JAVA 프로그래밍

Chapter 16 JavaFX

목차

JavaFX 개요

- JavaFX는 크로스 플랫폼에서 실행하는 리치 클라이언트 애플리케이션(Rich Client Application)을 개발하기 위한 그래픽과 미디어 패키지를 말한다.

- AWT(Abstract Window Toolkit)
  - 운영체제가 제공하는 네이티브(native) UI 컴포넌트를 이용해서 개발
- Swing
  - 네이티브 UI 컴포넌트를 사용하지 않고 자신만의 UI(look and feel)를 갖도록 개발.
  - 메모리를 많이 사용하고 실행 성능이 느림.
- JavaFX
  - JavaFX는 플래쉬(flash)와 실버라이트(silverlight)의 대항마로 만들어 짐.
  - 화면 레이아웃(FXML)과 스타일(CSS), 애플리케이션 로직을 분리하여 개발이 가능.
  - 표준GUI 라이브러리였던 Swing을 대체.
  - 데스크탑 및 웹 브라우저에서 실행 가능.
  - JavaFX 애플리케이션에서 UI생성, 이벤트 처리, 멀티미디어 재생, 웹 뷰 등과 같은 기능은 JavaFX API로 개발하고 그 이외의 기능은 자바 표준 API를 활용해서 개발

JavaFX 애플리케이션 개발 시작

❖ 메인 클래스

- JavaFX 애플리케이션을 시작하는 메인 클래스는 추상 클래스인 javafx.application.Application을 상속받고, start() 메서드를 재정의 해야 한다.
- main() 메서드는 Application의 launch() 메서드를 호출한다.
- launch() 메서드는 메인 클래스의 객체를 생성하고, 메인 윈도우를 생성한 다음 start() 메서드 호출하고, 메인 윈도우를 start()의 primaryStage 매개값으로 제공.

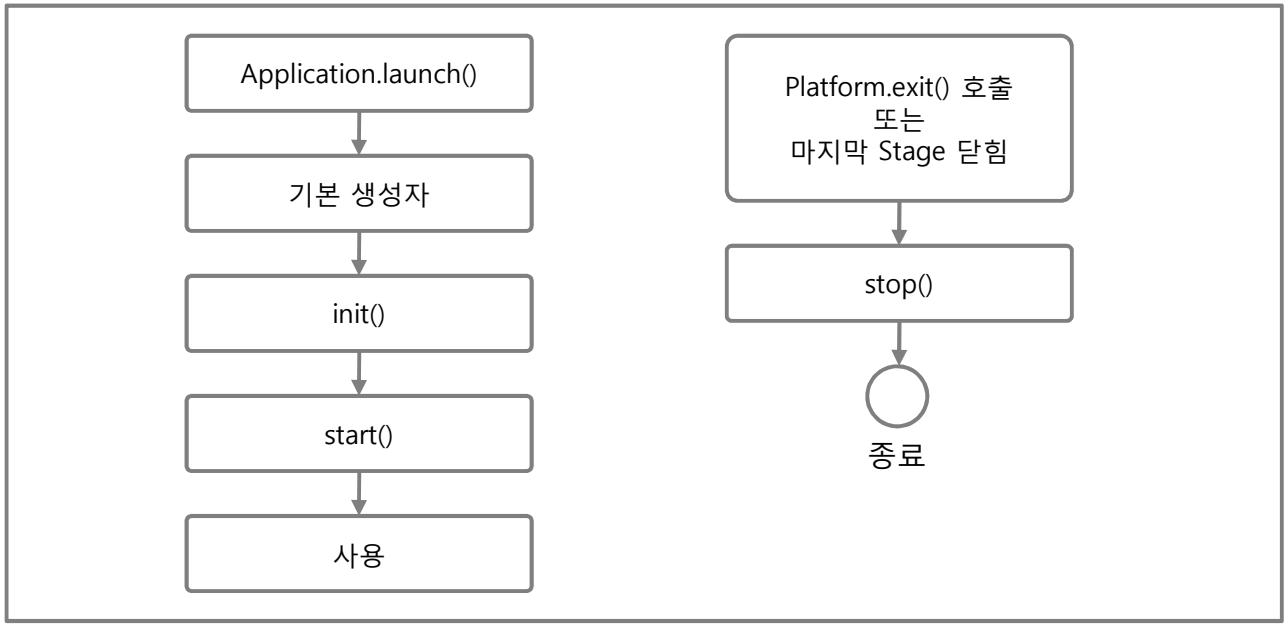
```
public class AppMain extends Application{
  @Override
  public void start(Stage primaryStage) throws Exception{
    primaryStage.show();    // 윈도우 보여주기
  }

  public static void main(String[] args){
    launch(args);           // AppMain 객체 생성 및 메인 윈도우 생성
  }
}
```

JavaFX 애플리케이션 개발 시작

❖ JavaFX 라이프사이클

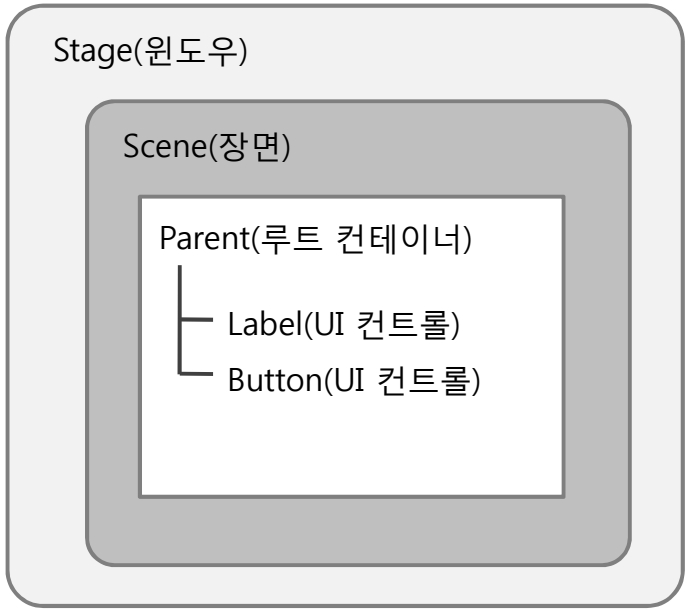
- init() : 메인 클래스의 실행 매개값을 얻어 어플에서 이용할 수 있게 해줌.
- start() : 메인 윈도우 실행
- JavaFX-Launcher 스레드 : init() 실행
- JavaFX Application Thread 스레드 : 메인 클래스 기본 생성자, start() 및 stop() 실행



JavaFX 애플리케이션 개발 시작

❖ 무대(Stage)와 장면(Scene)

- 무대(Stage)는 한 번에 하나의 장면(Scene)을 가짐.
- 장면(Scene)은 무대를 구성할 다양한 요소(UI 컨트롤 등)들을 가진다.
- Scene scene = new Scene(Parent root); // root = javafx.scene.layout 패키지 컨테이너들
- primaryStage.setScene(scene);



JavaFX 레이아웃

- 프로그램적 레이아웃
  - 자바 코드로 UI 컨트롤을 배치하는 것.
  - 자바 코드로 작성하여 레이아웃이 복잡해지면 난해한 프로그램이 될 수 있음.
  - 디자이너와 협업이 어려움.
- FXML 레이아웃
  - FXML은 XML 기반의 마크업 언어로, 태그를 통해서 UI 레이아웃을 배치하는 것.
  - 애플리케이션의 UI 레이아웃을 자바 코드에서 분리해서 태그로 선언하는 방법을 제공.
  - 개발 후 레이아웃 변경 시 FXML 태그만 수정하면 됨.

JavaFX 레이아웃

❖ FXML 작성 규칙

- FXML로 선언된 태그는 자바 코드로 변환되어 실행되기 때문에 자바 코드와 매핑 관계가 존재한다.

