

## Mediator Pattern

### 상대는 하나뿐

# 01. Mediator 패턴

---

- ❑ 멤버 10명이 서로 공동 작업을 하는데, 서로 각자 의견을 교환하고 지시를 한다면 대혼란이 발생한다.
  - 중간에 의견을 조정하는 사람을 두면 좋겠다.
- ❑ Mediator 패턴
  - '조정자', '중재자'라는 뜻
  - '카운셀러'라고 생각하면 좋다
    - 모든 멤버(colleague)는 '카운셀러'에게 보고한다.
    - '카운셀러'는 각 멤버가 올린 보고를 토대로 대국적인 지시를 내린다.
    - 모든 멤버는 이 지시에 따른다.

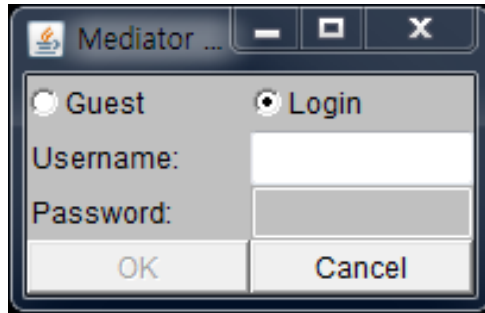
## 02. 예제 프로그램

---

### □ GUI 애플리케이션

- '이름과 패스워드를 입력하는 로그인 다이얼로그'
- Guest/Login 선택과 여러 상황에 따라
  - Username과 Password 입력란의 활성화 여부가 결정됨
  - OK/Cancel 버튼 활성화 여부가 결정됨

## 02. 예제 프로그램



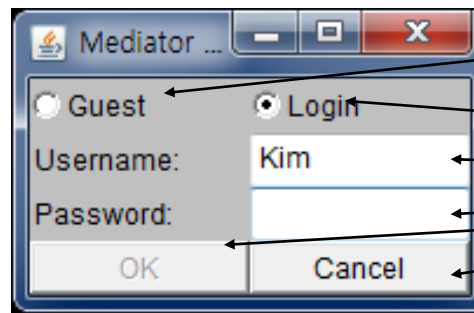
Mediator ...

☐ Guest ☒ Login

Username:

Password:

OK Cancel



Mediator ...

☐ Guest ☒ Login

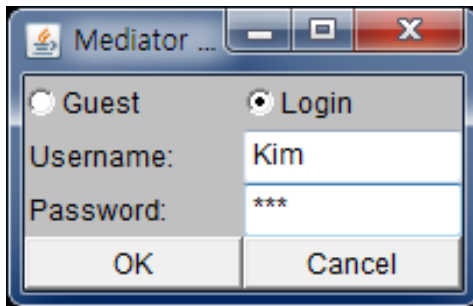
Username:

Password:

OK Cancel

Mediator

중재한다.



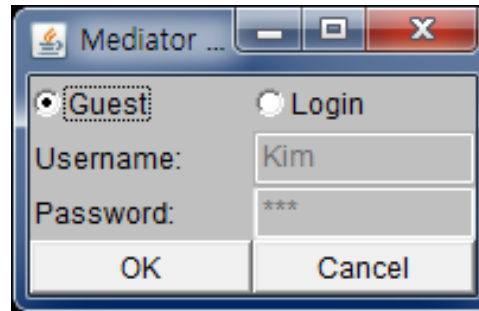
Mediator ...

☐ Guest ☒ Login

Username:

Password:

OK Cancel



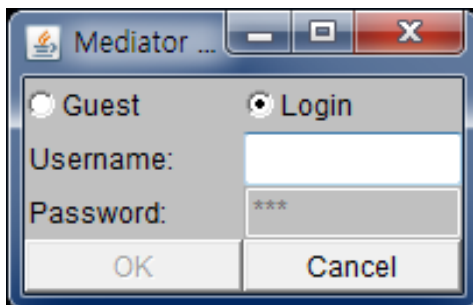
Mediator ...

