HW5

1분반

32211792

박재홍

2024년 11월 13일

IProgramLauncherCommnad 인터페이스)

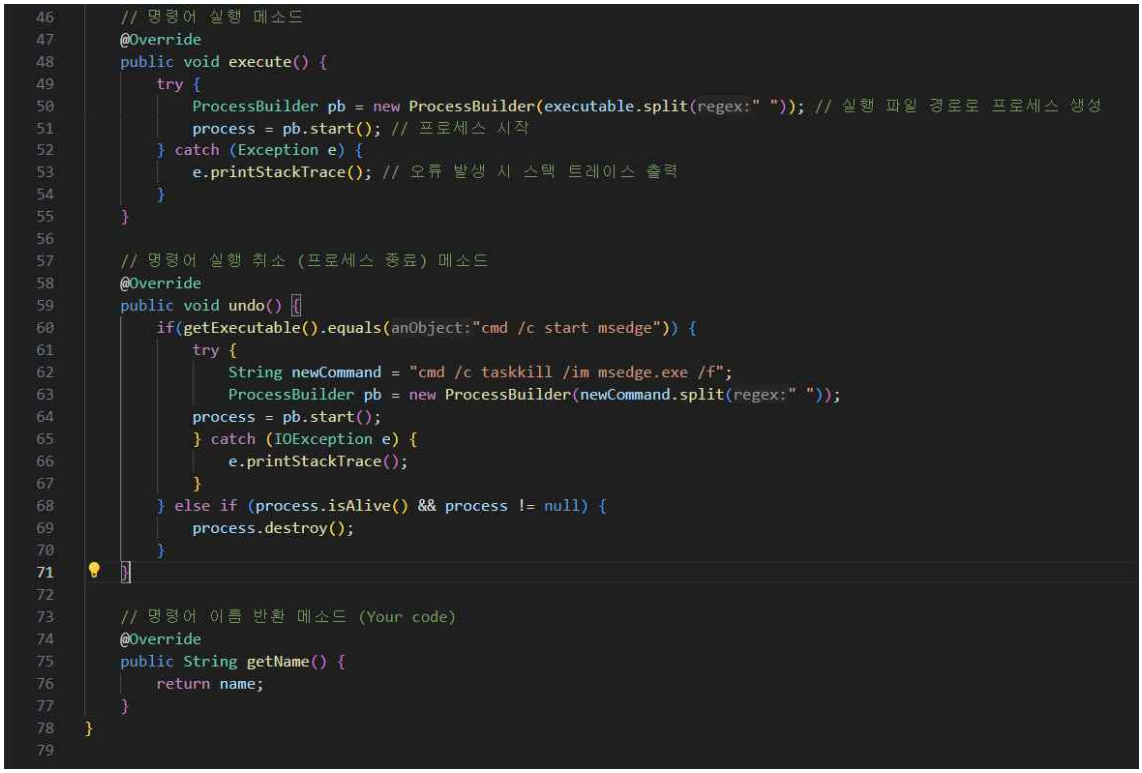
HW6 > src > IProgramLauncherCommand.java

```java
public interface IProgramLauncherCommand {
    void execute(); // 명령어 실행 메소드
    void undo(); // 명령어 실행 취소 메소드
    String getName(); // (Your Code)
}

```

IprogramLauncherCommand 인터페이스에는 명령어 실행 메소드인 execute와 명령어 취소 메소드인 undo 메소드를 선언해주었고 그다음 이름을 반환해주는 메소드인 getName을 선언해주었다.

