**package** io;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.ObjectOutputStream;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Vector;

**import** javax.swing.JFileChooser;

**import** javax.swing.JOptionPane;

**import** javax.swing.table.DefaultTableModel;

**public** **class** ScoreImpl **implements** Score {

**private** List<ScoreDTO> list;

**public** ScoreImpl() { // 생성자

list = **new** ArrayList<ScoreDTO>();

}

@Override

**public** **void** intput(ScoreDTO dto) {

list.add(dto);

JOptionPane.*showMessageDialog*(**null**, "등록 완료"); // 여기선 This라는 Frame이 없으니 null로 잡아줌

}

@Override

**public** **void** print(DefaultTableModel model) {

model.setRowCount(0); // RowCount(0)의 의미는 그 안에 있는 모든 내용을 지우라는 의미

// 기존에 있는 걸 지우고 새로이 입력한 값들이 들어오게끔!!!

// ex) [io.ScoreDTO@b20281a]을 지우고 [io.ScoreDTO@b20281a, io.ScoreDTO@215d5581]

// 새로이 입력하면

// 우리 눈에는 [io.ScoreDTO@215d5581]만 새롭게 들어온 것처럼 보인다

**for** (ScoreDTO dto : list) {// 모든 데이터가 list에 죄다 쌓여 있다

Vector<String> v = **new** Vector<String>();

// 돌 때마다 Vector가 생겨야 한다. 한줄당 Vector로 들어가니까

v.add(dto.getHak());

v.add(dto.getName());

v.add(dto.getKor() + ""); // 숫자를 문자열로 변환 - dto.getKor()은 int기본형이다

v.add(dto.getEng() + "");

v.add(dto.getMath() + "");

v.add(dto.getTot() + "");

v.add(String.*format*("%2f", dto.getAvg()));

model.addRow(v); // Table에다 Vector를 붙이면 한줄이 붙게 되는 거지~~

} // for

} // interface의 구현체

@Override

**public** **void** search(DefaultTableModel model) {

String hak = JOptionPane.*showInputDialog*(**null**, "학번을 입력해주세요", "검색", JOptionPane.***QUESTION\_MESSAGE***);

**if** (hak == **null** || hak.length() == 0)

**return**; // 만약에 취소버튼을 누르면 null값 가져온다 - 취소버튼 누르면 그냥 나가라~

// hak.length() == 0 - 글씨를 넣지도 않았는데 확인버튼이 눌러지는 걸 막기위해 써주는 것이고, 글자수가 0이면 입력을 안했다는

// 것을 의미

**int** sw = 0;

**for** (ScoreDTO dto : list) {

// = if(int i = 0; i<model.getRowcount(); i++) {

**if** (hak.equals(dto.getHak())) { // 학번이 테이블에 있는 학번이랑 같으냐

model.setRowCount(0); // 테이블에 있는 값을 초기화시키고

// list에서 꺼내온 내용을 Vector에 집어넣는다

Vector<String> v = **new** Vector<String>();

v.add(dto.getHak());

v.add(dto.getName());

v.add(dto.getKor() + "");

v.add(dto.getEng() + "");

v.add(dto.getMath() + "");

v.add(dto.getTot() + "");

v.add(String.*format*("%2f", dto.getAvg()));

model.addRow(v);

sw = 1;

}

} // for

**if** (sw == 0)

JOptionPane.*showMessageDialog*(**null**, "찾고자 하는 학번이 없습니다");

}

@Override

**public** **void** tot\_desc() {

}

@Override

**public** **void** save() {

JFileChooser chooser = **new** JFileChooser();

**int** result = chooser.showSaveDialog(**null**); // 저장 다이얼로그

// int result 로 받는 건 저장 버튼이 눌렸는지 취소버튼이 눌렸는지 알기 위해서다

File file = **null**;

**if** (result == JFileChooser.***APPROVE\_OPTION***) { // save버튼 눌렀으면

file = chooser.getSelectedFile(); // 선택한 파일을 가져와라

// \* 객체는 IO를 통해서 못가니까 byte단위로 쪼개서 보내야 하는데

// 이 쪼개는 과정을 객체 직렬화한다고 한다

// byte 단위로 쪼개려면 DTO에 implements Serializable(직렬화) 입력

// Serializable은 추상메소드가 없으니 Override시켜줄 것도 없다

}

// ----------------------------------

**if** (file == **null**)

**return**; // 파일에서 선택하지 않고 취소를 누르면 null값이 들어온다 나가라~~

**try** {

ObjectOutputStream oos = **new** ObjectOutputStream(**new** FileOutputStream(file));

// 파일 집어넣기

**for** (ScoreDTO dto : list) {

oos.writeObject(dto); // dto를 넘겨줘라

} // for

oos.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

@Override

**public** **void** load() {

}

}

// Interface : Score(부모) - 추상메소드

// Class : ScoreImpl(자식) - ScoreService 대신

// 모든 데이터를 받아 데이터를 뿌리고 함수들의 입력 출력 등의 결과창이 될 수 있도록 하는 역할이

// 바로 이 ScoreImpl

// ScoreImpl에 잡혀있는 모든 애들은 interface를 거쳐야하며

// 그 구현부는 ScoreImpl

// 생성할 때 ScoreImpl impl = new ScoreImpl(); // 1 : 1 관계

// Score score = new ScoreImpl(); 같은 생성이지만 결합도를 낮춰준다

// Inteface는 하나인데 Class가 여러개로 넘쳐나면 Class를 대표해 주는 것이 Interface(Score)가 하나뿐이다

// 클래스는 여러개인데 참조하는 부모가 하나뿐인 것이 좋다

// 항상 접근은 부모걸로 접근하고 생성은 자식걸로 생성하는 것이 좋다