☒ Guest ☐ Login

Username:

Password:

OK Cancel



Mediator ...

☒ Guest ☐ Login

Username:

Password:

OK Cancel

## 02. 예제 프로그램

---

- ❑ 이러한 프로그램을 어떻게 만들 것인가?
  - 라디오 버튼, 텍스트 필드, 버튼은 각각 다른 클래스로 되어 있다.
  - 앞에서의 유/무효 결정 로직을 각 클래스에 분산시키면 코딩하기가 힘들어 진다.
    - 왜냐하면, 각각의 객체가 서로 관련되어 있어, 서로가 서로를 컨트롤하는 상황에 빠져 버리기 때문이다.
- ❑ 이와 같이, 다수의 객체를 조정해야 하는 경우, Mediator 패턴을 사용한다.
  - 각 컨트롤의 로직을 중재자(카운셀러) 안에만 기술한다.

## 02. 예제 프로그램

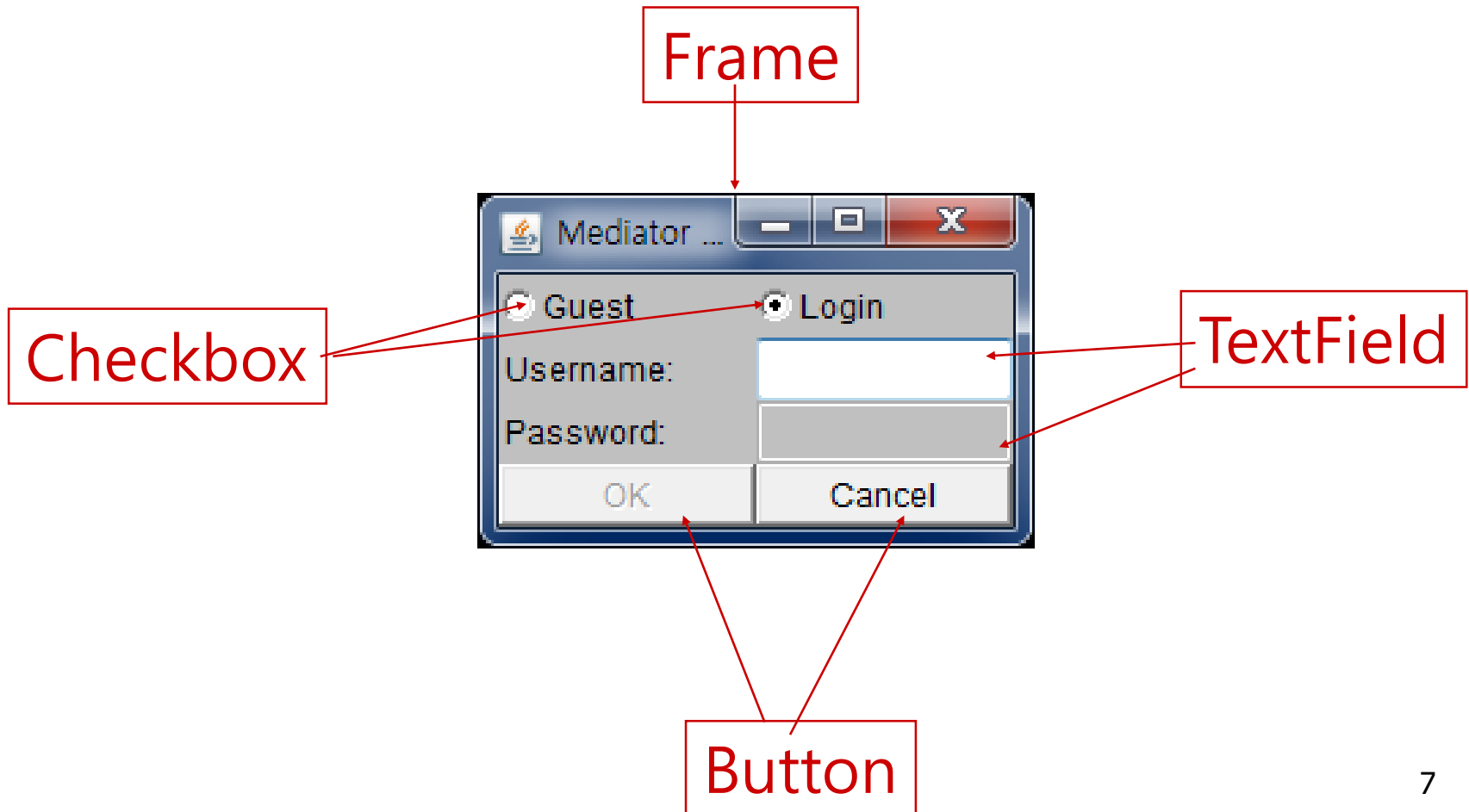
---

이름	해설
Mediator	'카운셀러'의 인터페이스(API)를 정하는 인터페이스
Colleague	'멤버'의 인터페이스(API)를 정하는 인터페이스
ColleagueButton	Colleague 인터페이스를 구현. 버튼을 나타내는 클래스