프로그램적 레이아웃 자바 코드	FXML 레이아웃 태그

HBox hbox = new HBox();
hbox.setPadding(new Insets(10, 10, 10, 10));
hbox.setSpacing(10);
	<HBox xmlns:fx="http://javafx.com/fxml">
	<padding>
	<Insets top="10" right="10"
	bottom="10" left="10"/>
	</padding>
	<spacing>10<spacing>
	</HBox>

TextField textField = new TextField(); textField.setPrefWidth(200);	<TextField> <prefWidth>200</prefWidth> </TextField>
Button button = new Button(); button.setText("확인");	<Button> <text>확인</text> </Button>

JavaFX 레이아웃 - FXML 작성 규칙

❖ 패키지 선언

- 자바 코드의 패키지 선언(import 패키지) 과 매핑되는 FMXL 태그는 <?import 패키지?>이다.
- <?import 패키지?> 작성 위치 : <?xml version ="1.0" encoding="UTF-8">과 루트 컨테이너 태그 사이

자바 코드	FXML 태그
`import javafx.scene.layout.HBox;`	`<?import javafx.scene.layout.HBox?>`
`import javafx.scene.control.*;`	`<?import javafx.scene.control.*?>`

JavaFX 레이아웃 - FXML 작성 규칙

❖ 태그 선언

- FXML태그는 <와>사이에 태그 이름을 작성하는 것, 반드시 시작 태그가 있으면 끝 태그도 있다.
- <태그이름> … 태그내용 … </태그이름>
- <태그 이름/> //시작 태그와 끝 태그 사이에 태그 내용이 없을 경우 작성 방법
- 태그 이름은 JavaFX의 클래스명이거나, Setter의 메소드명이 될 수 있다.

자바 코드	FXML 태그

`Button button = new Button();`
`button.setText("확인");`
	`<Button>`
	`  <text>확인</text>`
	`</Button>`

JavaFX 레이아웃 - FXML 작성 규칙

❖ 속성 선언

- FXML 태그는 속성을 가질 수 있다. 속성이란 태그가 가지는 속성값을 가지는 것
- 속성값은 큰 따옴표(") 또는 작은 따옴표(')로 반드시 감싸야 한다.
- <태그이름 속성명 ="값" 속성명='값'> ....</태그이름>
- 속성명은 Setter 메소드명이 오는데, 8가지 기본타입과 String을 세팅하는 Setter만 올 수 있다.

자바 코드	FXML Setter 태그	FXML Setter 속성

`Button button = new Button();`
`button.setText(“확인”);`
	`<Button>`
	`  <text>확인</text>`
	`</Button>`
		`<Button text=“확인”/>`

JavaFX 레이아웃 - FXML 작성 규칙

❖ 객체 선언

- Setter 메소드가 기본 타입과 String 타입이 아닌 다른 타입의 객체를 매개 값으로 갖는다면 속성으로 작성할 수 없고, 태그로 작성해야 한다.

❖ <클래스 속성 ="값"/>

- FXML태그로 기본 생성자를 호출하는 객체 생성 작성 방법
- <클래스> // 의미: new 연산자로 기본 생성자를 호출하는 객체를 생성
- FXML태그로 매개변수가 있는 생성자를 호출하는 객체 생성 2가지 작성 방법

  - 매개 변수에 @NamedArg(javafx.beans.NameArg)어노테이션이 적용되어 있는 경우에 사용.

  1. <클래스 매개변수 ="값">....</클래스>
  2. <클래스>

     <매개변수>값</매개변수>

     </클래스>

JavaFX 레이아웃 - FXML 작성 규칙

❖ <클래스 fx:value="값"/>

- 클래스가 valueOf(String str) 메서드를 제공해서 객체를 생성하는 경우.
- String, Integer, Double, Boolean 등의 Wrapper 클래스.

자바 코드	FXML 태그

`String.valueOf(“Hello, World”);`
`Integer.valueOf(“120”);`
`Double.valueOf(“1.0”);`
`Boolean.valueOf(“false”);`
	`<String fx:value=“Hello, World”/>`
	`<Integer fx:value=“120”/>`
	`<Double fx:value=“1.0”/>`
	`<Boolean fx:value=“false”/>`

❖ <클래스 fx:constant="상수"/>

- 클래스에 정의된 상수 값을 얻은 경우

자바 코드	FXML 태그

`Button button = new Button();`
`button.setMaxWidth(Double.MAX_VALUE);`
	`<Button>`
	`  <maxWidth>`
	`  <Double fx:constant=“MAX_VALUE”/>`
	`  </maxWidth>`
	`</Button>`

JavaFX 레이아웃 - FXML 작성 규칙

❖ <클래스 fx:factory="정적메소드"/>

- 어떤 클래스는 new 연산자로 객체를 생성할 수 없고, 정적 메소드로 객체를 얻어야 하는 경우가 있다.

자바 코드	FXML 태그

`ComboBox combo = new ComboBox();`
`combo.setItems(`
`  FXCollections.observableArrayList(`
`  “공개”, “비공개”`
`  )`
`);`
	`<ComboBox>`
	`  <items>`
	`  <FXCollections`
	`fx:factory=“observableArrayList”>`
	`  <String fx:value=“공개”/>`
	`  <String fx:value=“비공개”/>`
	`  </FXCollections>`
	`  </items>`
	`</ComboBox>`

JavaFX 레이아웃

❖ FXML 로딩과 Scene 생성

- FXML 파일을 읽어드려 선언된 내용을 객체화해야 한다.
- javafx.fxml.FXMLLoader 클래스의 load() 메서드를 가지고 FXML 파일을 로딩 할 수 있다.

```
Parent root = FXMLLoader.load(getClass().getResource("xxx.fxml"));
```

```
FXMLLoader loader
      = new FXMLLoader(getClass().getResource("xxx.fxml"));
Parent root = (Parent)loader.load();
```

JavaFX 컨테이너

- 컨테이너란 컨트롤들을 쉽게 배치할 수 있도록 도와주는 클래스를 말한다.
- javafx.scene.layout 패키지에 다양한 컨테이너 클래스들이 존재

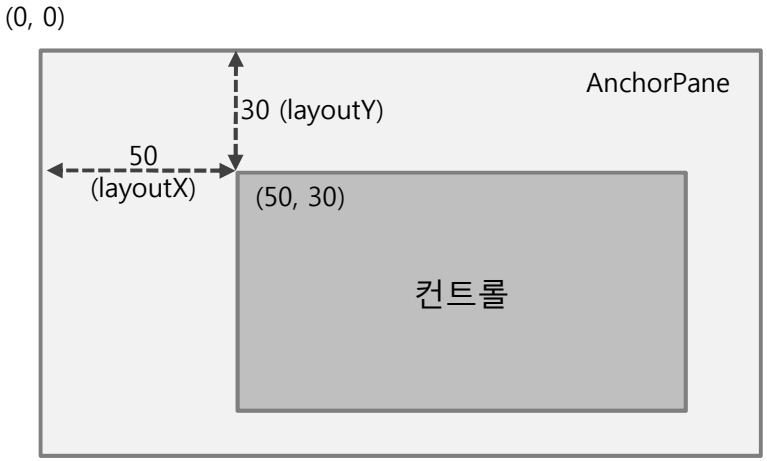
컨테이너	설명
AnchorPane	컨트롤을 좌표로 배치하는 레이아웃
BorderPane	위, 아래, 오른쪽, 왼쪽, 중앙에 컨트롤을 배치하는 레이아웃
FlowPane	행으로 배치하되 공간이 부족하면 새로운 행에 배치하는 레이아웃
GridPane	그리드로 배치하되 셀의 크기가 고정적이지 않은 레이아웃
StackPane	컨트롤을 겹쳐서 배치하는 레이아웃
TilePane	그리드로 배치하되 고정된 셀의 크기를 갖는 레이아웃
HBox	수평으로 배치하는 레이아웃
VBox	수직으로 배치하는 레이아웃

JavaFX 컨테이너

❖ AnchorPane 컨테이너

- 좌표를 이용하여 AnchorPane의 좌상단(0, 0)을 기준으로 컨트롤을 배치한다.