ProGramLauncherCommand 클래스)

```java
public class ProgramLauncherCommand implements IProgramLauncherCommand {
    private String executable; // 실행 파일 경로를 나타내는 필드
    private String icon; // 아이콘 경로를 나타내는 필드
    private Process process; // 실행된 프로세스를 나타내는 필드
    private String name; // 명령어 이름을 나타내는 필드 (Your code)

    // getter 메소드: 실행 파일 경로 반환
    public String getExecutable() {
        return this.executable;
    }

    // setter 메소드: 실행 파일 경로 설정
    public void setExecutable(String executable) {
        this.executable = executable;
    }

    // getter 메소드: 아이콘 경로 반환
    public String getIcon() {
        return this.icon;
    }

    // setter 메소드: 아이콘 경로 설정
    public void setIcon(String icon) {
        this.icon = icon;
    }

    // getter 메소드: 프로세스 반환
    public Process getProcess() {
        return this.process;
    }

    // setter 메소드: 프로세스 설정
    public void setProcess(Process process) {
        this.process = process;
    }

    // 생성자: 실행 파일 경로, 아이콘 경로, 명령어 이름을 초기화
    public ProgramLauncherCommand(String executable, String icon, String name) {
        this.executable = executable;
        this.icon = icon;
        this.name = name;
    }
```



```java
    // 명령어 실행 메소드
    @Override
    public void execute() {
        try {
            ProcessBuilder pb = new ProcessBuilder(executable.split(regex:" ")); // 실행 파일 경로로 프로세스 생성
            process = pb.start(); // 프로세스 시작
        } catch (Exception e) {
            e.printStackTrace(); // 오류 발생 시 스택 트레이스 출력
        }
    }

    // 명령어 실행 취소 (프로세스 종료) 메소드
    @Override
    public void undo() {
        if(getExecutable().equals(anObject:"cmd /c start msedge")) {
            try {
                String newCommand = "cmd /c taskkill /im msedge.exe /f";
                ProcessBuilder pb = new ProcessBuilder(newCommand.split(regex:" "));
            process = pb.start();
            } catch (IOException e) {
                e.printStackTrace();
            }
        } else if (process.isAlive() && process != null) {
            process.destroy();
        }
    }

    // 명령어 이름 반환 메소드 (Your code)
    @Override
    public String getName() {
        return name;
    }
}

```

실행 파일 경로를 나타내는 필드, 아이콘 경로를 나타내는 필드, 실행된 프로세스를 나타내는 필드, 명령어 이름을 나타내는 필드를 선언해주었고 각 필드에 대한 getter 와 setter를 선언해주고 각 필드에 대한 생성자를 생성해 초기화 시켜주었다. 그 후 명령어 실행 메소드인 execute 메소드를 선언해 실행파일 경로로 프로세스가 생성되도록 해주었다. undo 메소드에서는 프로세스가 존재하는지 확인하고 undo 버튼이 눌렸을 때 프로세스를 종료하도록 선언했다.

ProGramLauncherCommandImporter 클래스)

HW6 > src > ProgramLauncherCommandImporter.java

```java
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.FileReader;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class ProgramLauncherCommandImporter {

    // JSON 파일에서 명령어 목록을 읽어들여, 각 명령어를 ProgramLauncherCommand 객체로 생성하여 맵에 저장합니다.

    public static Map<String, ProgramLauncherCommand> loadCommandsFromJson(String filename) {
        Map<String, ProgramLauncherCommand> commandsMap = new HashMap<>();
        try {
            JSONParser parser = new JSONParser(); // JSON 파서를 생성
            JSONObject jsonObject = (JSONObject) parser.parse(new FileReader(filename)); // 파일을 읽고 JSON 객체로 변환

            // JSON 객체에서 "commands" 배열을 가져옴
            JSONArray commandsArray = (JSONArray) jsonObject.get("commands");

            // 배열에 있는 각 명령어 정보를 반복하여 처리
            for (Object obj : commandsArray) {
                JSONObject commandData = (JSONObject) obj; // 각 명령어 정보를 JSON 객체로 변환
                String name = (String) commandData.get("name"); // 명령어 이름을 가져옴
                String executable = (String) commandData.get("executable"); // 실행 파일 경로를 가져옴
                String icon = (String) commandData.get("icon"); // 아이콘 경로를 가져옴

                // ProgramLauncherCommand 객체 생성 (name, executable, icon 필드 사용)
                ProgramLauncherCommand command = new ProgramLauncherCommand(executable, icon, name);
                commandsMap.put(name, command); // 명령어 이름을 키로, 객체를 값으로 맵에 저장
            }
        } catch (IOException | ParseException e) {
            e.printStackTrace(); // 파일 읽기 또는 파싱 오류 발생 시 스택 트레이스 출력
        }
        return commandsMap; // 생성된 명령어 맵 반환
    }
}
```

ProGramLauncherCommandImporter 클래스에서는 JSON 파일에서 명령어 목록을 읽어들여, 각 명령어를 ProgramLauncherCommand 객체로 생성하여 Map에 저장 시키도록 해주었다. try-catch 문을 통해 JSONParser를 생성하고 파일을 읽고 JSON 객체로 변환 시켜주었고 JSON 객체에서 commands 배열을 가져왔다. 그 후 배열에 있는 각 명령어 정보를 반복하여 처리해주었고 각 명령어 정보를 JSON 객체로 변환시켜주고 명령어 이름을 가져오고 실행파일 경로를 가져오고 아이콘 경로를 가져오도록 해주었다. 그 다음 ProgramLauncher 객체를 생성했고 명령어 이름을 키로 , 객체를 값으로 맵에 저장해주었다.