ColleagueTextField	Colleague 인터페이스를 구현. 텍스트를 입력하는 클래스
ColleagueCheckbox	Colleague 인터페이스를 구현. 체크박스(여기서는 라디오버튼)를 나타내는 클래스
LoginFrame	Mediator 인터페이스를 구현. 로그인 다이얼로그를 나타내는 클래스
Main	동작 테스트용 클래스

## 02. 예제 프로그램

---

- AWT(Abstract Window Toolkit) 객체를 이용한 GUI
  - java.awt 패키지에 있다.



## 02. 예제 프로그램

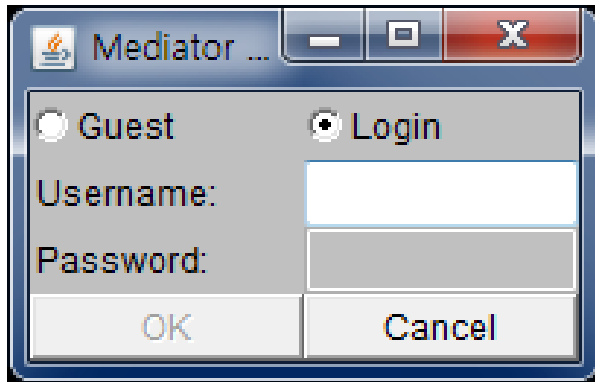
---

- 이벤트 리스너(Event Listener)
  - GUI 컴포넌트에 사용자 액션이 일어나면 이벤트가 발생한다.
  - 이벤트가 발생하면, 등록되어 있던 이벤트 리스너의 메소드가 실행된다.



## 02. 예제 프로그램

– 예: 버튼의 경우



1. 사용자가 버튼을 누르면, `ActionEvent`가 발생한다.
2. 등록되어 있던 `ActionListener`의 `actionPerformed(ActionEvent e)` 메소드가 실행된다.

