**HW7**

보고서

lab 07 자바 프로그래밍 2분반 32207522 양상훈

제출일 : 24/11/19

**실행결과**

**텍스트, 스크린샷, 소프트웨어, 멀티미디어 소프트웨어이(가) 표시된 사진

자동 생성된 설명**

**FileImporter, FileLoader Interface**

// 자바프로그래밍 2분반 32207522 양상훈

import java.util.Map;

// 기존 파일을 기반으로 데이터를 읽고 쓰기

public interface FileImporter<T> {

Map<String, T> importFile(String filepath); // 파일 경로에서 데이터 가져오기

void exportFile(String filepath, Map<String, T> map); // 데이터를 파일로 내보내기

}

// 자바프로그래밍 2분반 32207522 양상훈

import java.util.Map;

// 새로운 파일 형식을 생성하고 데이터를 저장할 때 사용

public interface FileLoader<T> {

Map<String, T> load(String filepath); // 파일 경로에서 데이터를 로드

void save(String filepath, Map<String,T> map); // 데이터를 파일로 저장

}

기능은 비슷하나 서로 다른 두 인터페이스이다.

**ProgramLauncherCommand Class**

// 자바프로그래밍 2분반 32207522 양상훈

public class ProgramLauncherCommand {

private String name; // 프로그램 이름

private String executable; // 실행할 명령어

private String icon; // 아이콘의 경로

public ProgramLauncherCommand(String name, String executable, String icon) {

this.name = name;

this.executable = executable;

this.icon = icon;

}

public String getName() {

return name;

}

public String getExecutable() {

return executable;

}

public String getIcon() {

return icon;

}

@Override

public String toString() {

return String.format("Name: %s, Executable: %s, Icon: %s", name, executable, icon);

}

}

command파일을 읽어 들여 커맨드 데이터(name, executable, icon)를 객체로 저장하는 클래스이다.

**ProgramLauncherCommandJSONFileImporter Class**

// 자바프로그래밍 2분반 32207522 양상훈

// Gson 라이브러리를 이용하여 JSON 파일을 읽음 : Pretty Print

import com.google.gson.Gson;

import com.google.gson.GsonBuilder;

import com.google.gson.JsonObject;

import com.google.gson.reflect.TypeToken;

import java.io.FileReader;

import java.io.FileWriter;

import java.io.IOException;

import java.lang.reflect.Type;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class ProgramLauncherCommandJSONFileImporter implements FileImporter<ProgramLauncherCommand> {

// Gson 객체 생성, Pretty Printing이 목적

private final Gson gson = new GsonBuilder().setPrettyPrinting().create();

// JSON 파일에서 데이터를 읽어서 Map<String, ProgramLauncherCommand> 형태로 반환하는 메서드

@Override

public Map<String, ProgramLauncherCommand> importFile(String filepath) {

// FileReader를 사용해 JSON 파일을 읽는다.

try (FileReader reader = new FileReader(filepath)) {

// GSON을 이용해 JsonObject로 변환한다.

JsonObject root = gson.fromJson(reader, JsonObject.class);

// "commands" 배열 읽기

Type listType = new TypeToken<List<ProgramLauncherCommand>>() {}.getType();

List<ProgramLauncherCommand> commands = gson.fromJson(root.get("commands"), listType);

// 배열을 Map으로 변환

Map<String, ProgramLauncherCommand> map = new HashMap<>();

for (ProgramLauncherCommand command : commands) {

map.put(command.getName(), command);

}

return map;

} catch (IOException e) {

throw new RuntimeException("Failed to import JSON file", e);

}

}

// Map<String, ProgramLauncherCommand> 데이터를 JSON 파일로 저장하는 메서드

@Override

public void exportFile(String filepath, Map<String, ProgramLauncherCommand> map) {

// FileWriter를 이용하여 JSON파일을 쓸 수 있도록 한다.

try (FileWriter writer = new FileWriter(filepath)) {

// Map 객체를 JSON 배열로 변환한다. "commands" 키 아래 저장한다.

