HW7

1분반

2024년 11월 20일

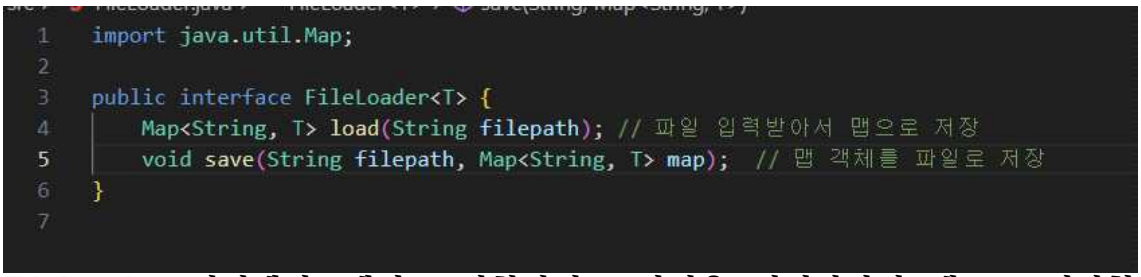
32211792

박재홍

FileImporter 인터페이스)

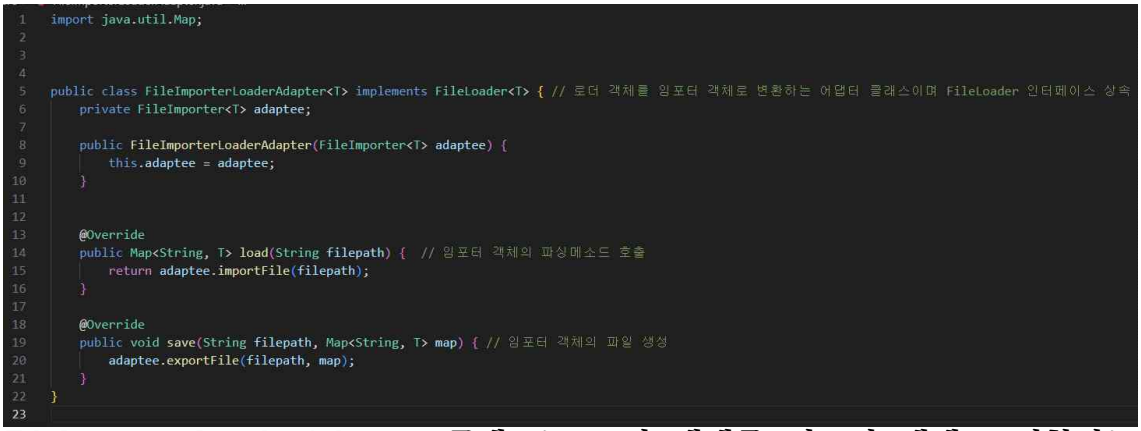
FileImporter 인터페이스에서 파일을 입력받아서 맵으로 저장할 수 있게 했고 맵 객체를 파일로 저장 시켰다.

FileLoader 인터페이스)



FileLoader 인터페이스에서도 마찬가지고 파일을 입력받아서 맵으로 저장할 수 있게 했고 맵 객체를 파일로 저장시켜주었다.