태그 및 속성	설명	적용
prefWidth	폭을 설정	AnchorPane
prefHeight	높이를 설정	AnchorPane
layoutX	컨트롤의 X 좌표	컨트롤
layoutY	컨트롤의 Y 좌표	컨트롤
<children>	컨트롤을 포함	AnchorPane



JavaFX 컨테이너

❖ HBox와 VBox 컨테이너

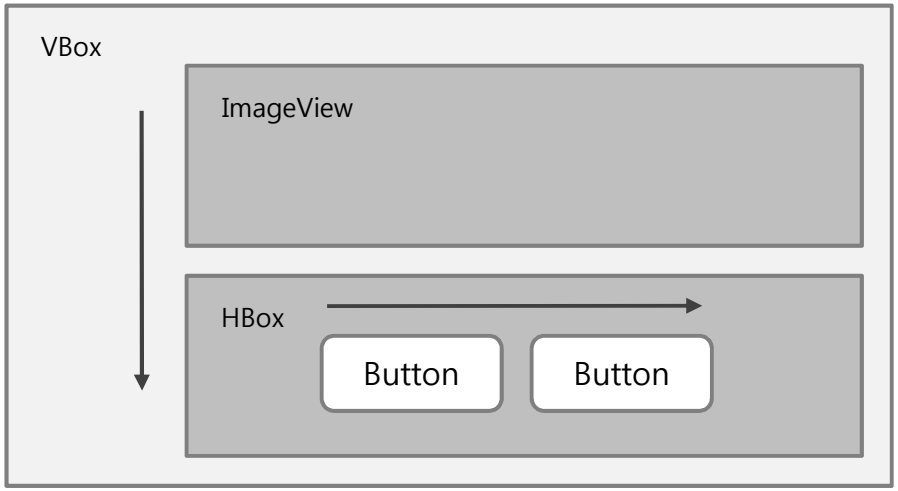
- HBox와 VBox는 수평과 수직으로 컨트롤을 배치한는 컨테이너이다.

태그 및 속성	설명	적용
prefWidth	폭을 설정	HBox, VBox
prefHeight	높이를 설정	HBox, VBox
alignment	컨트롤의 정렬을 설정	HBox, VBox
spacing	컨트롤의 간격을 설정	HBox, VBox
fillWidth	컨트롤의 폭 확장 여부 설정	VBox
fillHeight	컨트롤의 높이 확장 여부 설정	HBox
<children>	컨트롤을 포함	HBox, VBox
<HBox.hgrow> <Priority fx:constant="ALWAYS"/> </HBox.hgrow>	HBox의 남은 폭을 채움	컨트롤
<VBox.vgrow> <Priority fx:constant="ALWAYS"/> </VBox.vgrow>	VBox의 남은 높이를 채움	컨트롤

JavaFX 컨테이너

❖ HBox와 VBox 컨테이너

- HBox와 VBox는 수평과 수직으로 컨트롤을 배치한는 컨테이너이다.

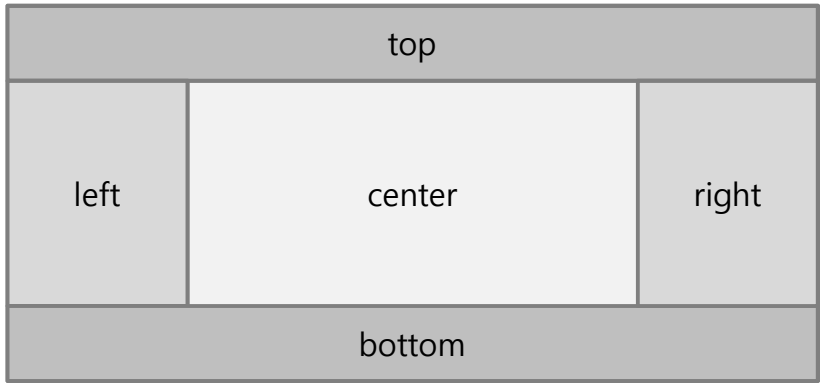


JavaFX 컨테이너

❖ BorderPane 컨테이너

- BorderPane은 top, bottom, left, right, center 셀에 컨트롤을 배치하는 컨테이너이다.

태그 및 속성	설명	적용
prefWidth	폭을 설정	BorderPane
prefHeight	높이를 설정	BorderPane
<top>	top에 배치될 컨트롤을 포함	BorderPane
<bottom>	bottom에 배치될 컨트롤을 포함	BorderPane
<right>	right에 배치될 컨트롤을 포함	BorderPane
<left>	left에 배치될 컨트롤을 포함	BorderPane
	center에 배치될 컨트롤을 포함	BorderPane

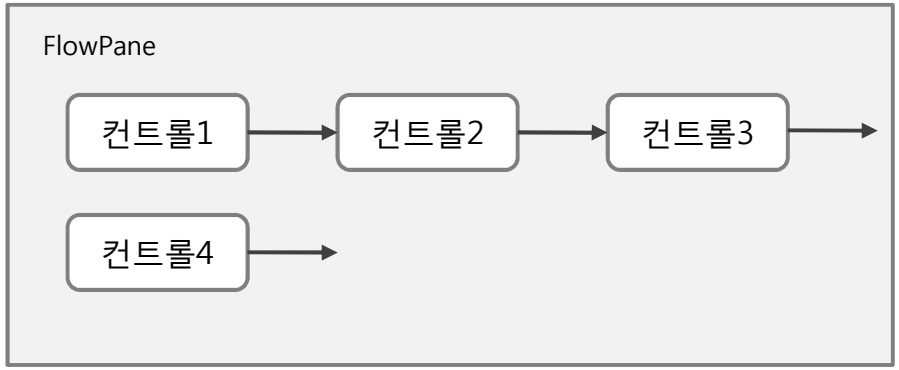


JavaFX 컨테이너

❖ FlowPane 컨테이너

- FlowPane은 행으로 컨트롤을 배치하되 공간이 부족하면 새로운 행에 배치하는 컨테이너.

태그 및 속성	설명	적용
prefWidth	폭을 설정	FlowPane
prefHeight	높이를 설정	FlowPane
hgap	컨트롤의 수평 간격을 설정	FlowPane
vgap	컨트롤의 수직 간격을 설정	FlowPane
<children>	컨트롤을 포함	FlowPane



JavaFX 컨테이너

❖ TilePane 컨테이너

- TilePane은 그리드로 컨트롤을 배치하되 고정된 셀(타일) 크기를 갖는 컨테이너.

태그 및 속성	설명	적용
prefWidth	폭을 설정	TilePane
prefHeight	높이를 설정	TilePane
prefTileWidth	타일의 폭을 설정	TilePane
prefTileHeight	타일의 높이를 설정	TilePane
<children>	컨트롤을 포함	TilePane

JavaFX 컨테이너

❖ GridPane 컨테이너

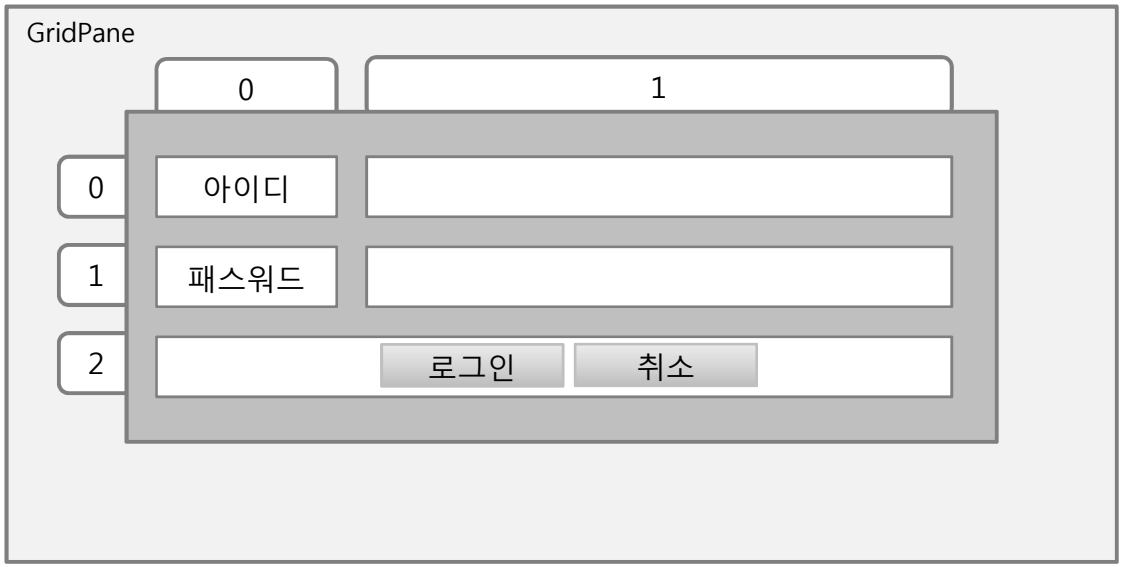
- GridPane은 그리드로 컨트롤을 배치하되 셀의 크기가 고정적이지 않고 유동적인 컨테이너.

