

17.3 java FX 레이아웃

- Scene에는 다양한 컨트롤이 포함되는데 이를 배치하는 것이 레이아웃이다.

17.3.1 프로그램적 레이아웃

- 자바 코드로 UI 컨트롤을 배치하는 것을 말한다
- 자바 코드로만 개발하기 때문에 다른 언어를 익힐 필요가 없다
- 코드를 잘 정리 하지 않으면 난해한 프로그램이 될 확률이 높아진다
- 디자이너와 협력해서 개발하는 것이 어렵다
- 간단한 레이아웃 변경이나 스타일 변경이라도 자바 소스를 수정하고 재 컴파일해야 한다

```

1 package sec03.exam01_programmatical_layout;
2
3 import javafx.application.Application;
4 import javafx.stage.Stage;
5 import javafx.scene.layout.HBox;
6 import javafx.geometry.Insets;
7 import javafx.scene.control.TextField;
8 import javafx.scene.control.Button;
9 import javafx.scene.Scene;
10 import javafx.collections.ObservableList;
11
12 public class AppMain extends Application {
13     @Override
14     public void start(Stage primaryStage) throws Exception {
15         HBox hbox = new HBox();//HBox 컨테이너 생성
16         hbox.setPadding(new Insets(10,10,10,10));//안쪽 여백 설정
17         hbox.setSpacing(10);//컨트롤간의 수평 간격 설정
18
19         TextField textField = new TextField();//TextField 컨트롤 생성
20         textField.setPrefWidth(200);//TextField의 폭 설정
21
22         Button button = new Button();//Button 컨트롤 생성
23         button.setText("확인");//Button 글자 설정
24
25         ObservableList list = hbox.getChildren();//HBox의 ObservableList 얻기
26         list.add(textField);//TextField 컨트롤 배치
27         list.add(button); //Button의 컨트롤 배치
28
29         Scene scene = new Scene(hbox);//화면의 루트 컨테이너로 HBox 지정
30
31         primaryStage.setTitle("AppMain");//윈도우 창 제목 설정
32         primaryStage.setScene(scene);//윈도우 창에 화면 설정
33         primaryStage.show();//윈도우 창 보여주기
34     }
35
36     public static void main(String[] args) {
37         launch(args);
38     }
39 }

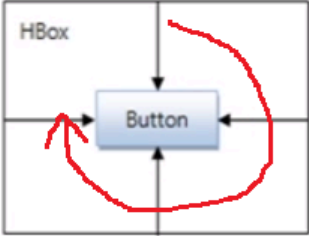
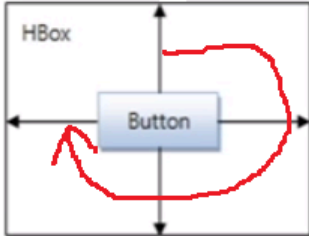
```

Colored by Color Scripter cs

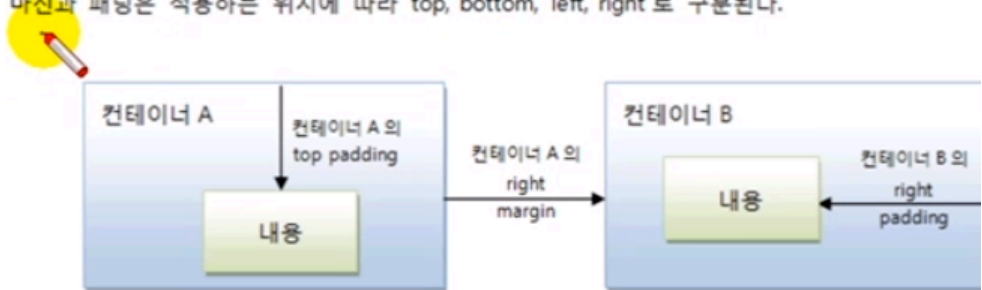
17.3.2 FXML 레이아웃

- FXML은 XML 기반의 마크업 언어이다.
- JavaFX UI 레이아웃을 자바 코드에서 분리해서 태그로 선언하는 방법을 제공한다.
- 웹 애플리케이션 및 안드로이드(Android) 앱을 개발하는 방법과 유사하다
- 디자이너와 협업이 가능하다
- 간단한 레이아웃 변경이나 스타일 변경시 자바 소스를 수정할 필요가 없다. FXML 태그만 수정하면 된다
- 레이아웃이 비슷한 장면(Scene)들간에 재사용이 가능하다