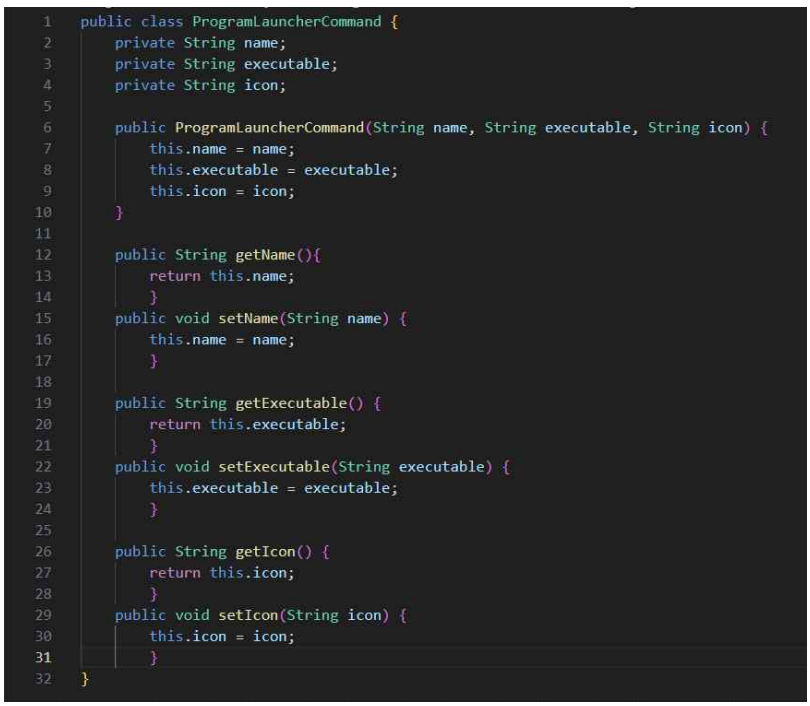
FileImporterLoaderAdapter 클래스)



```java
import java.util.Map;



public class FileImporterLoaderAdapter<T> implements FileLoader<T> { // 로더 객체를 임포터 객체로 변환하는 어댑터 클래스이며 FileLoader 인터페이스 상속
    private FileImporter<T> adaptee;

    public FileImporterLoaderAdapter(FileImporter<T> adaptee) {
        this.adaptee = adaptee;
    }


    @Override
    public Map<String, T> load(String filepath) {  // 임포터 객체의 파싱메소드 호출
        return adaptee.importFile(filepath);
    }

    @Override
    public void save(String filepath, Map<String, T> map) { // 임포터 객체의 파일 생성
        adaptee.exportFile(filepath, map);
    }
}

```

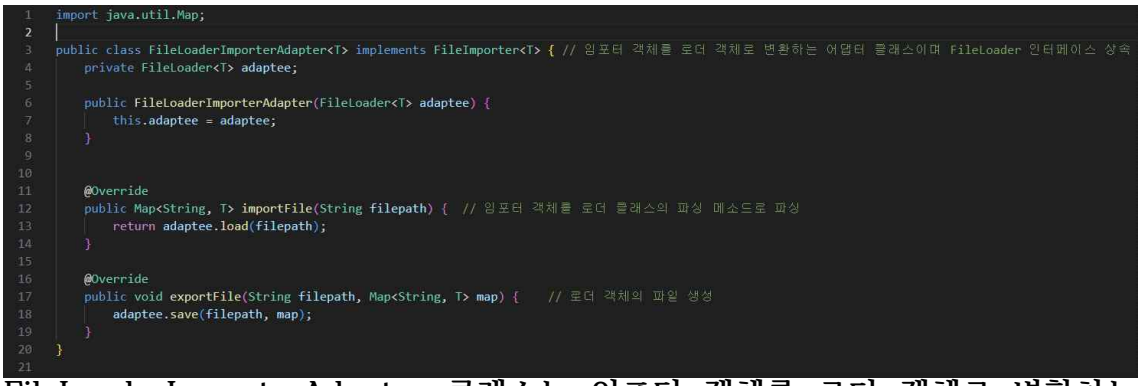
FileImporterLoaderAdapter 클래스는 로더 객체를 임포터 객체로 변환하는 어댑터 클라스이며 FileLoader 인터페이스를 상속했다. 그 후 load 메소드에서 임포터 객체의 파싱메소드를 호출했고 save 메소드에서 임포터 객체의 파일을 생성해주었다.

ProgramLauncherCommand 클래스)



```java
public class ProgramLauncherCommand {
    private String name;
    private String executable;
    private String icon;

    public ProgramLauncherCommand(String name, String executable, String icon) {
        this.name = name;
        this.executable = executable;
        this.icon = icon;
    }

    public String getName(){
        return this.name;
        }
    public void setName(String name) {
        this.name = name;
        }

    public String getExecutable() {
        return this.executable;
        }
    public void setExecutable(String executable) {
        this.executable = executable;
        }

    public String getIcon() {
        return this.icon;
        }
    public void setIcon(String icon) {
        this.icon = icon;
        }
}
```