태그 및 속성	설명	적용
prefWidth	폭을 설정	GridPane
prefHeight	높이를 설정	GridPane
hgap	컨트롤의 수평 간격을 설정	GridPane
vgap	컨트롤의 수직 간격을 설정	GridPane
<children>	컨트롤을 포함	GridPane
GridPane.rowIndex	컨트롤이 위치하는 행 인덱스를 설정	컨트롤
GridPane.columnIndex	컨트롤이 위치하는 열 인덱스를 설정	컨트롤
GridPane.rowSpan	행 병합 수를 설정	컨트롤
GridPane.columnSpan	열 병합 수를 설정	컨트롤
GridPane.hgrow	수평 빈 공간 채우기를 설정	컨트롤
GridPane.vgrow	수직 빈 공간 채우기를 설정	컨트롤
GridPane.halignment	컨트롤의 수평 정렬을 설정	컨트롤
GridPane.valignment	컨트롤의 수직 정렬을 설정	컨트롤

JavaFX 컨테이너

❖ GridPane 컨테이너

- GridPane은 그리드로 컨트롤을 배치하되 셀의 크기가 고정적이지 않고 유동적인 컨테이너.

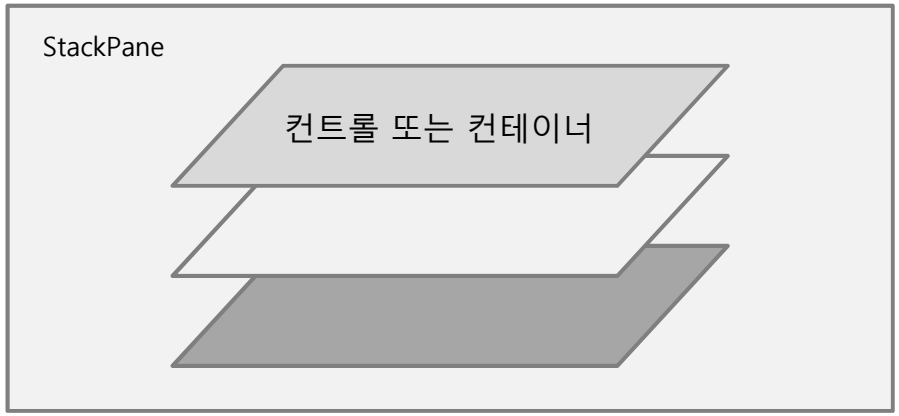


JavaFX 컨테이너

❖ StackPane 컨테이너

- StackPane은 컨트롤을 겹쳐 배치하는 컨테이너로써, 카드 레이아웃이라고 한다.

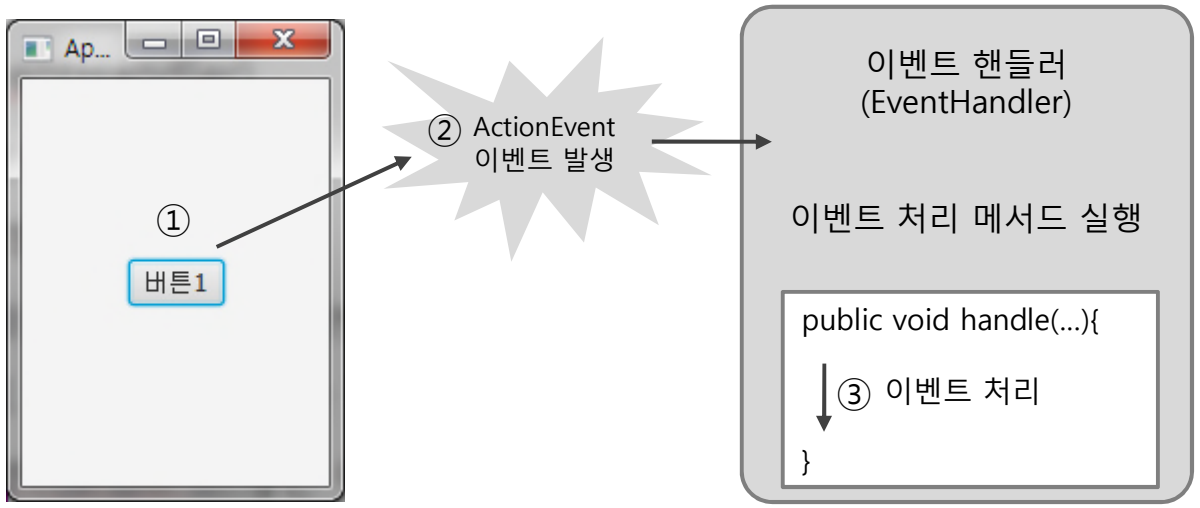
태그 및 속성	설명	적용
prefWidth	폭을 설정	StackPane
prefHeight	높이를 설정	StackPane
<children>	컨트롤을 포함	StackPane



JavaFX 이벤트 처리

❖ 이벤트 핸들러(EventHandler)

- JavaFX는 이벤트 발생 컨트롤과 이벤트 핸들러(EventHandler)를 분리하는 위임형 (Delegation) 방식을 사용한다.
- 위임형 방식이란 컨트롤에서 이벤트가 발생하면, 컨트롤이 직접 처리하지 않고 이벤트 핸들러에게 이벤트 처리를 위임하는 방식이다.
- 컨트롤에 EventHandler 등록은 이벤트 이름에 해당 하는 setOnXXX() 이름을 가진 메서드로 등록한다.



JavaFX 이벤트 처리

- FXML 컨트롤러(Controller)
  - FXML 레이아웃은 FXML 파일당 별도의 컨트롤러(Controller)를 지정해서 이벤트를 처리할 수 있기 때문에 FXML 레이아웃과 이벤트 처리 코드를 완전히 분리할 수 있다.
- fx:controller 속성과 컨트롤러 클래스
  - FXML 파일의 루트 태그에 fx:controller 속성으로 컨트롤러를 지정하면 UI 컨트롤에서 발생하는 이벤트를 컨트롤러가 처리
  - initialize() 오버라이드 : 주로 UI 컨트롤 초기화, 이벤트 핸들러 등록, 속성 감시 등의 코드 작성

```
<루트컨테이너 fx:controller="packageName.ControllerName">
</루트컨테이너>
```

```
public classs ControllerName implements Initializable{
  @Override
  public void initialize(URL location, ResourceBundle resources){}
}
```

JavaFX 이벤트 처리

❖ fx:id 속성과 @FXML 컨트롤 주입

- FXML파일에 포함된 컨테이너 및 컨트롤의 참조하기 위해 fx:id 속성을 가짐
- fx:id 속성을 가진 컨트롤들은 컨트롤러의 @FXML 어노테이션이 적용된 필드에 자동 주입
- fx:id 속성값과 필드명은 동일해야 한다.

```
<HBox … fx:controller="event.Ex02EventRootController">
<children>
   <Button fx:id="btn1" text="버튼1"/>
   <Button fx:id="btn2" text="버튼2"/>
   <!-- #메서드명을 이용한 이벤트 처리 메서드 매핑 방식 -->
   <Button fx:id="btn3" text="버튼3" onAction="#handleBtn3Action"/>
</children>
</HBox>
```

```
public class Ex02EventRootController implements Initializable{

    @FXML private Button btn1;        // btn1 객체 주입
    @FXML private Button btn2;        // btn2 객체 주입
    @FXML private Button btn3;        // btn3 객체 주입
    @Override
    public void initialize(URL location, ResourceBundle resources) {}
}
```

