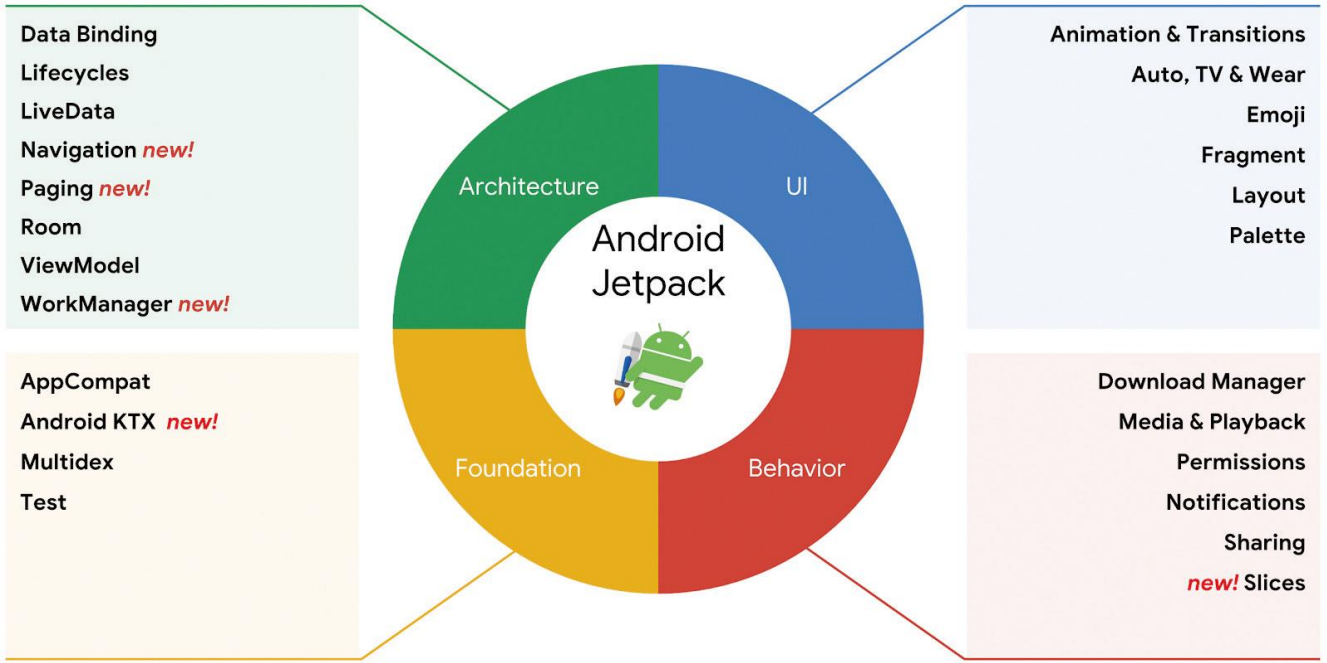
27장. 데이터 바인딩

27.1 제트팩과 AAC



구글 IO 행사에서 발표된 안드로이드 앱 개발을 위한 패키지 묶음
제트팩의 구성요소 중 아키텍처와 관련된 것이 AAC

27.2 데이터 바인딩 vs ButterKnife vs findViewById

데이터 바인딩은 이름 그대로 액티비티/프래그먼트의 데이터를 화면에 출력하는 부분을 도와주는 AAC 의 기법

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
android:orientation="vertical" android:layout_width="match_parent"
android:layout_height="match_parent">
<TextView
android:id="@+id/nameResultView"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
<Button
android:id="@+id/button"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
</LinearLayout>
```

27.2.1 findViewById() 방식

27.2 데이터 바인딩 vs ButterKnife vs findViewById

27.2.1 findViewById() 방식

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
TextView nameResultView;
Button button;
public String nameResult = "kkang";
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
nameResultView = findViewById(R.id.nameResultView);
button = findViewById(R.id.button);
nameResultView.setText(nameResult);
button.setOnClickListener(this);
}

@Override
public void onClick(View v) {
Toast.makeText(MainActivity.this, "show toast..", Toast.LENGTH_SHORT).show();
}
}
```