JsonObject root = new JsonObject();

root.add("commands", gson.toJsonTree(map.values()));

// Pretty Print형식으로 JSON 파일을 저장한다.

gson.toJson(root, writer);

} catch (IOException e) {

throw new RuntimeException("Failed to export JSON file", e);

}

}

}

Gson을 이용하여 JSON파일을 생성할 때 한 줄 출력이 아닌 Pretty Print 형식으로 파일을 저장한다.

FileImporter 인터페이스를 implements 하므로 기능들을 구현해야한다.

public Map<String, ProgramLauncherCommand> importFile(String filepath) : 파일 경로를 받아JSON파일을 읽고 Map으로 반환한다.

public void exportFile(String filepath, Map<String, ProgramLauncherCommand> map) : Gson을 이용해 Map 데이터로 지정된 경로에 JSON 파일을 생성한다. commands라는 키 아래에 배열로 저장한다.

FileWriter에 의해 새로운 파일을 생성한다.

**ProgramLauncherCommandXMLFileImporter Class**

// 자바프로그래밍 2분반 32207522 양상훈

import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Element; // Element 이용

import java.io.FileWriter;

import java.util.HashMap;

import java.util.Map;

public class ProgramLauncherCommandXMLFileImporter implements FileImporter<ProgramLauncherCommand> {

// XML 파일을 읽어와 Map으로 변환

@Override

public Map<String, ProgramLauncherCommand> importFile(String filepath) {

Map<String, ProgramLauncherCommand> map = new HashMap<>();

try {

var doc = DocumentBuilderFactory.newInstance().newDocumentBuilder().parse(filepath);

var commands = doc.getDocumentElement().getElementsByTagName("command");

// 각 <command> 요소를 순회하며 데이터를 추출

for (int i = 0; i < commands.getLength(); i++) {

var element = (Element) commands.item(i);

String name = element.getElementsByTagName("name").item(0).getTextContent();

String executable = element.getElementsByTagName("executable").item(0).getTextContent();

String icon = element.getElementsByTagName("icon").item(0).getTextContent();

// 추출한 데이터를 Map에 저장

map.put(name, new ProgramLauncherCommand(name, executable, icon));

}

} catch (Exception e) {

throw new RuntimeException("Failed to import XML file", e);

}

return map;

}

// Map 데이터를 XML 형식으로 저장

@Override

public void exportFile(String filepath, Map<String, ProgramLauncherCommand> map) {

try (FileWriter writer = new FileWriter(filepath)) {

writer.write("<commands>\n");

// Map의 값 가져오기

for (Map.Entry<String, ProgramLauncherCommand> entry : map.entrySet()) {

ProgramLauncherCommand command = entry.getValue();

writer.write(" <command>\n");

writer.write(" <name>" + command.getName() + "</name>\n");

writer.write(" <executable>" + command.getExecutable() + "</executable>\n");

writer.write(" <icon>" + command.getIcon() + "</icon>\n");

writer.write(" </command>\n");

}

writer.write("</commands>");

} catch (Exception e) {

throw new RuntimeException("Failed to export XML file", e);

}

}

}

FileImporter 인터페이스를 implements 하므로 기능들을 구현해야한다.

public Map<String, ProgramLauncherCommand> importFile(String filepath) : 파일 경로를 받아 XML 형식의 파일을 읽고 Map으로 반환한다.

public void exportFile(String filepath, Map<String, ProgramLauncherCommand> map) : 주어진 Map 데이터로 지정된 경로에 XML 파일을 생성한다.