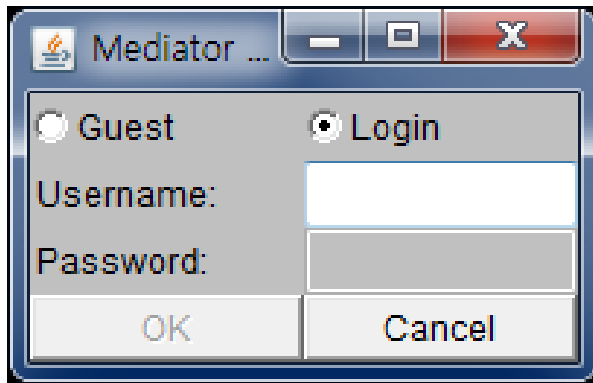
<<interface>>  
`ActionListener`

`actionPerformed(ActionEvent e)`

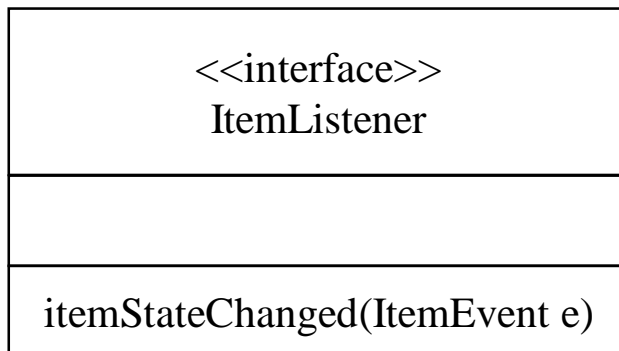
`ActionListener`가 되려면, `ActionListener` 인터페이스를 구현한다.

## 02. 예제 프로그램

– 예: 체크 박스의 경우



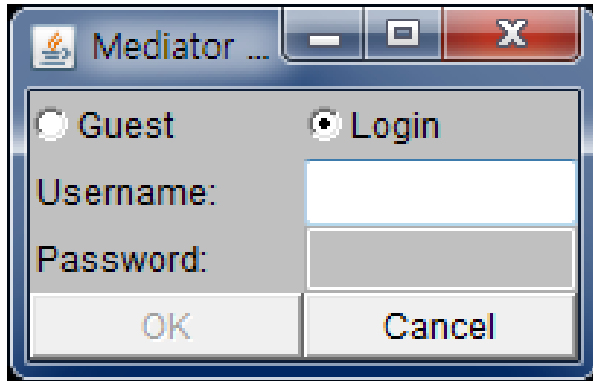
1. 사용자가 체크박스들 중 하나를 선택하면, `ItemEvent`가 발생한다.
2. 등록되어 있던 `ItemListener`의 `itemStateChanged(ItemEvent e)` 메소드가 실행된다.



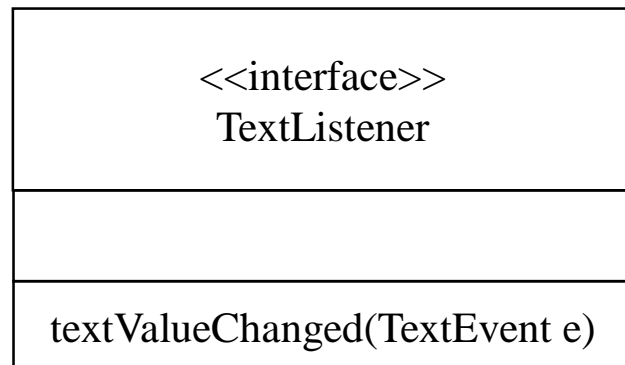
`ItemListener`가 되려면, `ItemListener` 인터페이스를 구현한다.