FileLoaderImporterAdapter 클래스)



```java
import java.util.Map;

public class FileLoaderImporterAdapter<T> implements FileImporter<T> { // 임포터 객체를 로더 객체로 변환하는 어댑터 클래스이며 FileLoader 인터페이스 상속
    private FileLoader<T> adaptee;

    public FileLoaderImporterAdapter(FileLoader<T> adaptee) {
        this.adaptee = adaptee;
    }


    @Override
    public Map<String, T> importFile(String filepath) {  // 임포터 객체를 로더 클래스의 파싱 메소드로 파싱
        return adaptee.load(filepath);
    }

    @Override
    public void exportFile(String filepath, Map<String, T> map) {    // 로더 객체의 파일 생성
        adaptee.save(filepath, map);
    }
}

```

FileLoaderImporterAdapter 클래스는 임포터 객체를 로더 객체로 변환하는 어댑터 클래스이며 FileLoader인터페이스를 상속했다. 그리고 importFile 메소드를 통해 임포터 객체를 로더 클래스의 파싱 메소드로 파싱했고, exportFile메소드를 통해 로더 객체의 파일을 생성해주었다.

ProgramLauncherCommandJSONFileImporter 클래스)

```java
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.lang.reflect.Type;
import java.util.Map;
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.google.gson.reflect.TypeToken;

public class ProgramLauncherCommandJSONFileImporter implements FileImporter<ProgramLauncherCommand> { // json 파일 로드하고 파일로 생성하는 클래스
    private Gson gson = new GsonBuilder().setPrettyPrinting().create(); // Gson 사용해서 파싱과정 간소화

    @Override
    public Map<String, ProgramLauncherCommand> importFile(String filepath) {  // JSON 파일 읽어서 맵 객체로 반환
        try (FileReader reader = new FileReader(filepath)) {
            Type type = new TypeToken<Map<String, ProgramLauncherCommand>>() {}.getType();  // 제네릭 사용으로 인해 런타임에 객체의 타입을 알 수 없음
            return gson.fromJson(reader, type);  // 타입 토큰을 사용해서 명시한 제네릭 타입을 사용해서 맵 객체로 반환
        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }

    @Override
    public void exportFile(String filePath, Map<String, ProgramLauncherCommand> map) {    // 맵 객체를 JSON 파일로 저장
        try (FileWriter writer = new FileWriter(filePath)) {
            gson.toJson(map, writer);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ProgramLauncherCommandJSONFileImporter 클래스는 Json 파일을 로드하고 파일로 생성한다. Gson을 사용해서 피싱과정을 간소화 하였고 importFile 메소드를 통해 Json파일 읽어서 맵 객체로 반환 해주었고 try-catch문을 통해 타입토큰을 사용해서 명시한 제네릭 타입을 사용해서 맵 객체로 반환시켜주었다. exportFile 메소드를 통해 맵 객체를 JSON파일로 저장해주었다.

ProgramLauncherCommandXMLFileImporter 클래스)

```java
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;



public class ProgramLauncherCommandXMLFileImporter implements FileImporter<ProgramLauncherCommand> {
    // XML 파일을 읽고(Map 객체로 변환), Map 데이터를 XML 형식으로 저장하는 클래스

    @Override
    public Map<String, ProgramLauncherCommand> importFile(String filePath) { // XML 파일을 읽어 Map<String, ProgramLauncherCommand>로 변환
        Map<String, ProgramLauncherCommand> commands = new HashMap<>();      // 결과 데이터를 저장할 Map 객체
        try {
            File xmlFile = new File(filePath); // XML 파일 객체 생성
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance(); // DocumentBuilderFactory와 Builder를 통해 XML 문서 파싱 준비
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document document = builder.parse(xmlFile); // XML 문서 파싱
            NodeList commandList = document.getElementsByTagName(tagname:"command"); // "command" 태그를 기준으로 XML 데이터를 추출

            for (int i = 0; i < commandList.getLength(); i++) { // 각 "command" 태그를 반복하면서 데이터 추출
                Element commandElement = (Element) commandList.item(i);