27.2 데이터 바인딩 vs ButterKnife vs findViewById

27.2.2 의존성 주입 방식

```
public class MainActivity extends AppCompatActivity {
@BindView(R.id.nameResultView)
TextView nameResultView;
@BindView(R.id.button)
Button button;
public String nameResult="kkang";
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
ButterKnife.bind(this);

nameResultView.setText(nameResult);
}
@OnClick(R.id.button)
public void onButtonClick(View view) {
Toast.makeText(this, "Button Click", Toast.LENGTH_SHORT).show();
}
}
```

27.2 데이터 바인딩 vs ButterKnife vs findViewById

27.2.3 데이터 바인딩 방식

```
public class MainActivity extends AppCompatActivity {
ActivityMainBinding binding;
public String nameResult = "kkang";
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
binding = DataBindingUtil.setContentView(this, R.layout.activity_main);
binding.setMyData(this);
}
public void onButtonClick(View view){
Toast.makeText(this,"show toast...",Toast.LENGTH_SHORT).show();
}
}
```

27.3 데이터 바인딩 기초

build.gradle

```
android {
…
dataBinding {
enabled = true
}
}
```

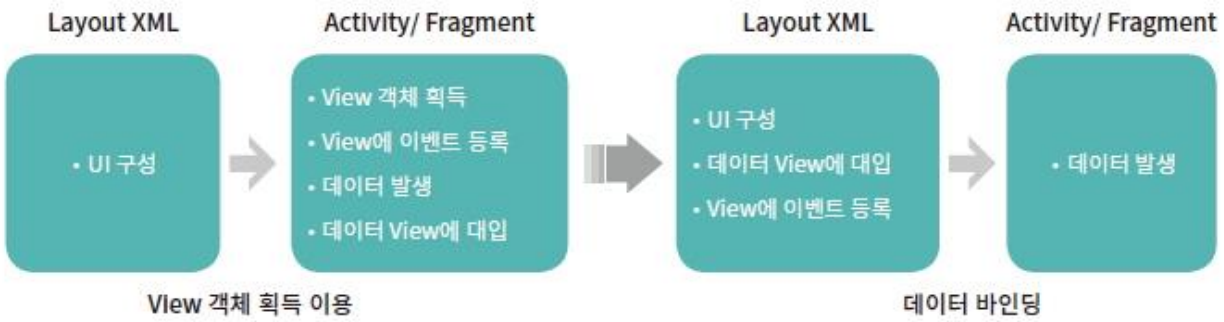
Layout xml

```
<?xml version="1.0" encoding="utf-8"?>
<layout xmlns:android="http://schemas.android.com/apk/res/android">
<data>
<variable
name="myData"
type="com.example.test10_27.MainActivity"/>
</data>
<LinearLayout … >
<TextView …
android:text="@{myData.nameResult}"/>
<Button …
android:onClick="@{myData::onButtonClick}"/>
</LinearLayout>
</layout>
```

27.3 데이터 바인딩 기초

Activity

```
public class MainActivity extends AppCompatActivity {
ActivityMainBinding binding;
public String nameResult = "kkang";
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
binding = DataBindingUtil.setContentView(this, R.layout.activity_main);
binding.setMyData(this);
}
public void onButtonClick(View view) {
Toast.makeText(this,"show toast...",Toast.LENGTH_SHORT).show();
}
}
```



27.4 다양한 타입의 데이터 바인딩

27.4.1 모델 클래스 바인딩

바인딩을 위한 모델 클래스는 POJO(Plain Old Java Object)로 만든다.

```
public class User {
public String firstName;
private String lastName;
public String phone;
public User(String firstName, String lastName, String phone) {
this.firstName = firstName;
this.lastName = lastName;
this.phone = phone;
}
public String getFirstName() {  return firstName; }
public String getLastName() { return lastName; }
public String getEmail() { return "a@a.com"; }
public String address() {
return "seoul";
}
public String getIcon() {
return "getIcon()....";
}
public String icon() {
return "icon().....";
}
}
```

