSDConstant.COMMAND_ERASE:
21                 validator = new EraseCommandValidator(); break;
22             case SSDConstant.COMMAND_FLUSH:
23                 validator = new FlushCommandValidator(); break;
24             default: validator = null;
25         }
26         if (validator == null) {
27             return new
28             CmdData(SSDConstant.COMMAND_ERROR, -1,
29             SSDConstant.COMMAND_ERROR);
30         }
31         return validator.validate(cmdParam);
32     }
33 }
```



```
8 public class CmdValidChecker {
9     public CmdData
10     cmdValidCheckAndParsing(String[] cmdParam) {
11         if (cmdParam.length == 0) {
12             return new CmdData(ERROR, -1,
13             ERROR.name());
14         }
15         CommandType cmdType = CommandType.fromCode(cmdParam[0]);
16         CommandValidator validator = cmdType.getCommandValidator();
17         return validator.validate(cmdParam);
18     }
19 }
```

```
@Getter
public enum CommandType {
    WRITE("W", new WriteCommandValidator()),
    ERASE("E", new EraseCommandValidator()),
    READ("R", new ReadCommandValidator()),
    FLUSH("F", new FlushCommandValidator()),
    ERROR("ERROR", null);

    private final String code;
    private final CommandValidator commandValidator;

    CommandType(String code, CommandValidator commandValidator) {
        this.code = code;
        this.commandValidator = commandValidator;
    }

    public static CommandType fromCode(String code) {
        return Arrays.stream(values())
            .filter(c -> c.code.equals(code))
            .findFirst()
            .orElseThrow(() -> new IllegalArgumentException("Unknown command: " + code));
    }
}
```

# 리팩토링 [6]

## ■ Smell - Long Parameter List

:함수 인자수를 줄임

```
public static void run(CmdData cmdData) {  
    SSDManager ssdManager = new SSDManager(cmdData.getCommand(), cmdData.getLba(), cmdData.getValue());  
    SSDManager ssdManager = new SSDManager(cmdData, FileManager.getInstance());  
    ssdManager.cmdExecute();  
}
```

## ■ Rename - 가독성 증가

```
~  
- log.info("test script3 테스트");  
41 + log.info("[Script3] LBA_FIRST & LBA_LAST 무작위 쓰기/검증 시작");  
42 for (int i = 0; i < LOOP_100 * 2; i++) {  
- boolean firstValue = writeAndVerify(LBA_FIRST, RandomHex.getInstance().getRandomValue());  
- boolean lastValue = writeAndVerify(LBA_LAST, RandomHex.getInstance().getRandomValue());  
- if (!isValidFirstLastValue(firstValue, lastValue)) return false;  
43 + boolean firstOk = writeAndVerify(LBA_FIRST, randomHex.getRandomValue());  
44 + boolean lastOk = writeAndVerify(LBA_LAST, randomHex.getRandomValue());
```

youngsik now

Owner ...

firstValue -> firstOk, lastValue -> lastOk

가독성 높일 수 있도록 의미있는 변수명으로 rename 리팩토링 . 좋습니다.!

# 리팩토링 [7]

- Invert-if - 부정 조건 보다는 **긍정** 조건 사용

yb

```
97 + private static boolean checkWriteValue(String[] cmdParam) {  
98 +     if (!cmdParam[0].equals("W")) return false;
```

youngsik now

Owner ...

부정인경우에 false가 나오는 부분이라 코드 이해가 쉽진 않았습니다.  
리팩토링시 한번 고려해주세요

# 소감

## ■ TDD (Test Driven Development)

: 먼저 Unit Test를 작성

익숙하지 않아 시간이 오래 걸리고 작성의 어려움이 있었습니다.

그러나!!!!

## ■ 버그 감소와 안정성 향상

기존 기능을 수정하더라도 테스트가 이를 체크해 주어 심리적으로 안정감을 줍니다.

*“수정 후 깨진 기능이 있는지를 즉시 확인”*

## ■ 리팩토링이 용이

리팩토링 전후에 테스트가 보장되므로 안심하고 리팩토링 할 수 있습니다.

*“깨지지 않을 것이라는 확신이 있으니 마음 편히 리팩토링을 진행”*



시연