<pre> 1 package sec03.exam02_fxml_layout; 2 3 import javafx.application.Application; 4 import javafx.fxml.FXMLLoader; 5 import javafx.scene.Parent; 6 import javafx.scene.Scene; 7 import javafx.stage.Stage; 8 9 public class AppMain extends Application { 10 @Override 11 public void start(Stage primaryStage) throws Exception { 12 Parent root = FXMLLoader.load(getClass().getResource("root.fxml")); 13 Scene scene = new Scene(root); 14 15 primaryStage.setTitle("AppMain"); 16 primaryStage.setScene(scene); 17 primaryStage.show(); 18 } 19 20 public static void main(String[] args) { 21 launch(args); 22 } 23 } </pre> <p>Colored by Color Scripter cs</p>	<pre> 1 <?xml version="1.0" encoding="UTF-8"?> 2 3 <?import javafx.scene.layout.HBox?> 4 <?import javafx.geometry.Insets?> 5 <?import javafx.scene.control.*?> 6 7 <HBox xmlns:fx="http://javafx.com/fxml" > 8 <padding><!-- 안쪽 여백 설정 --> 9 <Insets top="10" right="10" bottom= 10 </padding> 11 <spacing>10</spacing> <!-- 컨트롤간의 12 13 <children><!-- 자식 컨트롤 추가 --> 14 <TextField><!-- TextField 선언 --> 15 <prefWidth>200</prefWidth> <!-- 16 </TextField> 17 18 <Button ><!-- Button 컨트롤 선언 -- 19 <text>확인</text> <!-- Button 글자 20 </Button> 21 </children> 22 </HBox> </pre>
---	---

구분	HBox 의 패딩	Button 의 마진
개념	<p>(HBox 기준)</p> 	<p>(Button 기준)</p> 
자바 코드	<pre>HBox hbox = new HBox(); hbox.setPadding(new Insets(50));</pre>	<pre>Button button = new Button(); HBox.setMargin(button, new Insets(50));</pre>
FXML 태그	<pre><HBox> <padding> <Insets topRightBottomLeft="50"/> </padding> </HBox></pre>	<pre><Button> <HBox.margin> <Insets topRightBottomLeft="50"/> </HBox.margin> </Button></pre>

마진과 패딩은 적용하는 위치에 따라 top, bottom, left, right 로 구분된다.



마진과 패딩같은 javafx.geometry.Insets 객체로 제공해야하는데 다음과 같이 생성한다.

```
//top, right, bottom, left 를 모두 동일한 값으로 설정할 때
new Insets(double topRightBottomLeft);

// top, right, bottom, left 를 다른 값으로 설정할 때
new Insets(double top, double right, double bottom, double left)
```

```
1 package sec03.exam03_margin_padding;
2
3 import javafx.application.Application;
4 import javafx.geometry.Insets;
5 import javafx.scene.Scene;
6 import javafx.scene.control.Button;
7 import javafx.scene.layout.HBox;
8 import javafx.stage.Stage;
9
10 public class AppMain extends Application {
11     @Override
12     public void start(Stage primaryStage) throws Exception {
13         //패딩 설정-----
14         /*HBox hbox = new HBox();
15         hbox.setPadding(new Insets(50, 10, 10, 50));
16         Button button = new Button();
17         button.setPrefSize(100, 100);*/
18
19         //마진 설정-----
20         HBox hbox = new HBox();
21         Button button = new Button();
22         button.setPrefSize(100, 100);
23         HBox.setMargin(button, new Insets(10, 10, 50, 50));
24
25         hbox.getChildren().add(button);
26
27         Scene scene = new Scene(hbox);
```