                // 태그 내 데이터 추출
                String name = commandElement.getElementsByTagName(name:"name").item(index:0).getTextContent();
                String executable = commandElement.getElementsByTagName(name:"executable").item(index:0).getTextContent();
                String icon = commandElement.getElementsByTagName(name:"icon").item(index:0).getTextContent();
                // ProgramLauncherCommand 객체 생성 및 Map에 추가
                ProgramLauncherCommand command = new ProgramLauncherCommand(name, executable, icon);
                commands.put(name, command);
            }
        } catch (ParserConfigurationException | SAXException | IOException e) {
            e.printStackTrace(); // 예외 발생 시 스택 트레이스 출력
        }

        return commands; // 변환된 Map 반환
    }

    @Override
    public void exportFile(String filePath, Map<String, ProgramLauncherCommand> map) { // Map<String, ProgramLauncherCommand> 데이터를 XML 파일로 저장
        try {
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance(); // XML 문서를 생성하기 위한 DocumentBuilder
            DocumentBuilder builder = factory.newDocumentBuilder();
            Document document = builder.newDocument();

            Element root = document.createElement(tagName:"commands"); // 루트 태그 "commands" 생성
            document.appendChild(root);

            for (Map.Entry<String, ProgramLauncherCommand> entry : map.entrySet()) { // Map 데이터를 XML 형식으로 변환
                ProgramLauncherCommand command = entry.getValue();

                Element commandElement = document.createElement(tagName:"command"); // "command" 태그 생성

                Element name = document.createElement(tagName:"name"); // "command" 태그 생성
                name.setTextContent(command.getName());

                Element executable = document.createElement(tagName:"executable"); // "executable" 태그 생성 및 데이터 추가
                executable.setTextContent(command.getExecutable());

                Element icon = document.createElement(tagName:"icon"); // "icon" 태그 생성 및 데이터 추가
                icon.setTextContent(command.getIcon());
                    // "command" 태그에 "name", "executable", "icon" 추가
                    commandElement.appendChild(name);
                    commandElement.appendChild(executable);
                    commandElement.appendChild(icon);
                    root.appendChild(commandElement); // 루트 태그에 "command" 추가
                }

                // Transformer 설정: 출력 결과 포매팅 적용
                TransformerFactory transformerFactory = TransformerFactory.newInstance();
                Transformer transformer = transformerFactory.newTransformer();
                transformer.setOutputProperty(javax.xml.transform.OutputKeys.INDENT, value:"yes"); // 들여쓰기 적용
                transformer.setOutputProperty(name:"{http://xml.apache.org/xslt}indent-amount", value:"4"); // 들여쓰기 공백 설정