JavaFX 속성 감시와 바인딩

- JavaFX는 컨트롤의 속성(property)을 감시하는 리스너를 설정할 수 있다.
- 속성 감시
  - JavaFX 컨트롤 속성은 세 가지 메서드로 구성된다.
  - Getter와 Setter 그리고 Property 객체를 리턴하는 메서드로 구성.
  - xxxProperty 클래스 : JavaBeans의 프로퍼티 외에 JavaFX에서 이를 확장하여 바인딩과 리스너 개념을 추가함.

```
private StringProperty text = new SimpleStringProperty();  // 값이 저장될 필드
public void setText(String newValue) { text.set(newValue); }  // Setter
public String getText() { return text.get(); }                 // Getter
public StringProperty textProperty() { return text; }    // Property 메서드
```

```
textProperty().addListener(new ChangeListener<String>(){
  @Override
  public void changed(ObservableValue<? extends String> observable,
      String oldValue, String newValue){
  }
});
```

JavaFX 속성 감시와 바인딩

❖ 속성 바인딩

- JavaFX 속성은 다른 속성과 바인딩될 수 있다.
- 바인딩된 속성들은 하나가 변경되면 자동적으로 다른 하나도 변경된다.
- xxxProperty() 메서드가 리턴하는 Property 구현 객체의 bind() 메서드를 이용하면 된다.
- 단방향 바인딩 : bind() 메서드, 바인딩 받는 쪽은 변경 불가함
- 양방향 바인딩 : bindBidirectional()

```
TextArea textArea1;
TextArea textArea2;

// 단방향 바인딩, textArea2는 수정이 불가함.
textArea2.textProperty().bind(textArea1.textProperty());

// 양방향 바인딩
textArea2.textProperty().bindBidirectional(textArea1.textProperty());
Bindings.bindBidirectional(textArea1.textProperty(), textArea2.textProperty());
```

JavaFX 속성 감시와 바인딩

❖ Bindings 클래스

- Bindings의 정적 메서드는 속성을 연산하거나, 다른 타입으로 변환한 후 바인딩하는 기능을 제공한다.

메서드	설명
add, substract, multiply, divide	속성값에 덧셈, 뺄셈, 곱셈, 나눗셈 연산을 수행하고 바인딩 함.
max, min	속성값과 어떤 수를 비교해서 최대, 최소값을 얻고 바인딩 함
greaterThan, greaterThanOrEqual	속성값이 어떤 값보다 큰지, 같거나 큰지를 조사해서 true/false로 변환하여 바인딩 함
lessThan, lessThanOrEqual	속성값이 어떤 값보다 작거나, 같거나 작은지를 조사해서 true/false로 변환하여 바인딩 함
equal, notEquals	속성값이 어떤 값과 같은지, 다른지를 조사해서 true/false로 변환하여 바인딩 함
equalIgnoreCase, notEqualIgnoreCase	대소문자와 상관없이 속성값이 어떤 문자열과 같은지, 다른지를 조사해서 true/false로 변환하여 바인딩 함

JavaFX 속성 감시와 바인딩

❖ Bindings 클래스 (계속)

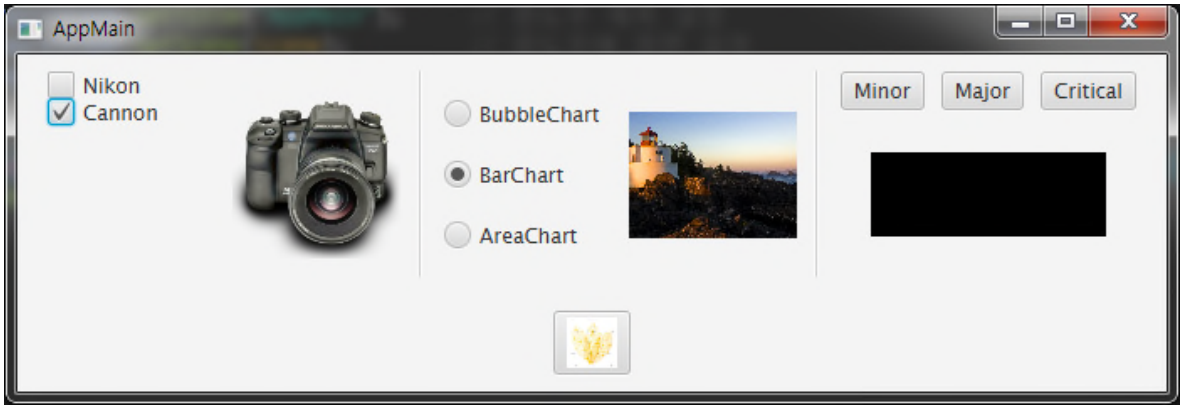
- Bindings의 정적 메서드는 속성을 연산하거나, 다른 타입으로 변환한 후 바인딩하는 기능을 제공한다.

메서드	설명
isEmpty, isNotEmpty	속성값이 비어있는지, 아닌지를 조사해서 true/false로 변환하여 바인딩 함
isNull, isNotNull	속성값이 null 또는 not null 인지를 조사해서 true/false로 변환하여 바인딩 함
length	속성값이 문자열일 경우 문자 수를 얻어 바인딩 함
size	속성 타입이 배열, List, Map, Set 일 경우 요소 수를 얻어 바인딩 함
and, or	속성값이 boolean 일 경우, 논리곱, 논리합을 얻어 바인딩 함
not	속성값이 boolean 일 경우, 반대값으로 바인딩 함
convert	속성값을 문자열로 변환해서 바인딩 함
valueAt	속성이 List, Map일 경우 해당 인덱스 또는 키의 값을 얻어 바인딩 함

JavaFX 컨트롤

❖ 버튼 컨트롤

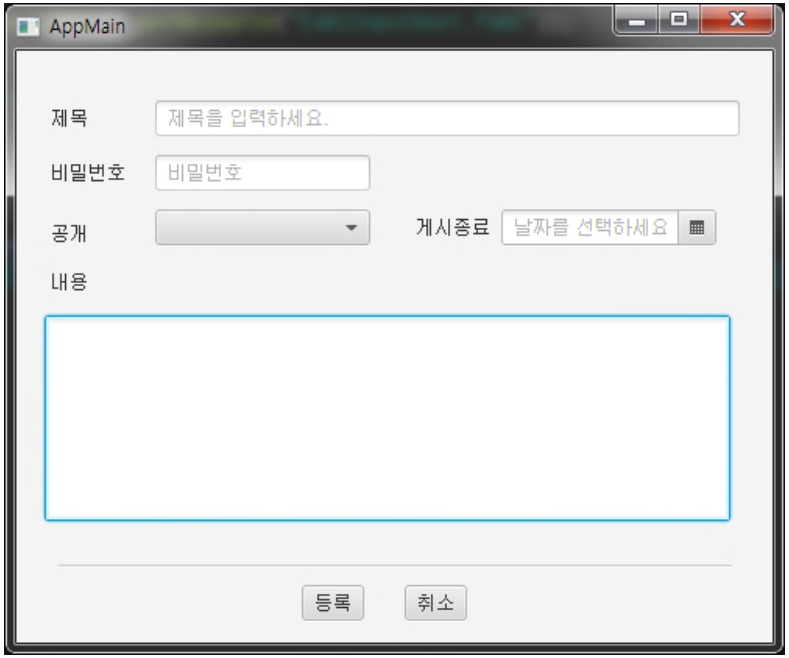
- 버튼 컨트롤은 마우스로 클릭할 수 있는 컨트롤로 ButtonBase를 상속하는 하위 컨틀롤을 말한다.
- Button, CheckBox, RadioButton, ToggleButton, Hyperlink 등.
- setGraphic() : 아이콘을 추가할 수 있다.
- userData 속성 : 프로그램에서 처리하는 데이터, Check, Radio, Toggle에 해당
- toggleGroup 속성 : Radio, Toggle에서 ToggleGroup으로 묶어 같은 그룹 내에서 하나의 컨트롤만 선택하게 할 수 있다.