## 02. 예제 프로그램

– 예: 텍스트 필드의 경우



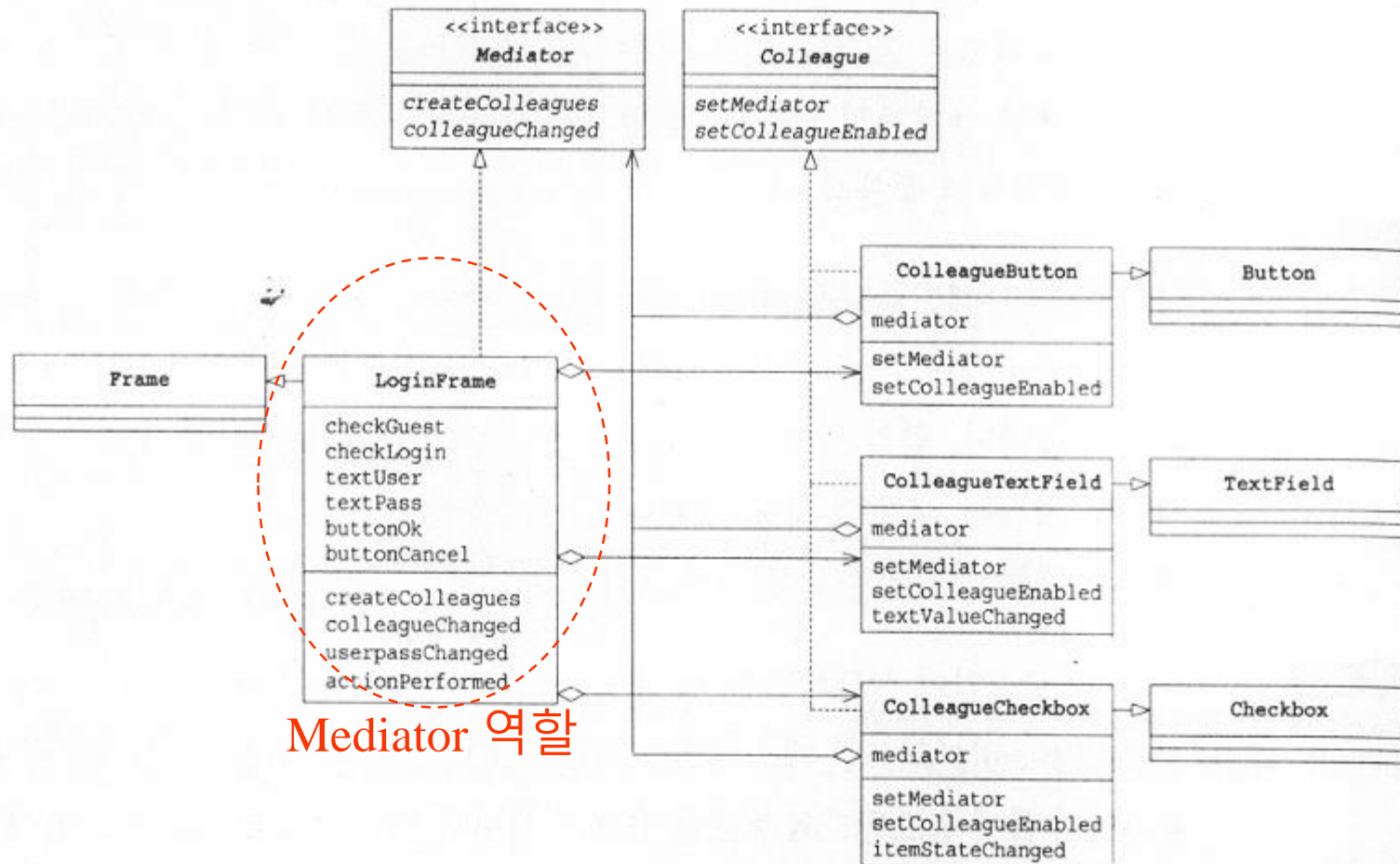
1. 사용자가 문자 입력을 시작하면, `TextEvent`가 발생한다.
2. 등록되어 있던 `TextListener`의 `textValueChanged(TextEvent e)` 메소드가 실행된다.



`TextListener`가 되려면, `TextListener` 인터페이스를 구현한다.

## 02. 예제 프로그램

### 클래스 다이어그램



## 02. 예제 프로그램

### 시퀀스 다이어그램



## 02. 예제 프로그램

---

### □ Mediator 인터페이스

- '카운셀러'를 표현하는 인터페이스
- createColleagues( )
  - mediator가 관리하는 멤버를 생성하는 메소드
- colleagueChanged(Colleague colleague)
  - 각 멤버가 '상담'을 요청할 때 호출하는 메소드
  - 라디오 버튼이나 텍스트 필드의 상태가 변화했을 때, 이 메소드가 호출된다.