                // XML 데이터를 파일로 변환하여 저장
                DOMSource source = new DOMSource(document);
                StreamResult result = new StreamResult(new FileWriter(filePath));
                transformer.transform(source, result);
            } catch (Exception e) {
                e.printStackTrace(); // 예외 발생 시 스택 트레이스 출력
            }
    }
}

```

ProgramLauncherCommandXMLFileImporter 클래스는 XML 파일을 읽고 Map 데이터를 XML형식으로 저장하는 클래스이다. importFile 메소드에서 XML 파일을 읽어 Map<String, ProgramLauncherCommand> 로 변환 시켜주었고 결과 데이터를 저장할 Map 객체를 선언해주었다. 그 후 try-catch 문을 통해 XML 파일 객체를 생성해주었고 DocumentBuilderFactory 와 Builder를 통해 XML 문서 파싱 준비를 해주었다. 그 후 XML 문서를 파싱해 주었고 'command' 태그를 기준으로 XML 데이터를 추출해주었다. 그리고 for문을 통해 각 'command' 태그를 반복하면서 XML데이터를 추출해주었다. 그다음, exportFile 메소드를 통해 Map<String, ProgramLauncherCommand> 데이터를 XML 파일로 저장해주었다. XML문서를 생성하기 위한 DocumentBuilder를 선언해주고 루트 태그 command를 생성 해주고 for문을 통해 Map데이터를 XML 형식으로 변환해주고 'commnad' 태그를 생성해주었다. 그 후 'name', 'executable', 'icon' 태그 생성 및 데이터를 추가해주었다. 마지막에 XML데이터를 파일로 변환하여 저장할 수 있도록 해주었다.

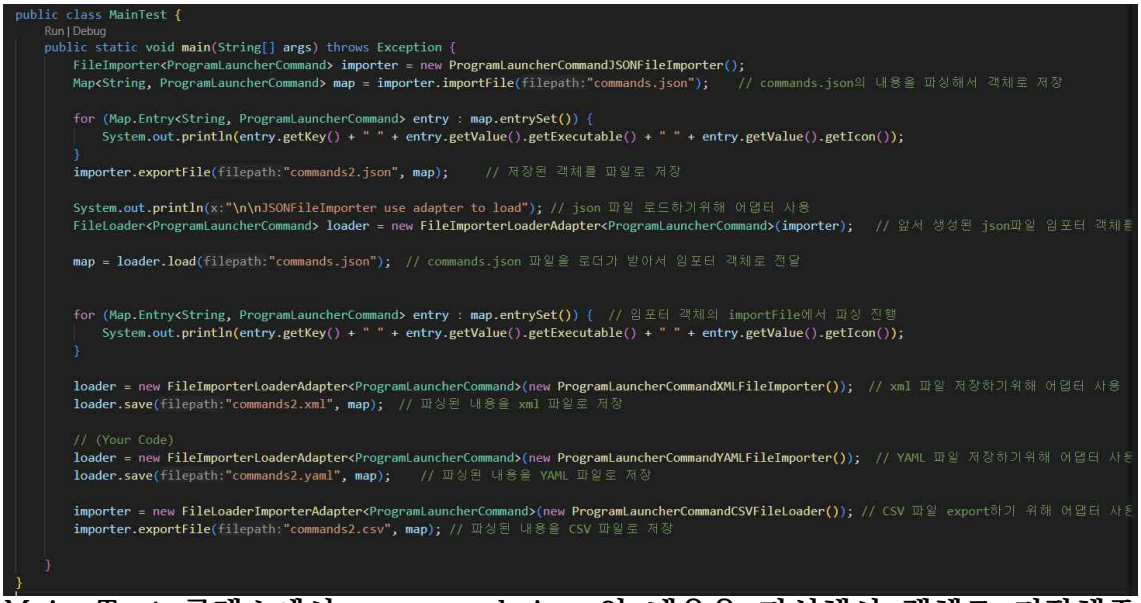
ProgramLauncherCommandCSVFileLoader 클래스)

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class ProgramLauncherCommandCSVFileLoader implements FileLoader<ProgramLauncherCommand> {   // CSV파일 읽고, 저장하는 클래스

    @Override
    public Map<String, ProgramLauncherCommand> load(String filepath) {  // 파일 읽어서 맵 객체로 반환
        Map<String, ProgramLauncherCommand> commands = new HashMap<>();

        try (BufferedReader reader = new BufferedReader(new FileReader(filepath))) {
            String line = reader.readLine();

            while ((line = reader.readLine()) != null) {   // 각 줄을 읽어와서 객체로 변환
                String[] data = line.split(regex:",");  // 각 줄을 콤마 단위로 파싱해서 맵으로 저장
                if (data.length == 3) {
                    String name = data[0];
                    String executable = data[1];
                    String icon = data[2];

                    ProgramLauncherCommand command = new ProgramLauncherCommand(name, executable, icon);    // ProgramLauncherCommand 객체 생성 후 Map에
                    commands.put(name, command);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }

        return commands;
    }
```

```java
    @Override
    public void save(String filepath, Map<String, ProgramLauncherCommand> map) {  // 맵 객체를 포매팅해서 파일로 저장
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(filepath))) {  // Map의 데이터를 CSV 형식으로 저장
            for (Map.Entry<String, ProgramLauncherCommand> entry : map.entrySet()) {
                ProgramLauncherCommand command = entry.getValue();
                String line = String.format(format:"%s,%s,%s", command.getName(), command.getExecutable(), command.getIcon());
                writer.write(line);
                writer.newLine();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ProgramLauncherCommandCSVFileLoader 클래스는 csv 파일을 읽고 저장하는 클래스이다. load 메소드에서 파일을 읽어서 맵 객체로 반환할 수 있도록 해주었다. 중간에 while 문을 통해 각 줄을 읽어와서 객체로 변환시켜주었고 각 줄을 콤마 단위로 파싱해서 맵으로 저장할 수 있게 해주었다. 그 후 programlauncherCommand 객체 생성후 map에 저장할 수 있게 해주었다. 그 후 save 메소드에서 맵 객체를 포매팅해서 파일로 저장하고 map의 데이터를 csv 형식으로 저장할 수 있게 했다.

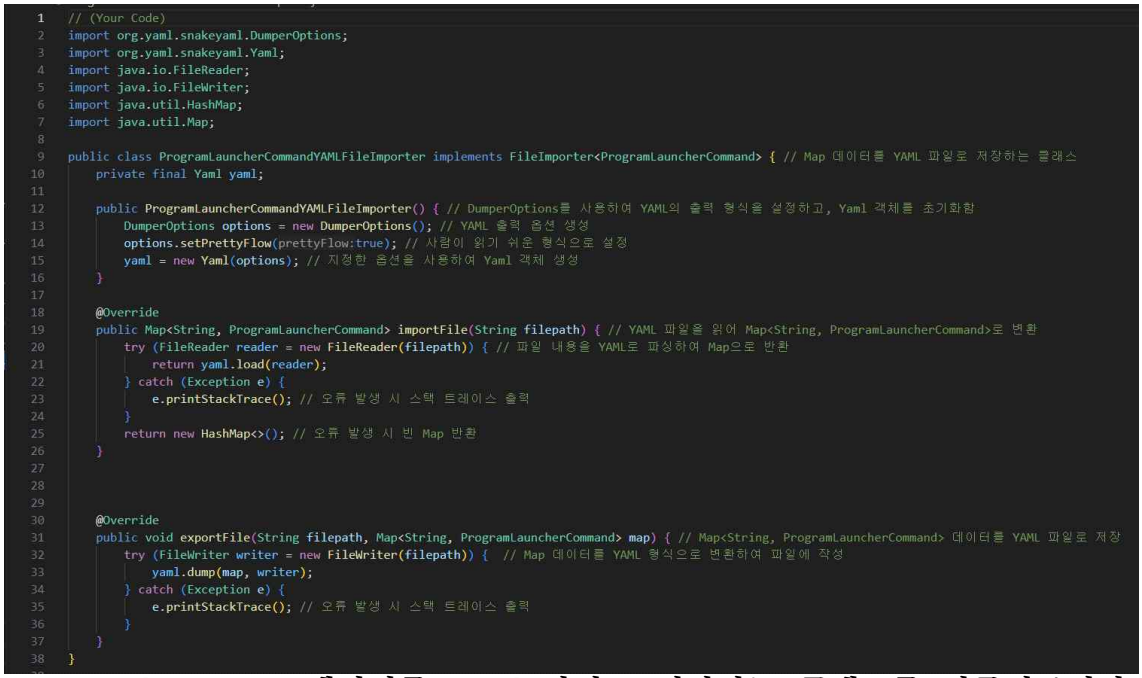
Main Test 클래스)



