Swing 특성

* Swing은 보통 AWT에 대한 비교를 많이 한다.
* 컴포넌트들은 플랫폼 독립적이다.
* 컴포넌트가 AWT에 비하여 가볍다. AWT는 OS(시스템콜)를 자주 호출하지만, Swing은 그럴 필요가 없다.
* pluggable look and feel이 가능하다. (실행시간에 프로그래머가 커스터마이즈 세팅으로 외관을 바꿀 수 있다.)
* 유용한 컴포넌트가 AWT에 비해 더 늘어났다.
* MVC모델을 지원한다.(Model View Controller)
  + MVC란 사용자 인터페이스, 데이터 및 논리 제어를 구현하는데 널리 사용되는 소프트웨어 디자인 패턴이다. 소프트웨어의 비즈니스 로직과 화면을 구분하는데 중점을 두고 있다.

Kotlin

* Kotlin은 JVM, Android, 브라우저를 위한 정적 타입의 프로그래밍 언어이다.
* 정적으로 타입이 지정되므로 JAVA와 동일한 타입 안정성을 갖는다.
* ClientEditor 프로젝트에서는 “위임”을 사용하기 위해 사용되었다.

**ClientEditor.java**

package clienteditor;

public class ClientEditor extends javax.swing.JPanel {

private Client client = Client.createTestClient();

public ClientEditor() {

initComponents();

bindingGroup.addBindingListener(new LoggingBindingListener(validationMsgLabel));

}

public Client **getClient**() {

return client;

}

public void **setClient**(Client client) {

Client oldClient = this.client;

this.client = client;

firePropertyChange("client", oldClient, client);

}

public static void **main**(String args[]) {

java.awt.EventQueue.invokeLater(new Runnable() {

public void run() {

javax.swing.JFrame frame = new javax.swing.JFrame("Client Editor");

frame.setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);

frame.getContentPane().add(new ClientEditor());

frame.pack();

frame.setVisible(true);

}

});

}

}

**Client.java**

package clienteditor;

// 위임 프로퍼티를 사용하기 위한 import

import java.beans.PropertyChangeListener;

import java.beans.PropertyChangeSupport; // 클래스의 리스너 목록을 관리한다. 이벤트가 들어오면 목록의 모든 리스너에게 이벤트를 통지한다.

public class **Client** {

private String firstName; // 이름

private String surname; // 성

private String nickname; // 별명

private int age; // 나이

/\*\* Sex of the client (0 - female, 1 - male). \*/

private int sex; // 0일경우 여자, 1일경우 남자

/\*\* Marital status of the client (0 - single, 1 - married, 2 - separated, 3 - divorced) \*/

private int maritalStatus; // 0일경우 미혼, 1일경우 기혼, 3일경우 분리, 4일경우 이혼

private String email; // 이메일

private String web; // 웹 사이트 주소

private String im; // 메신저

private final PropertyChangeSupport changeSupport = new PropertyChangeSupport(this);

// 코틀린이 제공해주는 관례중 독특하면서도 가장 중요한 기능인 위임 프로퍼티(delegated property)

// “위임”이란 자신이 직접 작업을 수행하지 않고 다른 객체에게 그 작업을 처리하도록 맡기는 디자인 패턴이다.

// 즉, 프로퍼티 필드에 접근하는 getter/setter 메소드를 가지는 다른 객체를 만들어서 그 객체에 프로퍼티 필드 접근 로직을 위임하는 것이다.

// 위임을 사용하는 이유는 상위 클래스의 내용이 변경되는 경우 하위 클래스가 상위 클래스에 의존하고 있던 내용들이 완전히 틀어지면서 에러가 발생하기 때문이다.(상속의 문제점)

// 상속과의 차이

상속은 상위 클래스의 변수와 메소드를 모두 받기 때문에 재 구현할 필요가 없어서 편리하지만 의존도가 높기 때문에 많은 문제가 발생하게 된다.

상속은 단 하나의 상위 클래스(부모)만 가능하지만, 위임은 여러개의(복수개)의 interface가 가능하다.

public void **addPropertyChangeListener**(PropertyChangeListener listener) {

changeSupport.addPropertyChangeListener(listener);

}

// changeSupport 리스너 목록에 listener을 추가한다.

// null의 경우는 실행되지 않는다.

public void **removePropertyChangeListener**(PropertyChangeListener listener) {

changeSupport.removePropertyChangeListener(listener);

}

// changeSupport 리스너 목록에서 listener을 제거한다. null일 경우 실행되지 않는다.