## 02. 예제 프로그램

---

### □ Colleague 인터페이스

- '카운셀러'에게 상담하러 오는 멤버를 나타내는 인터페이스
- 구체적인 멤버(ColleagueButton, ColleagueTextField, ColleagueCheckbox)는 이 인터페이스를 구현한다.
- setMediator(Mediator mediator)
  - Mediator 인터페이스를 구현한 LoginFrame 클래스가 첫번째로 호출하는 메소드
  - "내가 카운셀러니까 기억해두세요"라는 의미
- setColleagueEnabled(boolean enabled)
  - "카운셀러가 내리는 지시"에 해당함
  - 멤버의 상태를 "유효" 또는 "무효"로 바꾸는 일을 수행한다.
    - enabled가 true이면, 유효 상태로 바뀌고,
    - enabled가 false이면, 무효 상태로 바뀐다.

## 02. 예제 프로그램

---

### ❑ ColleagueButton 클래스

- java.awt.Button의 하위 클래스
- setMediator(Mediator mediator)
  - 입력 인자로 들어온 Mediator를 멤버 변수인 mediator에 할당함
- setColleagueEnabled(boolean enabled)
  - Button 클래스에서 물려받은 setEnabled(boolean)를 호출하여 유효/무효를 설정한다.



## 02. 예제 프로그램

---

### ❑ ColleagueTextField 클래스

- java.awt.TextField의 하위 클래스
- java.awt.event.TextListener 인터페이스도 구현함
  - 텍스트 필드의 내용이 바뀌었을 때, TextEvent를 처리하는 textValueChanged 메소드를 제공한다.
- setMediator(Mediator mediator)
  - 입력 인자로 들어온 Mediator를 멤버 변수인 mediator에 할당함
- setColleagueEnabled(boolean enabled)
  - TextField 클래스에서 물려받은 setEnabled(boolean)를 호출하여 유효/무효를 설정한다.
  - 또한, setBackground 메소드를 이용하여 유효일때는 백색, 무효일때는 회색으로 바꾼다.

## 02. 예제 프로그램

---

- ColleagueTextField 클래스(계속)
  - textValueChanged(TextEvent e)
    - TextField의 내용이 바뀌었을 때, TextEvent가 발생되고 등록된 TextListener의 이 메소드가 자동으로 호출된다.
    - 카운셀러의 colleagueChanged 메소드를 호출한다.

## 02. 예제 프로그램

---

### ❑ ColleagueCheckbox 클래스

- java.awt.Checkbox 클래스의 하위 클래스
- CheckboxGroup을 이용하여 라디오 버튼들을 정의한다.
- java.awt.event.ItemListener 인터페이스도 구현함
  - 라디오 버튼의 상태변화를 itemStateChanged 메소드에서 캐치한다.
- itemStateChanged(ItemEvent e)
  - 라디오버튼의 상태가 바뀌었을 때, ItemEvent가 발생되고, 등록된 ItemListener의 이 메소드가 자동으로 호출된다.

## 02. 예제 프로그램

---

### □ LoginFrame 클래스

- java.awt.Frame 클래스의 하위 클래스
  - GUI 애플리케이션을 만들기 위한 클래스
- 카운셀러(중재자)의 역할을 수행한다.
- 생성자에서 하는 일
  - 배경색의 설정(setBackground( ) 이용)
  - 레이아웃 매니저의 설정(setLayout( ) 이용)
    - 프레임을 구성하는 구성 요소의 배치를 결정한다.
    - GridLayout(4,2) : 프레임 영역을 4행 2열로 나눔
  - createColleagues 메소드에서 멤버들을 생성한다.
  - colleague들을 배치한다 (add( ) 이용)
  - checkGuest의 초기 상태에 따라 프레임의 초기 상태를 설정한다.
  - colleague 들을 보여준다.
    - pack( ) : 윈도우를 컴포넌트를 고려하여 적합한 크기로 조절한다.
    - show( ) : 윈도우를 보여준다.