JavaFX 컨트롤

❖ 입력 컨트롤

- 입력 컨트롤은 한 줄 입력을 위한 TextField, 다중 행 입력을 위한 TextArea, 패스워드 입력을 위한 PasswordField, 제한된 항목에서 선택하는 ComboBox 가 있다.
- DatePicker, ColorPicker, HTMLEditor 등.



JavaFX 컨트롤

❖ 뷰 컨트롤

- 뷰 컨트롤은 텍스트 또는 이미지 등을 보여주는데 목록 형태로 보여주는 ListView, 테이블 형태로 보여주는 TableView, 이미지를 보여주는 ImageView가 있다.

JavaFX 컨트롤 - 뷰 컨트롤

❖ ImageView 컨트롤

- ImageView는 이미지를 보여주는 컨트롤.
- fitWidth, fitHeight 속성 : ImageView의 폭과 높이를 지정.
- preserveRatio 속성 : 이미지의 종횡비를 유지 여부.

❖ ListView 컨트롤

- ListView는 항목들을 목록으로 보여주는 컨트롤.
- ListView에 항목을 추가하려면 setItems(ObservableList<T> value) 메소드를 사용
- ObservableList 구현 객체는 FXCollections.observableArrayList(E ... items) 정적 메소드로 생성
- 선택된 인덱스와 항목을 얻으려면 속성 감시를 이용할 수 있다.
- getSelectionModel()메소드 이용하여 MultipleSelectionModel을 얻고나서 selectedIndexProperty() 또는 selectedItemProperty()에 리스너를 설정

JavaFX 컨트롤 - 뷰 컨트롤

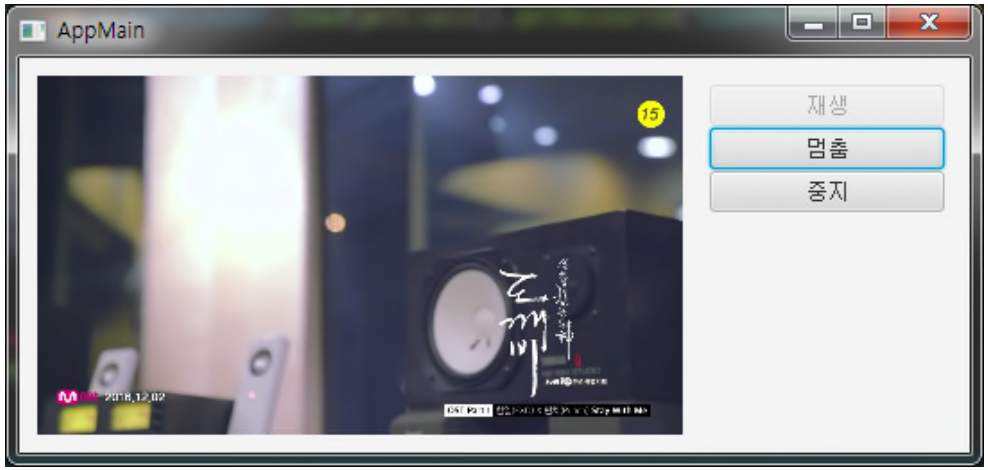
❖ TableView 컨트롤

- TableView는 테이블 형태로 데이터를 보여주는 컨트롤.
- TableView에 행(row)을 추가하려면 행의 데이터를 가지는 모델(model) 객체를 필요.
- 모델 속성 타입은 컬럼 값의 데이터 타입에 따라 javafx.beans.property 패키지의 SimpleXXXProperty를 사용하면 된다.
- TableColumn의 setCellValueFactory() 메서드는 매개값으로 제공되는 PropertyValueFactory("모델속성명")을 이용해서 모델 속성값을 TableColumn 값으로 세팅.
- TableView에서 선택된 행의 인덱스와 모델 객체를 얻으려면 속성 감시를 이용
- getSelectionModel() 메서드로 TableViewSelectionModel을 얻고, selectedIndexProprety 또는 selectedItemProperty에 리스너를 설정하면 된다.

JavaFX 컨트롤

❖ 미디어 컨트롤

- 미디어 컨트롤에는 비디오를 재생할 수 있는 MediaView 컨트롤과 볼륨 조절 및 재생 위치 조절을 위한 Slider 컨트롤 그리고 현재 진행 상태를 보여주는 ProgressBar, ProgressIndicator 컨트롤이 있다.
- MediaView 컨트롤은 비디오를 보여주는 용도로만 사용하며, 레이아웃상에서 비디오가 위치할 영역을 표시한다.
- MediaView 컨트롤은 비디오 재생을 위해 MediaPlayer가 있어야 한다.



JavaFX 컨트롤

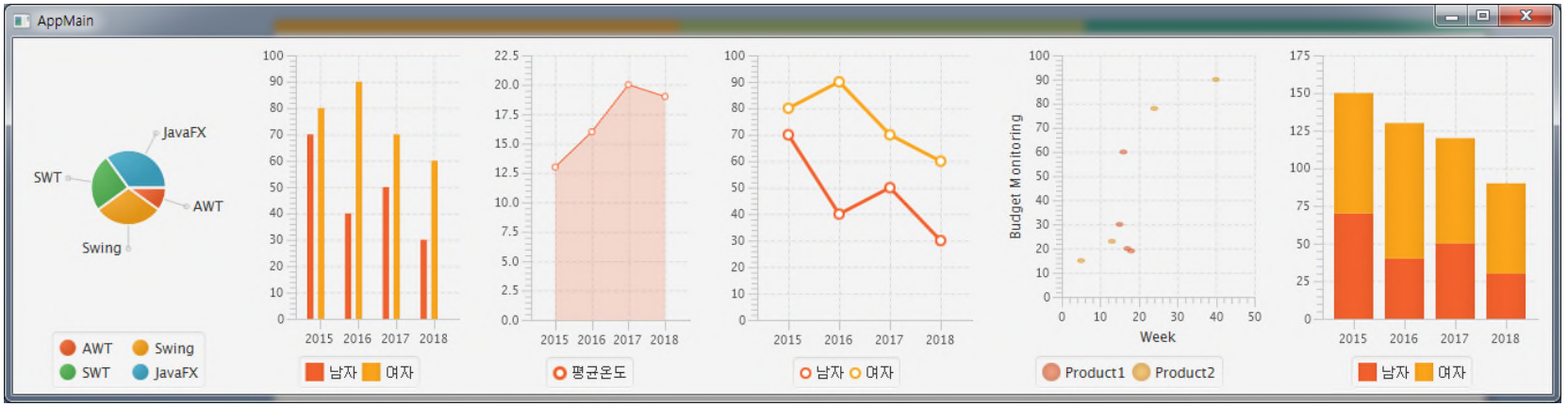
- MediaPlayer의 상태에 따라 자동 실행해야 할 코드들은 Runnable의 run() 메서드에 작성하고, setOnXXX() 메소드로 등록한다.
- MediaPlayer의 상태에 따라 자동 실행되는 Runnable 설정 메소드

상태	자동 실행 Runnable 설정 메소드	Runnable에 포함될 수 있는 코드
READY	setOnReady(Runnable r)	• currentTime 속성을 감시 • 재생 시간을 표시하는 리스너 등록 • 재생 버튼 활성화
PLAYING	setOnPlaying(Runnable r)	• 멈춤 및 정지 버튼 활성화
PAUSED	setOnPaused(Runnable r)	• 재생 및 정지 버튼 활성화
STOPPED	setOnStopped(Runnable r)	• 재생 버튼 활성화
EndOfMedia	setOnEndOfMedia(Runnable r)	• 재생 버튼 활성화