**ProgramLauncherCommandCSVFileLoader Class**

// 자바프로그래밍 2분반 32207522 양상훈

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.FileWriter;

import java.util.HashMap;

import java.util.Map;

public class ProgramLauncherCommandCSVFileLoader implements FileLoader<ProgramLauncherCommand> {

// CSV 파일을 로드하여 ProgramLauncherCommand 객체로 변환 후 Map에 저장

@Override

public Map<String, ProgramLauncherCommand> load(String filepath) {

Map<String, ProgramLauncherCommand> map = new HashMap<>();

// BufferedReader 객체 생성 : 파일 읽기 위함

try (BufferedReader reader = new BufferedReader(new FileReader(filepath))) {

String line;

while ((line = reader.readLine()) != null) {

String[] parts = line.split(",");

if (parts.length == 3) {

map.put(parts[0], new ProgramLauncherCommand(parts[0], parts[1], parts[2]));

}

}

} catch (Exception e) {

throw new RuntimeException("Failed to load CSV file", e);

}

return map;

}

// Map에 저장된 데이터를 CSV 파일로 저장

@Override

public void save(String filepath, Map<String, ProgramLauncherCommand> map) {

// FileWriter : 파일에 데이터 쓰기

try (FileWriter writer = new FileWriter(filepath)) {

for (ProgramLauncherCommand command : map.values()) {

writer.write(String.format("%s,%s,%s\n", command.getName(), command.getExecutable(), command.getIcon()));

}

} catch (Exception e) {

throw new RuntimeException("Failed to save CSV file", e);

}

}

}

ProgramLauncherCommandCSVFileLoader 클래스는 FileLoader 인터페이스를 구현한다. 메서드 기능을 정의해야한다.

public Map<String, ProgramLauncherCommand> load(String filepath) : 파일 경로를 받아 CSV파일을 읽어 데이터를 Map 형태로 반환한다.

public void save(String filepath, Map<String, ProgramLauncherCommand> map) : 주어진 Map 데이터를 지정된 경로에 CSV 파일 형식으로 저장한다.

**FileImporterLoaderAdapter Class**

// 자바프로그래밍 2분반 32207522 양상훈

import java.util.Map;

// Importer -> Loader

// FileLoader : Target

public class FileImporterLoaderAdapter<T> implements FileLoader<T> {

private final FileImporter<T> fileImporter; // FileImporter 참조

public FileImporterLoaderAdapter(FileImporter<T> fileImporter) {

this.fileImporter = fileImporter;

}

@Override

public Map<String, T> load(String filepath) {

return fileImporter.importFile(filepath); // Adapter를 이용하여 importFile을 수행하도록 해줌.

}

@Override

public void save(String filepath, Map<String, T> map) {

fileImporter.exportFile(filepath, map); // Adapter를 이용하여 exportFile을 수행하도록 해줌.

}

}

Target은 FileLoader이다. FileLoader 인터페이스를 구현한다.

FileImporter 객체를 참조하고 FileLoader 메서드 정의 안에 FileImporter 메서드를 이용한다.

이로서 FileImporter를 FileLoader로 Convert 하였다.

**FileLoaderImporterAdapter Class**

// 자바프로그래밍 2분반 32207522 양상훈

import java.util.Map;

// Loader ->Importer

// FileImporter : Target

public class FileLoaderImporterAdapter<T> implements FileImporter<T> {

private final FileLoader<T> fileLoader; // FileLoader 참조

public FileLoaderImporterAdapter(FileLoader<T> fileLoader) {

this.fileLoader = fileLoader;

}

@Override

public Map<String, T> importFile(String filepath) {

return fileLoader.load(filepath); // Adapter를 이용하여 load을 수행하도록 해줌.

}

@Override

public void exportFile(String filepath, Map<String, T> map) {

fileLoader.save(filepath, map); // Adapter를 이용하여 save을 수행하도록 해줌.

}

}

Target은 FileImporter이다. FileImporter인터페이스를 구현한다.

FileLoader객체를 참조하고 FileImporter메서드 정의 안에 FileLoader메서드를 이용한다.

이로서 FileLoader를 FileImporter로 Convert 하였다.

위의 두 어댑터를 이용하여 FileImporter와 FileLoader가 양방향으로 전환이 가능해졌다.