## 02. 예제 프로그램

---

### ❑ LoginFrame 클래스(계속)

- createColleagues( )
  - 각 colleague를 생성하고,
  - Mediator를 설정한 후,
  - 각 colleague에 해당 리스너를 연결한다.
- colleagueChanged(Colleague c)
  - colleague의 상태가 변화했을 때, 호출되는 메소드
  - colleague의 상태 변화에 따라 해당 colleague의 상태를 어떻게 변화시킬 것인가에 대한 로직을 가지고 있다.
- userpassChanged( )
  - TextField 상태 변화에 따라 버튼 활성화/비활성화
- actionPerformed(ActionEvent e)
  - ActionListener가 되기위해서 구현하는 메소드
  - OK, Cancel 버튼이 눌러졌을 때 실행됨

## 02. 예제 프로그램

---

- Main 클래스
  - LoginFrame의 인스턴스를 생성함

## 03. 등장 역할

---

- Mediator(조정자, 중개자)의 역할
  - Colleague 역할들과의 통신을 조정하기 위한 인터페이스를 정의하는 역할
  - 예제에서는 Mediator 인터페이스가 해당됨
  
- ConcreteMediator(구체적인 조정자, 중개자)의 역할
  - Mediator 역할의 인터페이스(API)를 구현하여, 실제의 조정을 수행하는 역할
  - 예제에서는 LoginFrame 클래스가 해당됨

## 03. 등장 역할

---

- ❑ Colleague(동료)의 역할
  - Mediator 역할과 통신을 할 인터페이스(API)를 정함
  - 예제에서는 Colleague 인터페이스가 해당됨
  
- ❑ ConcreteColleague(구체적인 동료)의 역할
  - Colleague 역할의 인터페이스(API)를 구현함
  - 예제에서는, ColleagueButton, ColleagueTextField, ColleagueCheckbox 가 해당됨



## 04. 독자의 사고를 넓혀주는 힌트

---

### □ 분산이 재앙이 되는 경우

- ColleagueButton, ColleagueTextField, ColleagueCheckbox 들 사이에 상태 변화 로직이 Mediator인 LoginFrame에 집중되어 있다.
- 이 로직이, 각각의 Colleague에 분산되어 있다면 디버그나 이해도 측면에서 좋지 않을 것이다.
- 객체지향에서는 한군데 집중하는 것을 피해 분산 처리하는 일, 즉 "divide and conquer"가 많이 사용된다.
- 이 경우는, 반대로 한군데에 로직을 집중시킨다.
- 분산 시킬 것은 분산시키고, 집중시킬 것은 집중시키는 것이 중요하다.

## 04. 독자의 사고를 넓혀주는 힌트

---

- 재사용할 수 있는 것은 무엇인가?
  - ConcreteColleague 역할을 하는 ColleagueButton, ColleagueTextField, ColleagueCheckbox는 다른 대화창에서도 재사용 가능하다
    - 이유: 특정 어플리케이션 관련 로직을 이 클래스들이 포함하고 있지 않기 때문에
  - ConcreteMediator 역할을 하는 LoginFrame 클래스는 재사용하기 힘들다
    - 이유: 특정 어플리케이션 관련 로직을 포함하고 있기 때문에

## 05. 관련 패턴

---

### ❑ Facade 패턴

- Mediator의 역할은 Colleague 역할의 중개자로서 주고받기를 수행함 → 양방향
- Façade 역할은 일방적으로 다른 역할을 이용해서 높은 레벨의 인터페이스(API)를 만듦 → 단방향

### ❑ Observer 패턴

- Mediator 역할과 Colleague 역할의 통신은 Observer 패턴을 사용해서 수행되는 경우가 있음

## 06. 요약

---

- ❑ 믿음직스러운 카운셀러가, 복잡하게 얽혀있는 객체들 간의 상호 통신을 중재시키고, 객체들 간의 상호 작용 로직을 Mediator에 집중시킴으로써 처리를 원할하게 한다.
- ❑ 특히, GUI 어플리케이션에서 효과적이다.