JavaFX 컨트롤

❖ 차트 컨트롤

- JavaFX는 다양한 차트를 생성하는 컨트롤을 제공한다.
- Javafx.scene.chart 패키지에 포함



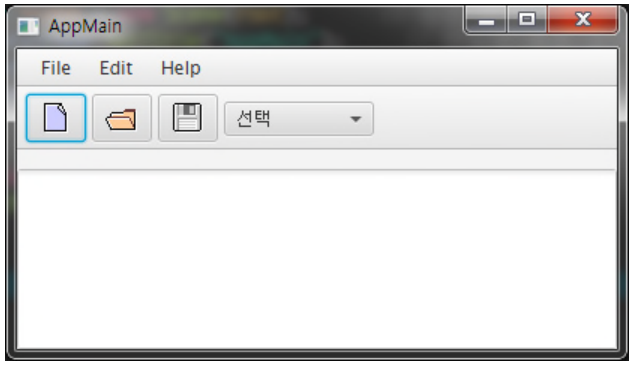
JavaFX 메뉴바와 툴바

❖ MenuBar 컨트롤

- MenuBar 컨트롤은 컨테이너 상단에 배치되어, 다양한 작업을 쉽게 선택하게 해준다.
- MenuBar 에는 Menu들이 배치된다.
- Menu에는 메뉴 아이템으로 MenuItem, CheckMenuItem, RadioMenuItem, CustomMenuItem, SeparatorMenuItem 을 추가 할 수 있다.
- Menu는 서브 메뉴를 갖는 Menu도 추가할 수 있다.

❖ Toolbar 컨트롤

- 자주 사용하는 메뉴를 버튼 형태의 모양으로 보여주는 컨트롤.
- Toolbar는 컨트롤이면서 컨테이너이기도 하다.



JavaFX 다이얼로그

- 다이얼로그(Dialog)는 주 윈도우에서 알림 또는 사용자의 입력을 위해서 실행되는 서브 윈도우라고 볼 수 있다.
- 다이얼로그는 자체적으로 실행될 수 없고, 주 윈도우(Owner)에 의해서 실행.
- 모달(modal)과 모달리스(modeless) 두 가지 종류가 있다.
- 모달(modal) : 다이얼로그를 닫기 전까지 소유자 윈도우 사용 불가
- 모달리스(modeless) : 소유자 윈도우를 계속 사용 가능
- FileChooser, DirectoryChooser, Popup 등이 있다.
- javafx.stage 패키지에 포함.

JavaFX 다이얼로그 - Popup

❖ Popup

- 투명한 컨테이너를 제공하는 모달리스 다이얼로그.
- Popup은 컨트롤의 툴팁(tooltip), 메시지 통지(notification), 드롭다운 박스(drop down boxes), 마우스 오른쪽 버튼 클릭시 메뉴 등을 만들 대 사용될 수 있다.

```
Popup popup = new Popup();
popup.getContent().add(FXMLLoader.load(getClass().getResource("popup.fxml")));

popup.show(primaryStage);                    // 화면 정중앙에서 실행
popup.show(primaryStage, anchorX, anchorY);  // 지정된 좌표에서 실행

popup.setAutoHide(true);     // 다른 윈도우로 포커스를 이동하면 Popup 자동으로 닫기
```

JavaFX 다이얼로그 - 커스텀 다이얼로그

❖ 커스텀 다이얼로그

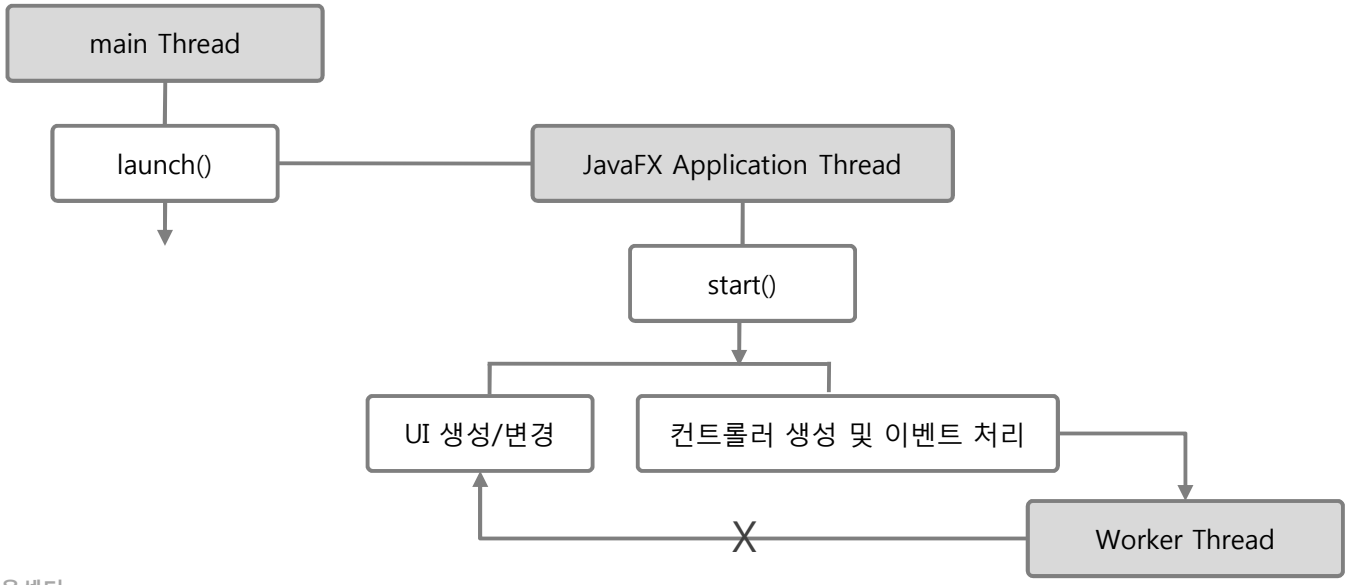
- 사용자가 정의하는 다이얼로그를 만들려면 Stage로 직접 생성해야 한다.

```
Stage dialog = new Stage(StageStyle.UTILITY); //윈도우 스타일 UTILITY
dialog.initModality(Modality.WINDOW_MODAL); // 모달 형태로 띄우기
dialog.initOwner(primaryStage); // 소유자 윈도우 지정
```

StageStyle 열거 상수	설명
DECORATED	일반적인 윈도우 스타일, 배경이 흰색, 제목줄에 장식(아이콘, 타이틀, 축소, 복원, 닫기 버튼 장식)이 있음
UNDECORATED	배경이 흰색, 제목줄 없음
TRANSPARENT	배경이 투명, 제목줄 없음
UNIFIED	DECORATED와 동일하나, 내용물의 경계선이 없음
UTILITY	배경이 흰색이고, 제목줄에 타이틀, 종료 버튼만 있음

JavaFX 스레드 동시성

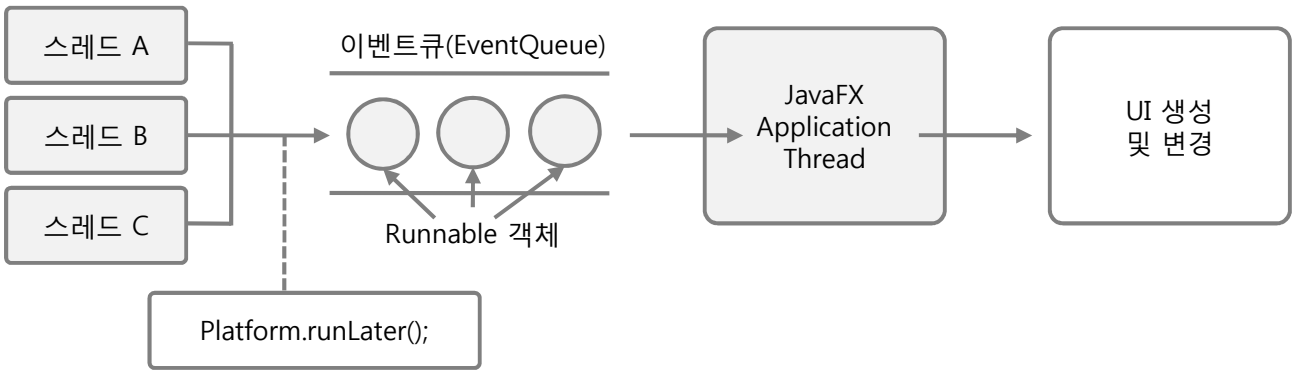
- JavaFX UI는 스레드에 안전하지 않기 때문에 UI를 생성하고 변경하는 작업은 JavaFX Application Thread가 담당하고, 다른 작업 스레드들은 UI를 생성하거나 변경할 수 없다.
- 작업 스레드가 직접 UI를 변경할 수 없기 때문에 다음 두 가지 방법으로 해결해야 함
  - Platform.runLater() 이용
  - javafx.concurrent API의 Task 또는 Service를 이용