**MainTest 클래스**

// 자바프로그래밍 2분반 32207522 양상훈

import java.util.Map;

public class MainTest {

public static void main(String[] args) {

FileImporter<ProgramLauncherCommand> jsonImporter = new ProgramLauncherCommandJSONFileImporter();

Map<String, ProgramLauncherCommand> jsonMap = jsonImporter.importFile("commands.json");

FileImporter<ProgramLauncherCommand> xmlImporter = new ProgramLauncherCommandXMLFileImporter();

Map<String, ProgramLauncherCommand> xmlMap = xmlImporter.importFile("commands.xml");

FileLoader<ProgramLauncherCommand> csvLoader = new ProgramLauncherCommandCSVFileLoader();

Map<String, ProgramLauncherCommand> csvMap = csvLoader.load("commands.csv");

// === JSON 파일 처리 ===

System.out.println("\n=== JSON File Processing ===");

for (Map.Entry<String, ProgramLauncherCommand> entry : jsonMap.entrySet()) {

System.out.println(entry.getKey() + " " + entry.getValue().getExecutable() + " " + entry.getValue().getIcon());

}

jsonImporter.exportFile("commands2.json", jsonMap);

// === XML 파일 처리 ===

System.out.println("\n=== XML File Processing ===");

for (Map.Entry<String, ProgramLauncherCommand> entry : xmlMap.entrySet()) {

System.out.println(entry.getKey() + " " + entry.getValue().getExecutable() + " " + entry.getValue().getIcon());

}

xmlImporter.exportFile("commands2.xml", xmlMap);

// === CSV 파일 처리 ===

System.out.println("\n=== CSV File Processing ===");

for (Map.Entry<String, ProgramLauncherCommand> entry : csvMap.entrySet()) {

System.out.println(entry.getKey() + " " + entry.getValue().getExecutable() + " " + entry.getValue().getIcon());

}

csvLoader.save("commands2.csv", csvMap);

// === Adapter를 사용하여 JSON 파일 처리 ===

System.out.println("\n=== JSON File Processing Using Adapter ===");

FileLoader<ProgramLauncherCommand> jsonLoaderAdapter = new FileImporterLoaderAdapter<>(jsonImporter);

Map<String, ProgramLauncherCommand> jsonAdapterMap = jsonLoaderAdapter.load("commands.json");

for (Map.Entry<String, ProgramLauncherCommand> entry : jsonAdapterMap.entrySet()) {

System.out.println(entry.getKey() + " " + entry.getValue().getExecutable() + " " + entry.getValue().getIcon());

}

jsonImporter.exportFile("commands3.json", jsonMap);

// === Adapter를 사용하여 XML 파일 처리 ===

System.out.println("\n=== XML File Processing Using Adapter ===");

FileLoader<ProgramLauncherCommand> xmlLoaderAdapter = new FileImporterLoaderAdapter<>(xmlImporter);

Map<String, ProgramLauncherCommand> xmlAdapterMap = xmlLoaderAdapter.load("commands.xml");

for (Map.Entry<String, ProgramLauncherCommand> entry : xmlAdapterMap.entrySet()) {

System.out.println(entry.getKey() + " " + entry.getValue().getExecutable() + " " + entry.getValue().getIcon());

}

xmlImporter.exportFile("commands3.xml", xmlMap);

// === Adapter를 사용하여 CSV 파일 처리 ===

System.out.println("\n=== CSV File Processing Using Adapter ===");

FileImporter<ProgramLauncherCommand> csvImporterAdapter = new FileLoaderImporterAdapter<>(csvLoader);

Map<String, ProgramLauncherCommand> csvAdapterMap = csvImporterAdapter.importFile("commands.csv");

for (Map.Entry<String, ProgramLauncherCommand> entry : csvAdapterMap.entrySet()) {

System.out.println(entry.getKey() + " " + entry.getValue().getExecutable() + " " + entry.getValue().getIcon());

}

csvLoader.save("commands3.csv", csvMap);

// yourcode : Iterator Pattern

System.out.println();

System.out.println("Using Iterator Pattern:");