```
public class MainTest {
    Run | Debug
    public static void main(String[] args) throws Exception {
        FileImporter<ProgramLauncherCommand> importer = new ProgramLauncherCommandJSONFileImporter();
        Map<String, ProgramLauncherCommand> map = importer.importFile(filepath:"commands.json");    // commands.json의 내용을 파싱해서 객체로 저장

        for (Map.Entry<String, ProgramLauncherCommand> entry : map.entrySet()) {
            System.out.println(entry.getKey() + " " + entry.getValue().getExecutable() + " " + entry.getValue().getIcon());
        }
        importer.exportFile(filepath:"commands2.json", map);     // 저장된 객체를 파일로 저장

        System.out.println(x:"\n\nJSONFileImporter use adapter to load"); // json 파일 로드하기위해 어댑터 사용
        FileLoader<ProgramLauncherCommand> loader = new FileImporterLoaderAdapter<ProgramLauncherCommand>(importer);   // 앞서 생성된 json파일 임포터 객체를

        map = loader.load(filepath:"commands.json");  // commands.json 파일을 로더가 받아서 임포터 객체로 전달


        for (Map.Entry<String, ProgramLauncherCommand> entry : map.entrySet()) {  // 임포터 객체의 importFile에서 파싱 진행
            System.out.println(entry.getKey() + " " + entry.getValue().getExecutable() + " " + entry.getValue().getIcon());
        }

        loader = new FileImporterLoaderAdapter<ProgramLauncherCommand>(new ProgramLauncherCommandXMLFileImporter());  // xml 파일 저장하기위해 어댑터 사용
        loader.save(filepath:"commands2.xml", map);  // 파싱된 내용을 xml 파일로 저장