// getter 메소드 (오브젝트 혹은 변수의 값을 가져옴)

// getter 규칙

- return type은 참조할 멤버변수의 자료형과 일치해야 한다.

- 이름 앞에 get을 붙이고 뒤에는 리턴할 멤버변수의 이름 혹은 해당 변수를 직관적으로 표현하는 단어가 와야한다.

public String **getFirstName**() { return firstName; } // 이름을 반환한다.

public String **getSurname**() { return surname; } // 성을 반환한다.

public String **getNickname**() { return nickname; } // 별명을 반환한다.

public int **getAge**() { return age; } // 나이를 반환한다.

public String **getEmail**() { return email; } // 이메일을 반환한다.

public String **getWeb**() { return getWeb; } // 웹 주소를 반환한다.

public String **getIm**() { return im; } // 메신저 반환

public int **getSex**() { return sex; } // 성별을 반환한다.

public int **getMaritalStatus**() { return maritalStatus; } // 결혼여부를 반환한다.

// setter 메소드 (오브젝트 혹은 변수의 값을 설정함)

// setter 규칙

- return type은 void 혹은 값의 설정 결과를 알려줄 수 있는 type이어야 한다.

- argument는 수정할 멤버변수와 같은 type이어야 한다.

- 이름 앞에 set을 붙이고 뒤에는 수정할 멤버변수의 이름 혹은 해당 변수를 직관적으로 표현하는 단어이어야 한다.

public void **setFirstName**(String firstName) {

String oldFirstName = this.firstName; // 문자열 oldFirstName에 현재 저장된 이름을 저장한다.

this.firstName = firstName; // 입력받은 이름을 저장한다.

changeSupport.firePropertyChange("firstName", oldFirstName, firstName);

// 변경 가능한 프로퍼티 “firstName”에 예전 이름과 현재 이름을 넣어 정의한다.

}

public void **setSurname**(String surname) {

String oldSurname = this.surname; // 문자열 oldSurname에 현재 저장된 성을 저장한다.

this.surname = surname; // 입력받은 성을 저장한다.

changeSupport.firePropertyChange("surname", oldSurname, surname);

// 변경 가능한 프로퍼티 “surname”에 예전 성과 현재 성을 넣어 정의한다.

}

public void **setNickname**(String nickname) {

String oldNickname = this.nickname; // 문자열 oldNickname에 현재 저장된 별명을 저장한다.

this.nickname = nickname; // 입력받은 별명을 저장한다.

changeSupport.firePropertyChange("nickname", oldNickname, nickname);

// 변경 가능한 프로퍼티 “nickname”에 예전 별명과 현재 별명을 넣어 정의한다.

}

public void **setAge**(int age) {

int oldAge = this.age; // 문자열 oldAge에 현재 저장된 나이을 저장한다.

this.age = age; // 입력받은 나이를 저장한다.

changeSupport.firePropertyChange("age", oldAge, age);

// 변경 가능한 프로퍼티 “age”에 예전 나이와 현재 나이를 넣어 정의한다.

}

public void **setEmail**(String email) {

String oldEmail = this.email; // 문자열 oldEmail에 현재 저장된 이메일을 저장한다.

this.email = email; // 입력받은 이메일을 저장한다.

changeSupport.firePropertyChange("email", oldEmail, email);

// 변경 가능한 프로퍼티 “email”에 예전 이메일과 현재 이메일을 넣어 정의한다.

}

public void **setWeb**(String web) {

String oldWeb = this.web; // 문자열 oldWeb에 현재 저장된 웹 주소를 저장한다.

this.web = web; // 입력받은 웹 주소를 저장한다.

changeSupport.firePropertyChange("web", oldWeb, web);

// 변경 가능한 프로퍼티 “web”에 예전 웹 주소와 현재 웹 주소를 넣어 정의한다.

}

public void **setIm**(String im) {

String oldIm = this.im; // 문자열 oldIm에 현재 저장된 연락처를 저장한다.

this.im = im; // 입력받은 연락처를 저장한다.

changeSupport.firePropertyChange("im", oldIm, im);

// 변경 가능한 프로퍼티 “im”에 예전 연락처와 현재 연락처를 넣어 정의한다.

}

public void **setSex**(int sex) {

int oldSex = this.sex; // 문자열 oldSex에 현재 저장된 성별을 저장한다.

this.sex = sex; // 입력받은 성별을 저장한다.

changeSupport.firePropertyChange("sex", oldSex, sex);

// 변경 가능한 프로퍼티 “sex”에 예전 성별과 현재 성별을 넣어 정의한다.