ProGramLauncherCommandInvoker 클래스)

HW6 > src > ProgramLauncherCommandInvoker.java

```java
import java.util.List;
import java.util.Stack;
import java.util.stream.Collectors;

public class ProgramLauncherCommandInvoker {
    private IProgramLauncherCommand command; // 현재 실행할 명령어를 나타내는 필드
    private Stack<IProgramLauncherCommand> commandStack = new Stack<>(); // 실행한 명령어들을 저장하는 스택

    // 새로운 명령어를 설정하는 메소드
    public void setCommand(IProgramLauncherCommand c){
        this.command = c;
    }

    // 설정된 명령어를 실행하고, 스택에 저장하는 메소드
    public void executeCommand(){
        if(command != null){ // 명령어가 설정되어 있는지 확인
            command.execute(); // 명령어 실행
            commandStack.push(command); // 실행한 명령어를 스택에 저장
        }
    }

    // 마지막으로 실행한 명령어를 되돌리는 메소드
    public void undoLastCommand() {
        if (!commandStack.isEmpty()){ // 스택이 비어있지 않은지 확인
            IProgramLauncherCommand lastCommand = commandStack.pop(); // 스택에서 마지막 명령어를 가져옴
            lastCommand.undo(); // 마지막 명령어의 undo 메소드 실행
        }
    }

    // 최근에 실행된 명령어의 이름 목록을 반환하는 메소드 (Your Code)
    public List<String> getRecentlyExecutedCommands() {
        return commandStack.stream() // 스택의 명령어들을 스트림으로 변환
                           .map(IProgramLauncherCommand::getName) // 각 명령어의 이름을 가져옴
                           .collect(Collectors.toList()); // 이름 목록을 리스트로 수집
    }
}
```

ProGramLauncherCommandInvoker 클래스 에서는 현재 실행할 명령어를 나타내는 필드를 선언해주고 실행한 명령어들을 저장하는 스택을 선언해주었다. 그 후 새로운 명령어를 설정하는 setCommand 메소드를 선언했고 설정된 명령어를 실행하고 스택에 저장하는 executeCommand 메소드를 선언해 명령어가 설정되어 있는지 확인하고 명령어를 실행하고 실행한 명령어를 스택에 저장하도록 해주었다. 그 후 마지막으로 실행한 명령어를 되돌리는 undoLastCommand 메소드를 선언해 스택이 비어있지 않은지 확인 하고 스택에서 마지막 명령어를 가져와 마지막 명령어의 undo 메소드를 실행시켜주었다.

ProGramLauncherCommandApp 클래스)

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Map;
import java.util.List;

public class ProgramLauncherCommandApp extends JFrame {

    private ProgramLauncherCommandInvoker launcher = new ProgramLauncherCommandInvoker(); // 명령어 실행 및 되돌리기를 담당하는 invoker 객체
    private Map<String, ProgramLauncherCommand> commandsMap; // 명령어 이름과 ProgramLauncherCommand 객체를 매핑한 맵

    public ProgramLauncherCommandApp() {
        setTitle("Program Launcher with Icons"); // 프로그램 타이틀 설정
        setSize(600, 600); // 창 크기 설정
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // 창 닫기 시 프로그램 종료
        setLayout(new GridLayout(0, 1)); // 레이아웃을 한 열로 설정 (버튼이 세로로 추가됨)

        // 최근 실행 명령어 조회 버튼 추가 (Your Code)
        JButton recentCommandsButton = new JButton("최근 실행 명령어 조회");
        recentCommandsButton.addActionListener(e -> showRecentCommands());
        add(recentCommandsButton);

        // JSON 파일에서 명령어를 로드하고 commandsMap에 저장
        commandsMap = ProgramLauncherCommandImporter.loadCommandsFromJson("commands.json");

        // 각 명령어에 대해 버튼을 생성하고 GUI에 추가
        for (Map.Entry<String, ProgramLauncherCommand> entry : commandsMap.entrySet()) {
            JButton button = createButtonWithIcon(entry.getKey(), entry.getValue()); // 명령어 버튼 생성
            add(button); // 버튼을 GUI에 추가
        }

