

实验报告二

课程名称： 软件设计模式实验

学 期： 2024 年~2025 年第1 学期

姓 名： 张斯涵

学 号： 2022112529

指导教师：何滨

本人郑重承诺：

* 所呈交的报告（包括代码），均系本人独立完成，没有抄袭他人的内容；
* 本人已清楚：对报告中的抄袭行为，将被取消本门课程的成绩。

学生签名：

签名日期：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 表述部分正确 (40%) | 编程部分正确(40%) | 报告清整洁度及高质量排版(20%) |  | 总分(100%) | 批阅人签字及批阅日期 |
|  |  |  |  |  |  |

要点：

相关OO设计原则之依赖倒转原则(Dependency Inversion Principle, DIP

一、简述依赖倒转

1、依赖倒转原则，高层模块不依赖低层模块，它们共同依赖同一个抽象

2、这个抽象接口通常是由高层模块定义，低层模块实现

3、同时，抽象不要依赖具体实现细节，具体实现细节依赖抽象

4、高层模块就是调用端，低层模块就是具体实现类

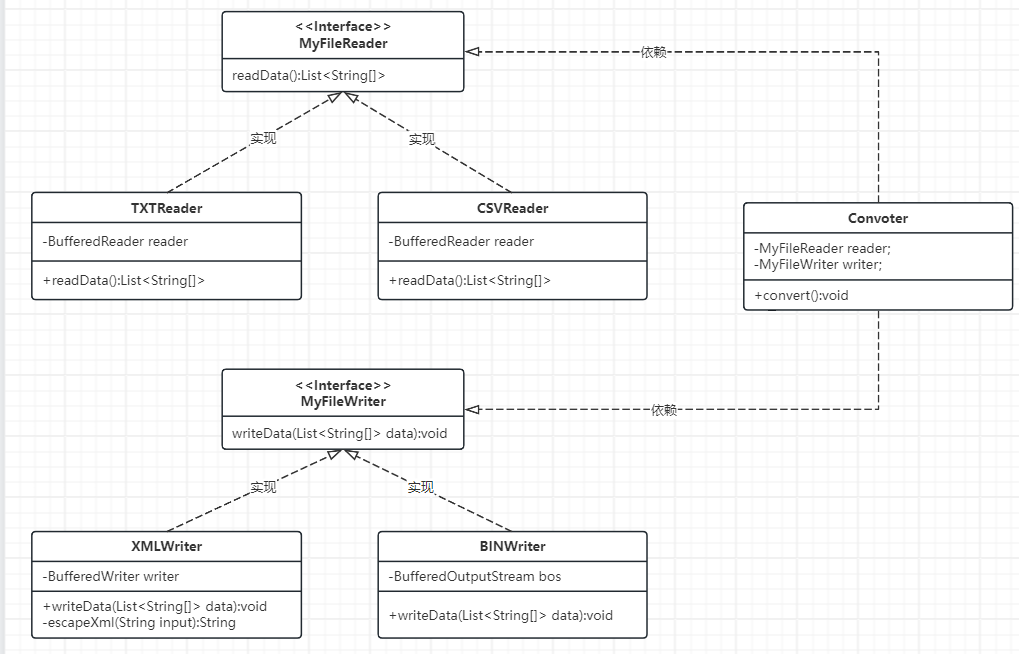
5、抽象就是指接口或抽象类，细节就是实现类

二、编程实现一个简易数据文件格式转换器

输入格式：txt、CSV

输出格式：xml、二进制

1. 用类图表示程序的主要结构



1. 主要代码

**MyFileReader.java作为读入接口：**

import java.io.IOException;

import java.util.List;

public interface MyFileReader {

List<String[]> readData() throws IOException;

}

**MyFileWriter.java作为输出接口：**

import java.io.IOException;

import java.util.List;

public interface MyFileWriter {

void writeData(List<String[]> data) throws IOException;

}

**MyFileReader有两个实现类：**

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

public class TXTReader implements MyFileReader {

private BufferedReader reader;

public TXTReader(String Path) throws FileNotFoundException {

FileReader fr = new FileReader(Path);

this.reader = new BufferedReader(fr); //利用装饰者模式包装reader

}

@Override

public List<String[]> readData() throws IOException {

List<String[]> data = new ArrayList<>();

String line;

while ((line = reader.readLine()) != null) { // 按行读取

line = line.trim();

if (line.isEmpty()) continue; // 跳过空行

// 假设TXT文件每行是以空格分隔的数据

String[] tokens = line.split("\\s+");

if (tokens.length >= 2) { // 确保至少有两个字段

data.add(new String[]{tokens[0], tokens[1]});

}

}

reader.close();

return data;