        // (Your Code)
        loader = new FileImporterLoaderAdapter<ProgramLauncherCommand>(new ProgramLauncherCommandYAMLFileImporter());  // YAML 파일 저장하기위해 어댑터 사
        loader.save(filepath:"commands2.yaml", map);    // 파싱된 내용을 YAML 파일로 저장

        importer = new FileLoaderImporterAdapter<ProgramLauncherCommand>(new ProgramLauncherCommandCSVFileLoader()); // CSV 파일 export하기 위해 어댑터 사
        importer.exportFile(filepath:"commands2.csv", map); // 파싱된 내용을 CSV 파일로 저장

    }
}
```

Main Test 클래스에서 commands.json의 내용을 파싱해서 객체로 저장해주었고 저장된 객체를 commands2.json라는 파일로 저장할 수 있게했다. 그 후 json 파일을 로드하기위해 어댑터를 사용했고 앞서 생성된 json파일을 임포터 객체로 전달해주었다. 그리고 commands.json 파일을 로더가 받아서 임포터 객체로 전달하게 해주었다. 그 후 임포터 객체의 importFile에서 파싱을 진행했고 xml 파일을 저장하기 위해 어댑터를 사용했고 파싱된 내용을 xml 파일로 저장해주었다. 동일하게 YAML 파일, CSV파일도 동일하게 진행했다.

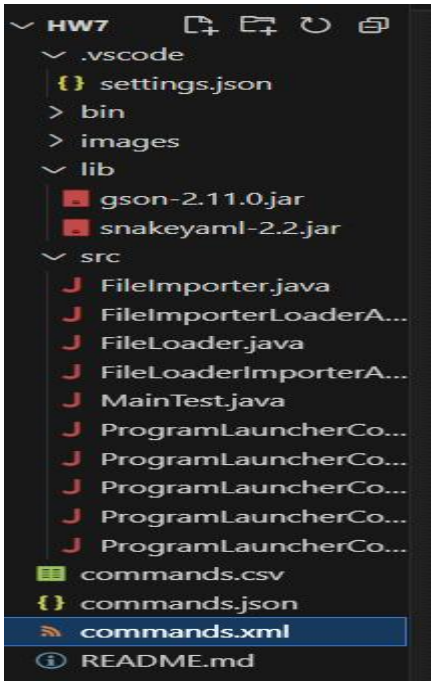
Your Code)

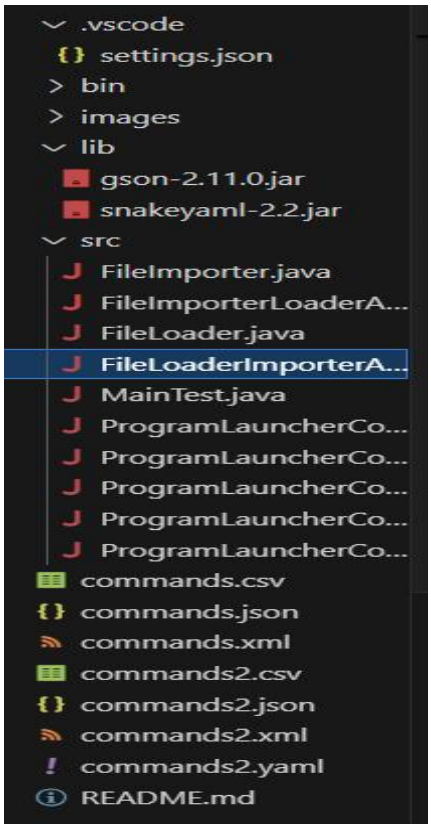


```java
// (Your Code)
import org.yaml.snakeyaml.DumperOptions;
import org.yaml.snakeyaml.Yaml;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.HashMap;
import java.util.Map;

public class ProgramLauncherCommandYAMLFileImporter implements FileImporter<ProgramLauncherCommand> { // Map 데이터를 YAML 파일로 저장하는 클래스
    private final Yaml yaml;

    public ProgramLauncherCommandYAMLFileImporter() { // DumperOptions를 사용하여 YAML의 출력 형식을 설정하고, Yaml 객체를 초기화함
        DumperOptions options = new DumperOptions(); // YAML 출력 옵션 생성
        options.setPrettyFlow(prettyFlow:true); // 사람이 읽기 쉬운 형식으로 설정
        yaml = new Yaml(options); // 지정한 옵션을 사용하여 Yaml 객체 생성
    }