        // Undo 버튼 추가 (마지막 명령어를 되돌리기)
        JButton undoButton = new JButton("Undo Last Command");
        undoButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                launcher.undoLastCommand(); // 마지막 명령어 되돌리기
            }
        });
        add(undoButton);

        setVisible(true); // 창을 화면에 보이게 설정
    }

    // 최근에 실행된 명령어 목록을 가져와서 JOptionPane으로 출력 (Your Code)
    private void showRecentCommands() {
        List<String> recentCommands = launcher.getRecentlyExecutedCommands(); // 최근 명령어 목록 가져오기
        String message = recentCommands.isEmpty()
                ? "최근 실행된 명령어가 없습니다."
                : String.join("\n", recentCommands);
        JOptionPane.showMessageDialog(this, message, "최근 실행 명령어", JOptionPane.INFORMATION_MESSAGE); // 메시지 박스로 출력
    }

    // 명령어 이름과 아이콘을 사용해 버튼을 생성
    private JButton createButtonWithIcon(String name, ProgramLauncherCommand command) {
        JButton button = new JButton(name); // 명령어 이름으로 버튼 생성
        button.setIcon(new ImageIcon(command.getIcon())); // 아이콘 설정
        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                launcher.setCommand(command); // invoker에 명령어 설정
                launcher.executeCommand(); // 명령어 실행
            }
        });
        return button; // 생성된 버튼 반환
    }
}

```

ProGramLauncherCommandApp 클래스에서는 우선적으로 JFrame을 상속받고 명령어 실행 및 되돌리기를 담당하는 invoker 객체를 생성해주었고 명령어 이름과 Programlaunchercommand 객체를 매핑한 맵을 선언했고

ProGramLauncherCommandApp 메소드를 선언해 프로그램 타이틀을 설정하고 창 크기를 600,600 으로 설정해주고 창을 닫을 때 프로그램이 종료되도록 해주었고 레이아웃을 한 열로 설정해주었다. 그 후 JSON파일에서 명령어를 로드하고 commandMap에 저장해주었고 각 명령어에 대한 버튼을 생성하고 GUI에 추가했다. 그리고 Undo 버튼을 추가해 마지막 명령어를 되돌릴 수 있게 해주었고 그 다음 명령어 이름과 아이콘을 사용해 버튼을 생성해주었다.

Your Code)

HW6 > src > IProgramLauncherCommand.java

```java
public interface IProgramLauncherCommand {
    void execute(); // 명령어 실행 메소드
    void undo(); // 명령어 실행 취소 메소드
    String getName(); // (Your Code)
}
```

```java
public class ProgramLauncherCommand implements IProgramLauncherCommand {
    private String executable; // 실행 파일 경로를 나타내는 필드
    private String icon; // 아이콘 경로를 나타내는 필드
    private Process process; // 실행된 프로세스를 나타내는 필드
    private String name; // 명령어 이름을 나타내는 필드 (Your code)