JavaFX 스레드 동시성

❖ Platform.runLater() 이용

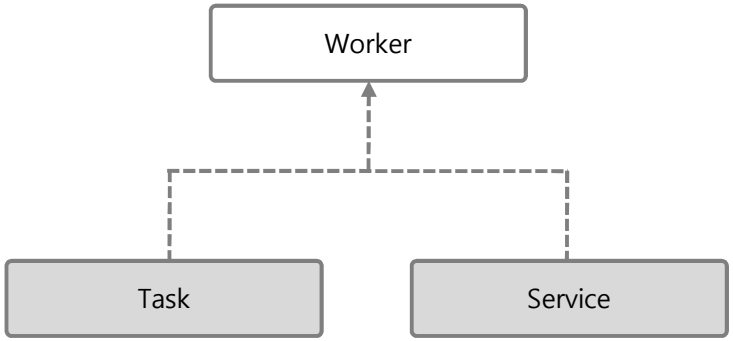
- 작업 스레드가 직업 UI를 변경할 수 없기 때문에 UI 변경이 필요할 경우, 작업 스레드는 UI 변경 코드를 Runnable로 생성하고 이것을 매개값으로 해서 Platform의 정적 메소드 runLater()를 호출할 수 있다.
- runLater() 메소드는 이벤트 큐(event queue)에 Runnable을 저장하고 즉시 리턴



JavaFX 스레드 동시성

❖ javafx.concurrent API의 Task 또는 Service를 이용

- javafx.concurrent 패키지는 스레드 동시성 API를 제공하고 있다.
- Worker 인터페이스와 두 가지의 구현 클래스인 Task, Service로 구성.
- Task 클래스 : JavaFX 애플리케이션에서 비동기 작업을 표현한 클래스.
- Service 클래스 : Task를 간편하게 시작, 취소, 재시작할 수 있는 기능 제공.



JavaFX 스레드 동시성

❖ Task 클래스

- Task 클래스는 작업 스레드에서 실행되는 하나의 작업을 표현한 추상 클래스.
- 하나의 작업을 정의할 때 Task를 상속해서 클래스를 생성한 후, call() 메서드를 재정의 하면 된다.
- call() 메서드의 리턴값은 작업 결과를 뜻한다.
- Task는 제네릭 타입인데, 타입 파라미터는 작업 결과 타입이며, call() 메서드의 리턴 타입과 동일해야 한다.

```
Task<Integer> task = new Task<Integer>(){
  @Override
  protected Integer call() throws
    Exception{

    int result = 0;
    // 작업 실행 코드
    return result;
  }
}
```

```
// Task 실행
Task<Integer> task = new Task<Integer>()
{ … };

Thread thread = new Thread(task);
thread.setDaemon(true);
thread.start();
```

JavaFX 스레드 동시성

❖ 작업 상태별 콜백

- 작업이 어떻게 처리되었는지에 따라 Task의 다음 세가지 메서드 중 하나가 자동 호출(콜백) 된다.
- 이 메소드들은 Task 클래스를 작성할 때 재정의해서 어플리케이션 로직으로 재구성 할 수 있습니다.
- 주목할 점은 이 메소드들은 작업 스레드상에서 호출되는 것이 아니라, JavaFX Application Thread 상에서 호출되기 때문에 안전하게 UI 변경 코드를 작성할 수 있습니다.

콜백 메서드	설명
succeeded()	성공적으로 call() 메서드가 리턴되었을 때
cancelled()	cancel() 메서드로 작업이 취소되었을 때
failed()	예외가 발생했을 때

JavaFX 스레드 동시성

❖ Service 클래스

- Service 클래스는 작업 스레드상에서 Task를 간편하게 시작, 취소, 재시작할 수 있는 기능을 제공한다.
- Service를 상속받고 createTask() 메서드를 재정의 하면 된다.
- createTask()는 작업 스레드가 실행할 Task 를 생성해서 리턴하면 된다.
- Service는 제네릭 타입인데, 타입 파라미터는 작업 결과 타입으로 지정.

```
class TimeService extends Service<Integer> {
  @Override
  protected Task<Integer> createTask() {
    Task<Integer> task = new Task<Integer>() { … };
    return task;
  }
}
```

```
TimeService timeService;
public void initialize(URL location, ResourceBundle resource){
  timeService = new TimeService();
  timeService.start();
}
```

JavaFX 스레드 동시성

❖ Service 시작, 취소, 재시작

- Service를 시작하려면 start() 메소드를 호출하면 되고, 취소하려면 cancel() 메소드를 호출하면 된다.
- 재시작이 필요한 경우 restart()를 호출하면 된다.
- 주의할 점은 이 메소드들은 JavaFX Application Thread상에서 호출해야 한다.
- start()와 restart()가 호출되면 매번 createTask()가 실행되고, 리턴된 Task를 받아 작업 스레드에서 실행합니다.

❖ 작업 상태별 콜백

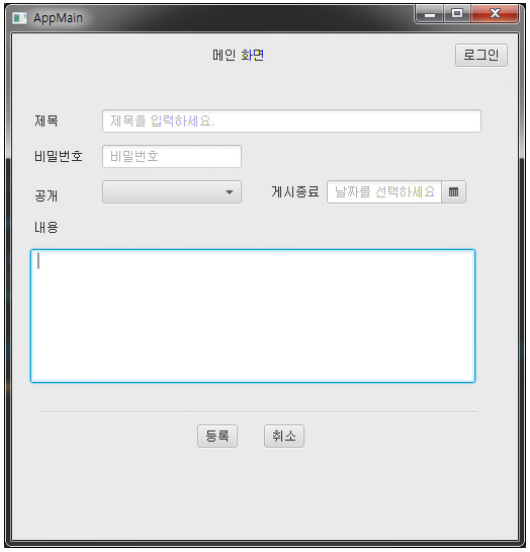
콜백 메서드	설명
succeeded()	성공적으로 call() 메서드가 리턴되었을 때
cancelled()	cancel() 메서드로 작업이 취소되었을 때
failed()	예외가 발생했을 때

화면 이동과 애니메이션

❖ 화면이동

- 화면 이동하는 가장 쉬운 방방은 Stage에서 새로운 Scene을 세팅하는 것

```
Parent login = FXMLLoader.load(getClass().getResource("login.fxml"));
Scene scene = new Scene(login);
Stage primaryStage = (Stage)btnLogin.getScene().getWindow(); // 현재 윈도우 가져오기
primaryStage.setScene(scene);
```



화면 이동과 애니메이션

❖ 애니메이션

- 애니메이션은 컨트롤 또는 컨테이너의 속성(Property) 변화를 주어진 시간 동안 진행함으로써 구현

```
Timeline timeline = new Timeline();
KeyValue keyValue = new KeyValue(parent.translateXProperty(), 종료값);
KeyFrame keyFrame = new KeyFrame(Duration.millis(지속시간), keyValue);
timeline.getKeyFrame().add(keyFrame);
timeline.play();
```

클래스	설명
Timeline	KeyFrame에 설정된 내용대로 애니메이션을 진행시키는 객체
KeyValue	타겟 속성(Property)과 종료값을 설정하는 객체
KeyFrame	애니메이션의 지속시간과 KeyValue를 설정하는 객체 (지속 시간 동안 타겟 속성의 값을 종료값까지 변화시킴)