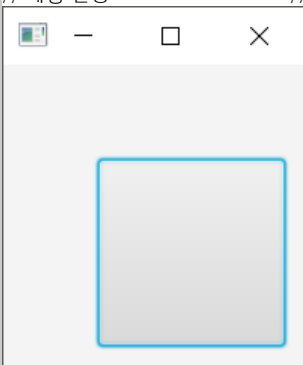
```

28
29     primaryStage.setTitle("AppMain");
30     primaryStage.setScene(scene);
31     primaryStage.show();
32 }
33
34 public static void main(String[] args) {
35     launch(args);
36 }
37 }

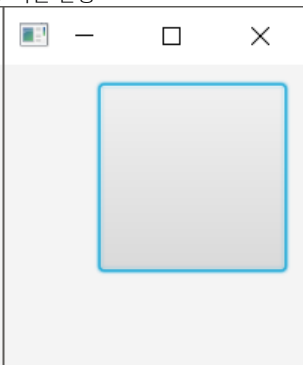
```

Colored by Color Scripter

// 패딩 설정



// 마진 설정



17.3.4 FXML 작성 규칙

- FXML 태그는 자바 코드로 변환되어 실행되기 때문에 자바 코드와 매핑 관계가 존재
- 매핑 관계만 잘 이해하면 JavaFX API를 참조해서 FXML 태그를 쉽게 작성

프로그램적 레이아웃 자바 코드	FXML 레이아웃 태그
<pre> HBox hbox = new HBox(); hbox.setPadding(new Insets(10,10,10,10)); hbox.setSpacing(10); </pre>	<pre> <HBox xmlns:fx="http://javafx.com/fxml" > <padding> <Insets top="10" right="10" bottom="10" left="10"/> </padding> <spacing>10</spacing> </HBox> </pre>
<pre> TextField textField = new TextField(); textField.setPrefWidth(200); </pre>	<pre> <TextField> <prefWidth>200</prefWidth> </TextField> </pre>
<pre> Button button = new Button(); button.setText("확인"); </pre>	<pre> <Button > <text>확인</text> </Button> </pre>
<pre> ObservableList list = hbox.getChildren(); list.add(textField); list.add(button); </pre>	<pre> <children> <TextField>...</TextField> <Button>...</Button> </children> </pre>

* 패키지 선언

- FXML 태그의 이름은 하나의 JavaFX API 클래스 이름과 매핑되기 때문에 해당 클래스가 존재하는 패키지를 반드시 <?import?> 태그로 선언해야 한다

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

```

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

<?import javafx.scene.layout.HBox?>
<?import javafx.scene.control.*?>

<루트컨테이너 xmlns:fx="http://javafx.com/fxml" >
  ...
</루트컨테이너>

```

네임스페이스 선언

* 태그 선언

FXML 태그는 < 와 > 사이에 태그 이름을 작성한 것인데, 반드시 시작 태그가 있으면 끝 태그도 있어야 한다. 그렇지 않으면 javax.xml.stream.XMLStreamException 예외가 발생한다.

<태그이름> ... </태그이름>

시작 태그와 끝 태그 사이에는 태그 내용이 작성되는데, 태그 내용이 필요 없을 경우에는 다음과 같이 시작 태그 끝에 />를 붙여야 한다.

<태그이름/>

태그 이름은 JavaFX 의 클래스명이거나, Setter 메소드명이 될 수 있다. 다음 표에서 Button 컨트롤을 자바 코드로 작성한 것과 FXML 태그로 작성한 것을 비교해보면 쉽게 이해가 될 것이다.

자바 코드	FXML
Button button = new Button(); button.setText("확인");	<Button > <text>확인</text> </Button>

* 속성 선언
FXML 태그는 다음과 같이 속성을 가질 수 있다. 속성값은 반드시 큰따옴표(") 또는 작은따옴표(')로 반드시 감싸야 한다. 그렇지 않으면 javax.xml.stream.XMLStreamException 예외가 발생한다.