    // 명령어 이름 반환 메소드 (Your code)
    @Override
    public String getName() {
        return name;
    }
}
```

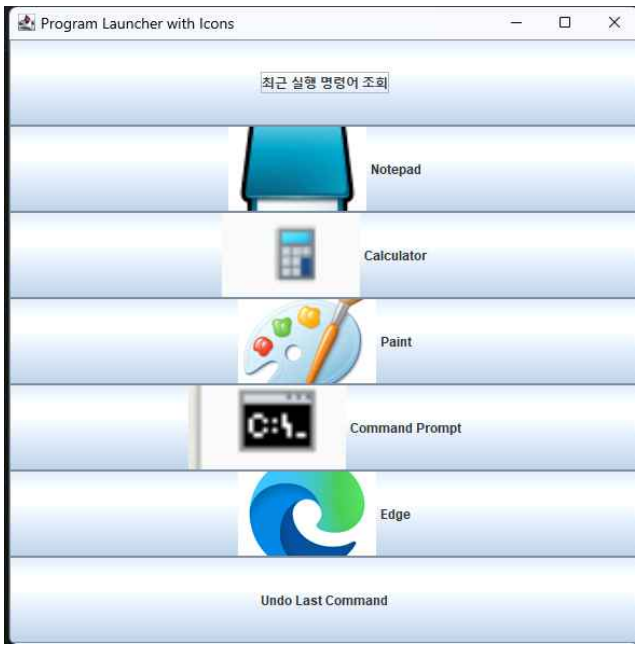
```java
// 최근에 실행된 명령어의 이름 목록을 반환하는 메소드 (Your Code)
public List<String> getRecentlyExecutedCommands() {
    return commandStack.stream() // 스택의 명령어들을 스트림으로 변환
                       .map(IProgramLauncherCommand::getName) // 각 명령어의 이름을 가져옴
                       .collect(Collectors.toList()); // 이름 목록을 리스트로 수집
}
}
```

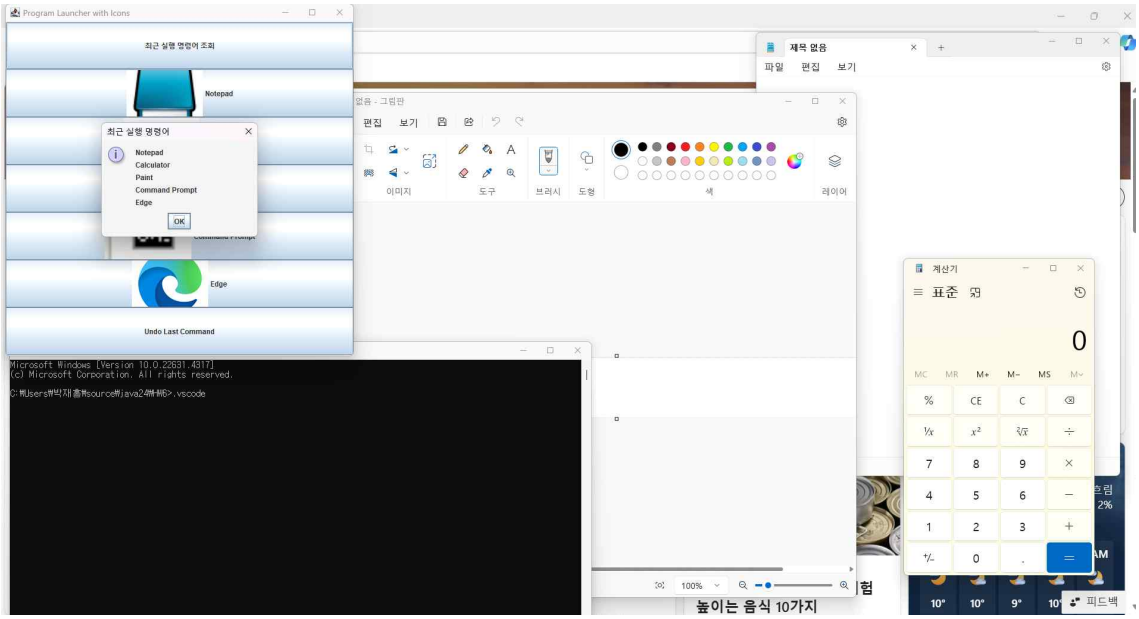
```java
// 최근 실행 명령어 조회 버튼 추가 (Your Code)
JButton recentCommandsButton = new JButton("최근 실행 명령어 조회");
recentCommandsButton.addActionListener(e -> showRecentCommands());
add(recentCommandsButton);
```

```java
// 최근에 실행된 명령어 목록을 가져와서 JOptionPane으로 출력 (Your Code)
private void showRecentCommands() {
    List<String> recentCommands = launcher.getRecentlyExecutedCommands(); // 최근 명령어 목록 가져오기
    String message = recentCommands.isEmpty()
            ? "최근 실행된 명령어가 없습니다."
            : String.join("\n", recentCommands);
    JOptionPane.showMessageDialog(this, message, "최근 실행 명령어", JOptionPane.INFORMATION_MESSAGE); // 메시지 박스로 출력
}
```

your code 로는 최근에 실행했던 명령어들을 출력해주는 것을 만들어주었다. 최근에 실행했던 명령어들 중 undo 버튼을 누르지 않은 명령어들에 대해 목록을 리스트로 수집 후 출력하게 해주었다. 추가적으로 command.json에 cmd 와 caculator를 추가적을 넣어주었다.

실행결과)


Program Launcher with Icons
최근 실행 명령어 조회
Notepad
Calculator
Paint
Command Prompt
Edge
Undo Last Command


Program Launcher with Icons
최근 실행 명령어 조회
Notepad
최근 실행 명령어
Notepad
Calculator
Paint
Command Prompt
Edge
OK
Edge
Undo Last Command
Microsoft Windows [Version 10.0.22631.4317]
(c) Microsoft Corporation. All rights reserved.
제목 없음
파일 편집 보기
계산기
표준
0
높이는 음식 10가지
피드백