}

public void **setMaritalStatus**(int maritalStatus) {

int oldMaritalStatus = this.maritalStatus;

// 문자열 oldMaritalStatus에 현재 저장된 결혼여부를 저장한다.

this.maritalStatus = maritalStatus; // 입력받은 결혼여부를 저장한다.

changeSupport.firePropertyChange("maritalStatus", oldMaritalStatus, maritalStatus);

// 변경 가능한 프로퍼티 “maritalStatus”에

// 예전 결혼 여부와 현재 결혼 여부를 넣어 정의한다.

}

// Client 생성을 테스트 해보는 메소드.

public static Client **createTestClient**() {

Client client = new Client();

client.setFirstName("George"); // 세터 setFirstName에 “George”를 전달한다.

client.setSurname("Foo"); // 세터 setSurname에 “Foo”를 전달한다.

client.setNickname("Juraj"); // 세터 setNickname에 “Juraj”를 전달한다.

client.setAge(30); // 세터 setAge에 30을 전달한다.

client.setEmail("g.foo@foo.org"); // 세터 setEmail에 [g.foo@foo.org](mailto:g.foo@foo.org)를 전달한다.

client.setWeb("https://beansbinding.dev.java.net"); // 세터 setWeb에 다음 주소를 전달한다.

client.setIm("ICQ: 53 25 89 76"); // 세터 setIm에 다음 연락처 정보를 전달한다.

client.setSex(1); // 세터 setSex에 1을 전달한다.(남성)

client.setMaritalStatus(2); // 세터 setMaritalStatus에 2를 전달한다. (separated)

return client; // 위 세터들을 통해 저장된 client를 반환한다.

}

}

**LoggingBindingListener.java**

package clienteditor;

import javax.swing.JLabel;

import org.jdesktop.beansbinding.AbstractBindingListener;

import org.jdesktop.beansbinding.Binding;

import org.jdesktop.beansbinding.Binding.SyncFailure;

//바인딩이란 컴퓨터 프로그래밍에서 각종 값들이 확정되어 더 이상 변경할 수 없는 구속(Bind)상태가 되는 것. 프로그램 내에서 변수, 배열, 라벨, 절차 등의 명칭, 즉 식별자(Identifier)가 그 대상인 메모리 주소, 데이터형 또는 실제 값으로 배정되는 것이 이에 해당되며, 원시 프로그램의 컴파일링 또는 링크 시에 확정되는 바인딩을 정적 바인딩(static binding)이라 하고, 프로그램의 실행되는 과정에서 바인딩 되는 것을 동적 바인당(dynamic binding)이라고 한다.

// LoggingBindingListener.java는 동기화 이벤트를 기록하는 데 사용하며, 동기화 실패에 대한 경고를 출력하는 클래스이다.

public class **LoggingBindingListener** extends AbstractBindingListener {

// AbstractBindingListener을 상속받는다.

private JLabel outputLabel;

Label 생성(경고 표시하는데 사용)

**LoggingBindingListener**(JLabel outputLabel) { // 생성자

if (outputLabel == null) throw new IllegalArgumentException();

// outputLabel이 null일 경우 에러 표시

this.outputLabel = outputLabel;

// this.outputLabel을 저장한다.

}

// Override : 슈퍼 클래스에 존재하는 필드나 메소드를 서브 클래스에서 재정의하여 사용

@Override

public void **syncFailed**(Binding binding, SyncFailure fail) { // 동기화 실패시

String description;

if ((fail != null) && (fail.getType() == Binding.SyncFailureType.VALIDATION\_FAILED)) {

//fail의 값이 null이 아니거나(fail일 경우) Binding의 타입의 유효성이 Fail인 경우

description = fail.getValidationResult().getDescription();

// description에 fail의 getValidationResult()메서드의 getDescription()의 값을 넣는다.