    @Override
    public Map<String, ProgramLauncherCommand> importFile(String filepath) { // YAML 파일을 읽어 Map<String, ProgramLauncherCommand>로 변환
        try (FileReader reader = new FileReader(filepath)) { // 파일 내용을 YAML로 파싱하여 Map으로 반환
            return yaml.load(reader);
        } catch (Exception e) {
            e.printStackTrace(); // 오류 발생 시 스택 트레이스 출력
        }
        return new HashMap<>(); // 오류 발생 시 빈 Map 반환
    }



    @Override
    public void exportFile(String filepath, Map<String, ProgramLauncherCommand> map) { // Map<String, ProgramLauncherCommand> 데이터를 YAML 파일로 저장
        try (FileWriter writer = new FileWriter(filepath)) {  // Map 데이터를 YAML 형식으로 변환하여 파일에 작성
            yaml.dump(map, writer);
        } catch (Exception e) {
            e.printStackTrace(); // 오류 발생 시 스택 트레이스 출력
        }
    }
}
```

your code로 map 데이터를 yaml 파일로 저장하는 클래스를 만들어주었다. dumperoption를 사용하여 yaml의 출력형식을 설정하고, yaml 객체를 초기화해주었고 yaml 출력 옵션을 생성, 저장한 옵션을 사용하여 yaml 객체를 생성해주었다. 그 후 yaml 파일을 읽어 amp<string, programlauncher commad> 로 변환시켜주었고 파일 내용을 yaml로 파싱하여 map으로 반환시켜주었다.

실행결과)


HW7
.vscode
settings.json
bin
images
lib
gson-2.11.0.jar
snakeyaml-2.2.jar
src
FileImporter.java
FileImporterLoaderA...
FileLoader.java
FileLoaderImporterA...
MainTest.java
ProgramLauncherCo...
ProgramLauncherCo...
ProgramLauncherCo...
ProgramLauncherCo...
ProgramLauncherCo...
commands.csv
commands.json
commands.xml
README.md


.vscode
settings.json
bin
images
lib
gson-2.11.0.jar
snakeyaml-2.2.jar
src
FileImporter.java
FileImporterLoaderA...
FileLoader.java
FileLoaderImporterA...
MainTest.java
ProgramLauncherCo...
ProgramLauncherCo...
ProgramLauncherCo...
ProgramLauncherCo...
ProgramLauncherCo...
commands.csv
commands.json
commands.xml
commands2.csv
commands2.json
commands2.xml
commands2.yaml
README.md

Commands2.csv)

commands2.csv > data

```
Notepad,notepad,images/Notepad-icon.png
Edge,cmd /c start msedge,images/MS-Edge-Icon.png
Paint,mspaint,images/MS-Paint/Icon.png

```

Commands2.json)

commands2.json > ...

```
{
  "Notepad": {
    "name": "Notepad",
    "executable": "notepad",
    "icon": "images/Notepad-icon.png"
  },
  "SnippingTool": {
    "name": "Edge",
    "executable": "cmd /c start msedge",
    "icon": "images/MS-Edge-Icon.png"
  },
  "Paint": {
    "name": "Paint",
    "executable": "mspaint",
    "icon": "images/MS-Paint/Icon.png"
  }
}
```

Commands2.xml)

commands2.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<commands>
    <command>
        <name>Notepad</name>
        <executable>notepad</executable>
        <icon>images/Notepad-icon.png</icon>
    </command>
    <command>
        <name>Edge</name>
        <executable>cmd /c start msedge</executable>
        <icon>images/MS-Edge-Icon.png</icon>
    </command>
    <command>
        <name>Paint</name>
        <executable>mspaint</executable>
        <icon>images/MS-Paint/Icon.png</icon>
    </command>
</commands>

```

Commands2.yaml)

commands2.yaml

```
Notepad: !!ProgramLauncherCommand {
  executable: notepad,
  icon: images/Notepad-icon.png,
  name: Notepad
}
SnippingTool: !!ProgramLauncherCommand {
  executable: cmd /c start msedge,
  icon: images/MS-Edge-Icon.png,
  name: Edge
}
Paint: !!ProgramLauncherCommand {
  executable: mspaint,
  icon: images/MS-Paint/Icon.png,
  name: Paint
}

```