<태그이름 속성명="값" 속성명='값'> ... </태그이름>

속성명은 Setter 메소드명이 오는데, 모든 Setter 가 사용될 수 있는 것은 아니고, 기본 타입(boolean, byte, short, char, int, long, float, double)의 값을 세팅하거나, String(문자열)을 세팅하는 Setter 만 올 수 있다. 예를 들어 Button 의 글자를 설정할 때 setText() 메소드를 사용하는데, 매개값이 문자열이므로 다음과 같이 text 속성으로 작성할 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"/>

* 객체 선언
-<클래스 속성="값"/>
일반적으로 다음과 같이 클래스명으로 태그를 작성하면 new 연산자로 기본 생성자를 호출해서 객체가 생성된다.

<클래스>

만약 생성자에 매개변수가 있고, 매개변수가 @NamedArg(javax.xml.beans.NamedArg) 어노테이션이 적용되어 있다면 속성명이나 자식 태그로 작성할 수 있다.

<클래스 속성="값">	<클래스> <매개변수>값</매개변수> </클래스>
--------------	-----------------------------------

자바 코드	FXML
HBox hbox = new HBox(); hbox.setPadding(new Insets(10,10,10,10));	<HBox> <padding> <Insets topRightBottomLeft="10"/> <Insets top="10" right="10" bottom="10" left="10"/> </padding> </HBox>

-<클래스 fx:value="값">

new 연산자로 객체를 생성하지 않고, 클래스가 valueOf(String) 메소드를 제공하는 경우가 있다. 예를 들어, String, Integer, Double, Boolean 클래스는 valueOf(String) 를 호출해서 객체를 생성한다. 이 경우 다음과 같이 FXML 태그를 작성할 수 있다.

<클래스 fxvalue="값" />

자바 코드	FXML
String.valueOf("Hello, World!");	<String fxvalue="Hello, World!"/>
Integer.valueOf("1");	<Integer fxvalue="1"/>
Double.valueOf("1.0");	<Double fxvalue="1.0"/>
Boolean.valueOf("false");	<Boolean fxvalue="false"/>

- <클래스 fx:constant="상수">
클래스에 정의된 상수값을 얻고 싶을 경우에는 다음과 같이 FXML 태그를 작성할 수 있다.

<클래스 fxconstant="상수" />

자바 코드	FXML
Button button = new Button(); button.setMaxWidth(Double.MAX_VALUE);	<Button> <maxWidth> <Double fxconstant="MAX_VALUE"/> </maxWidth> </Button>

- <클래스 fx:factory="정적메소드">
어떤 클래스는 new 연산자로 객체를 생성할 수 없고, 정적 메소드로 객체를 얻어야 하는 경우도 있다. 이 경우 다음과 같이 FXML 태그를 작성할 수 있다.

<클래스 fxfactory="정적메소드">

자바 코드	FXML
ComboBox combo = new ComboBox(); combo.setItems(FXCollections.observableArrayList("공개", "비공개"));	<ComboBox> <items> <FXCollections fx:factory="observableArrayList"> <String fxvalue="공개"/> <String fxvalue="비공개"/> </FXCollections> </items> </ComboBox>

17.3.5 FXML 로딩과 Scene 생성

> FXML 로딩

- FXML 파일을 읽어들이어 선언된 내용을 객체화하는 것을 말한다.
- FXMLLoader 의 load() 메소드를 이용

정적 메소드인 load()

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

인스턴스 메소드인 load()

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

```
Parent root = (Parent)loader.load();
```

- load()가 리턴하는 실제 객체는 FXML 파일에서 루트 태그로 선언된 컨테이너이다

```
HBox hbox = (HBox) FXMLLoader.load(getClass().getResource("xxx.fxml"));
```

> Scene 객체 생성

- FXML 로딩후 얻은 루트 컨테이너는 Scene을 생성할 때 매개값으로 사용된다.

```
Scene scene = new Scene(root);
```

17.3.6 JavaFX Scene Builder

- 드래그 앤 드롭 방식의 WYSIWYG 디자인 툴