ProgramLauncherCommandIterator jsonIterator = new ProgramLauncherCommandIterator(jsonMap);

while (jsonIterator.hasNext()) {

ProgramLauncherCommand command = jsonIterator.next();

System.out.println(command.getName() + " " + command.getExecutable() + " " + command.getIcon());

}

jsonImporter.exportFile("commands4.json", jsonMap);

System.out.println();

System.out.println("Using Iterator Pattern:");

ProgramLauncherCommandIterator xmlIterator = new ProgramLauncherCommandIterator(xmlMap);

while (xmlIterator.hasNext()) {

ProgramLauncherCommand command = xmlIterator.next();

System.out.println(command.getName() + " " + command.getExecutable() + " " + command.getIcon());

}

xmlImporter.exportFile("commands4.xml", xmlMap);

System.out.println();

System.out.println("Using Iterator Pattern:");

ProgramLauncherCommandIterator csvIterator = new ProgramLauncherCommandIterator(csvMap);

while (csvIterator.hasNext()) {

ProgramLauncherCommand command = csvIterator.next();

System.out.println(command.getName() + " " + command.getExecutable() + " " + command.getIcon());

}

csvLoader.save("commands4.csv", csvMap);

}

}

1. 기본 방식 : json, xml은 FileImporter, csv는 FileLoader 이용하여 파일 생성
2. Adapter 이용 : json, xml은 FileImporterLoaderAdapter 이용, csv는 FileLoaderImporter 이용하여 파일 생성
3. Iterator 이용하여 파일 생성

**YourCode**

Iterator 패턴을 이용하여 순회하며 데이터를 출력하고 파일을 저장한다. 제거 기능은 없으므로 remove는 구현하지 않았음.

**ProgramLauncherCommandIterator Class**

// 자바프로그래밍 2분반 32207522 양상훈

import java.util.Iterator;

import java.util.Map;

// yourcode

// Iterator 패턴을 사용하여 Map의 ProgramLauncherCommand 요소를 순회

public class ProgramLauncherCommandIterator implements Iterator<ProgramLauncherCommand> {

// Map의 값들을 순회하기 위한 Iterator

private final Iterator<ProgramLauncherCommand> iterator;

// Map을 받아서 해당 Map의 값들을 순회할 Iterator를 생성

public ProgramLauncherCommandIterator(Map<String, ProgramLauncherCommand> map) {

this.iterator = map.values().iterator(); // Map의 value를 기반으로 Iterator 생성

}

// 순회할 다음 요소가 있는지 여부를 반환

@Override

public boolean hasNext() {

return iterator.hasNext();

}

// 현재 Iterator가 가리키는 다음 요소를 반환

@Override

public ProgramLauncherCommand next() {

return iterator.next();

}

}

**MainTest Class에 추가**

// yourcode : Iterator Pattern

System.out.println();

System.out.println("Using Iterator Pattern:");

ProgramLauncherCommandIterator jsonIterator = new ProgramLauncherCommandIterator(jsonMap);

while (jsonIterator.hasNext()) {

ProgramLauncherCommand command = jsonIterator.next();

System.out.println(command.getName() + " " + command.getExecutable() + " " + command.getIcon());

}

jsonImporter.exportFile("commands4.json", jsonMap);

System.out.println();

System.out.println("Using Iterator Pattern:");

ProgramLauncherCommandIterator xmlIterator = new ProgramLauncherCommandIterator(xmlMap);

while (xmlIterator.hasNext()) {

ProgramLauncherCommand command = xmlIterator.next();

System.out.println(command.getName() + " " + command.getExecutable() + " " + command.getIcon());

}

xmlImporter.exportFile("commands4.xml", xmlMap);

System.out.println();

System.out.println("Using Iterator Pattern:");

ProgramLauncherCommandIterator csvIterator = new ProgramLauncherCommandIterator(csvMap);

while (csvIterator.hasNext()) {

ProgramLauncherCommand command = csvIterator.next();

System.out.println(command.getName() + " " + command.getExecutable() + " " + command.getIcon());

}

csvLoader.save("commands4.csv", csvMap);

}