}

}

**第二个：**

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

public class CSVReader implements MyFileReader {

private BufferedReader reader;

public CSVReader(String Path) throws IOException{

FileReader fr = new FileReader(Path);

this.reader = new BufferedReader(fr); //利用装饰者模式包装reader

}

@Override

public List<String[]> readData() throws IOException {

List<String[]> data = new ArrayList<>();

String line;

while ((line = reader.readLine()) != null) { // 按行读取

line = line.trim();

if (line.isEmpty()) continue; // 跳过空行

// 跳过仅包含逗号的行（如末尾多余的逗号）

if (line.equals(",")) continue;

// 假设CSV文件每行是以逗号分隔的数据

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

if (tokens.length >= 2) { // 确保至少有两个字段

data.add(new String[]{tokens[0], tokens[1]});

}

}

reader.close();

return data;

}

}

**MyFileWriter有两个实现类：**

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.util.List;

public class XMLWriter implements MyFileWriter{

// 用于写成XML文件

private BufferedWriter writer;

public XMLWriter(String filePath) throws IOException {

FileWriter fw = new FileWriter(filePath);

this.writer = new BufferedWriter(fw); // 使用 BufferedWriter 装饰

}

@Override

public void writeData(List<String[]> data) throws IOException {

writer.write("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n");

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

for (String[] record : data) {

// 确保记录有两个字段

if (record.length >= 2) {

String id = escapeXml(record[0]);

String stuName = escapeXml(record[1]);

writer.write(String.format("<String ID=\"%s\" stuName=\"%s\"/>\n", id, stuName));

}

}

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

writer.close();

}

//简单转义

private String escapeXml(String input) {

return input.replace("&", "&amp;")

.replace("\"", "&quot;")

.replace("'", "&apos;")

.replace("<", "&lt;")

.replace(">", "&gt;");

}

}

**第二个：**

import java.io.BufferedOutputStream;

import java.io.FileNotFoundException;

import java.io.FileOutputStream;

import java.io.IOException;

import java.util.List;

public class BINWriter implements MyFileWriter{

private BufferedOutputStream bos;

public BINWriter(String Path) throws FileNotFoundException {

FileOutputStream fos = new FileOutputStream(Path);

this.bos = new BufferedOutputStream(fos); // 使用 BufferedOutputStream 装饰

}

@Override

public void writeData(List<String[]> data) throws IOException {

for (String[] record : data) {

for (int i = 0; i < record.length; i++) {

bos.write(record[i].getBytes("UTF-8"));

if (i < record.length - 1) {

bos.write(','); // 使用逗号分隔字段

}

}

bos.write('\n'); // 换行分隔记录

}

bos.flush();

bos.close();

}

}

**设计一个接口，高层模块依赖这个接口，低层模块实现接口：**import java.io.IOException;

import java.util.List;

public class Converter {

private MyFileReader reader;

private MyFileWriter writer;

// 构造函数依赖于抽象接口，而非具体实现

public Converter(MyFileReader reader, MyFileWriter writer) {

this.reader = reader;

this.writer = writer;

}

public void convert() throws IOException {

List<String[]> data = reader.readData();

writer.writeData(data);

}

}

三、 体会和收获

通过本次实验，我对依赖倒转原则有了更加深刻的理解。

依赖倒转原则强调，高层模块不应依赖低层模块，而是它们应该依赖同一个抽象。这种设计思想使得程序具备更好的灵活性和可扩展性，能够较为容易地实现模块的替换、扩展以及维护。

以本次项目为例，如果在软件的生命周期中出现了额外的需求，比如 增加从xml到txt文件类型的转换，只需要在MyFileReader的接口下扩展实现XMLReader这一实现类。在MyFileWriter接口下实现TXTWriter这一实现类即可。同时该方法符合开闭原则，对扩展开放，对修改源码封闭。

通过将具体实现与抽象分离，程序的扩展性、灵活性、可维护性都得到了极大的提升。这种设计思想和方法不仅适用于本次的文件格式转换器，也同样适用于大型软件项目的设计与开发。这次实验为今后的编程实践奠定了良好的基础，增强了我在面向对象设计和设计模式方面的能力。

附录：格式示例

1. 格式一.txt

2019110001 李一

2019110002 王二

2019110003 张三

2019110004 刘四

2019110005 陈五

2019110006 杨六

1. 格式二.cvs (带逗号)

2019110001,李一

2019110002,王二

2019110003,张三

2019110004,刘四

2019110005,陈五

2019110006,杨六

,

1. 格式三. xml

<?xml version="1.0" encoding="UTF-8"?>

<StringTable>

<String ID="2019110001" stuName="李一"/>

<String ID="2019110002" stuName="王二"/>

<String ID="2019110003" stuName="张三"/>

<String ID="2019110004" stuName="刘四"/>

<String ID="2019110005" stuName="陈五"/>

<String ID="2019110006" stuName="杨六"/>

</StringTable>