* 데이터 원본에 대한 설명을 반환

} else {

description = "Sync failed!";

}

String msg = "[" + binding.getName() + "] " + description;

System.out.println(msg);

outputLabel.setText(msg); // 동기화 실패 메세지 출력

}

@Override

public void **synced**(Binding binding) { // 동기화 성공시

String bindName = binding.getName();

String msg = "[" + bindName + "] Synced";

System.out.println(msg); // 동기화 성공 메세지 출력

outputLabel.setText(""); // outputLabel의 텍스트를 초기화

}

}

참고 : <https://doc.formdev.com/beansbinding/org/jdesktop/beansbinding/Binding.SyncFailureType.html>

https://docs.microsoft.com/ko-kr/dotnet/api/system.componentmodel.dataannotations.validationattribute.getvalidationresult?view=net-6.0

**AgeConverter.java**

package clienteditor;

import org.jdesktop.beansbinding.Converter;

// 바인딩에서 사용되는 데이터로 변환을 적용

// 여기선 나이의 값을 변환한다.

public class **AgeConverter** extends Converter<Integer, String> {

// Converter<Integer, String>을 상속받는다. – 부모의 메소드를 그대로 직접 사용(오버라이딩 필요 X)

public String **convertForward**(Integer arg) { // int의 값을 String으로 변환하여 줌

return String.valueOf(arg);

}

public Integer **convertReverse**(String arg) { // String의 값을 int로 변환하여 줌

int value;

try {

value = (arg == null) ? 0 : Integer.parseInt(arg);

// value가 null일 경우 value의 값을 0으로 설정하고 null이 아닐 경우 int형으로 변환

} catch (NumberFormatException ex) {

value = 0; // 위 try에서 에러가 날 경우 value의 값은 0으로 지정

}

return value;

}

}

**AgeValidator.java**

package clienteditor;

import org.jdesktop.beansbinding.Validator;

// 유효성 검사 (어떤 타입의 객체를 검증할 때 사용)

// 여기서는 나이의 값을 검증한다.

public class **AgeValidator** extends Validator<Integer> {

// Validator<Integer>를 상속받는다.

public Validator.Result **validate**(Integer arg) { // arg의 값을 검증(유효성 검사)

if ((arg < 1) || (arg > 199)) { // 받아온 arg의 값이 1보다 작고 199보다 클 경우

return new Result(null, "Age range is 1-199"); // 1과 199 사이를 입력하라는 메세지 반환

}

return null; // arg의 값이 유효하다면 null값을 반환.

}

}

**EmailValidator.java**

package clienteditor;

import org.jdesktop.beansbinding.Validator;

// 입력한 Email을 검증한다.

public class **EmailValidator** extends Validator<String> {

public Validator.Result **validate**(String arg) { // 이메일로 사용할 문자열을 받아온다.

if ((arg.length() < 4) || !arg.contains("@") || !arg.contains(".")) {

return new Result(null, "Please enter a valid email");

}

// 이메일 문자열의 길이가 4보다 작거나, arg 문자열 내에 “@”와 “.”가 없을 경우 정확한 이메일을 입력해달라는 메세지를 반환한다.

return null;

// Email 형식에 맞으면 null값을 반환한다.

}

}

**MaritalStatusConverter.java**

package clienteditor;

import org.jdesktop.beansbinding.Converter;

// 데이터를 변환할 때 사용

// 여기서는 결혼여부의 값을 변환함.

public class **MaritalStatusConverter** extends Converter<Integer, String> {

public String **convertForward**(Integer arg) { // 입력받은 숫자를 받아온다.

String value = null; // value의 초기값을 null으로 설정한다.

switch (arg) {

case 0: value="Single"; break;

입력값이 0일 경우 value=“Single”

case 1: value="Married"; break;

입력값이 1일 경우 value=”Married”

case 2: value="Separated"; break;

입력값이 2일 경우 value=”Separated”

case 3: value="Divorced"; break;

입력값이 3일 경우 value=”Divorced”

}

return value;

value의 값을 반환한다.

}

public Integer **convertReverse**(String arg) {

int value = 0; // value의 초기값을 0으로 설정한다.

if ("Single".equals(arg)) { // 입력받은 값이 “Single”이랑 같을 경우

value = 0;

} else if ("Married".equals(arg)) { // 입력받은 값이 “Married”일 경우

value = 1;

} else if ("Separated".equals(arg)) { // 입력받은 값이 “Separated”일 경우

value = 2;

} else if ("Divorced".equals(arg)) { // 입력받은 값이 “Divorced”일 경우

value = 3;

}

return value;

}

}

**RequiredStringValidator.java**

// Required 검증

package clienteditor;

import org.jdesktop.beansbinding.Validator;

public class **RequiredStringValidator** extends Validator<String> {

// Validator<String>을 상속받는다.

public Validator.Result **validate**(String arg) {

if ((arg == null) || (arg.length() == 0)) { // 받아온 문자열이 null이거나 길이가 0일 경우

return new Validator.Result(null, "Empty value"); // 값이 비었다는 메세지를 반환한다.

}

return null; // 문자열 arg의 검증 결과 문제가 없을 시 null을 반환한다.

}

}