27.4 다양한 타입의 데이터 바인딩

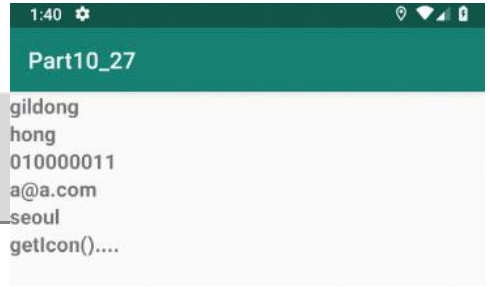
Layout XML

```
<layout xmlns:android="http://schemas.android.com/apk/res/android">
<data>
<variable name="user" type="com.example.test10_27.model.User"/>
</data>
<LinearLayout … >
<TextView …
android:text="@{user.firstName}"/>
<TextView …
android:text="@{user.lastName}"/>
<TextView …
android:text="@{user.phone}"/>
<TextView …
android:text="@{user.email}"/>
<TextView …
android:text="@{user.address}"/>
<TextView …
android:text="@{user.icon}"/>
</LinearLayout>
</layout>
```

27.4 다양한 타입의 데이터 바인딩

Activity 에서 모델 클래스 바인딩

```
ActivityTest2ModelBindingBinding binding = DataBindingUtil.setContentView(this,
R.layout.activity_test2__model_binding);
binding.setUser(new User("gildong","hong", "010000011"));
```



27.4.2. 리소스 바인딩

```
<TextView …
android:background="@{data.isErrorMsg ? @color/errorBgColor : @color/normalBgColor}"
android:text="@{@string/strResource}"/>
```

27.4.3. 컬렉션 타입 데이터 바인딩

```
<TextView …
android:text="@{Integer.toString(data.intArray[0])}"/>
<TextView …
android:text="@{data.strArray[1]}"/>
```

```
<TextView …
android:text="@{data.stringList.get(0)}"/>
<TextView …
android:text="@{data.userList.get(0).firstName}"/>
```

27.5 이벤트 바인딩

27.5.1 함수 참조 이벤트 바인딩

함수 참조란, ::를 이용하여 이벤트가 발생할 때 호출되어야 하는 함수 이름을 지정하는 방식 .

```
<Button android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Button1"
android:onClick="@{handler::onClickHandler}"/>
```

27.5.2 리스너 이벤트 바인딩

이벤트 등록을 위한 함수를 개발자 임의의 시그니처로 등록하려면 리스너 바인딩 방식을 이용 .

```
android:onClick="@{() -> handler.onClickListener1()}"
android:onClick="@{() -> handler.onClickListener2(model)}"
android:onClick="@{(view) -> handler.onClickListener3(view, model)}"
```

27.5.3 리턴 타입이 void 가 아닐 때 이벤트 바인딩

함수 참조, 리스너 바인딩 모두 이벤트 콜백 함수는 리턴 타입이 맞아야 한다..

27.6. <data> 구성요소

27.6.1 <import> 태그

XML 내에서 사용하고자 하는 클래스를 선언 .

```
<data>
<import type="com.example.test1_databinding.model.User6"/>
<variable name="model" type="User6"/>
</data>
```

27.6.2 <variable> 태그

<variable>은 XML에서 사용하고자 하는 변수를 선언하는 태그 .

```
<variable name="strData" type="String"/>
<variable name="intData" type="int"/>
<variable name="boolData" type="boolean"/>
<variable name="objData" type="Drawable"/>
…
<TextView …
android:text='@{"strData:"+strData +", intData:"+ intData +",
boolData:"+ boolData +", objData:"+ objData}'/>
```

27.6. <data> 구성요소

27.6.3 context 변수 활용

바인딩 식에서 context는 내장 변수이며 Context 객체를 지칭 .

```
<Button …
android:onClick="@{() -> ((AppCompatActivity)context).finish()}"/>
```

27.6.4 class 속성

자동으로 만들어지는 Binding 클래스의 이름과 패키지명을 변경

```
<data class="com.example.MyBindingClass">
```

27.6.5 <include> 태그

include 하는 곳에 선언된 <variable> 등이 include 대상이 되는 XML 까지 전달되지 않는다.

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:bind="http://schemas.android.com/apk/res-auto">
…
<include layout="@layout/test7_sub_include"
bind:model="@{model}"/>
```

27.7. 바인딩 연산식

수학: + - / * %
문자열 연결: +
논리: && ||
이항: & | ^
단항: + - ! ~
시프트: >> >>> <<
비교: == > < >= <=
instanceof
그룹화: ( )
리터럴: 문자, 문자열, 숫자, null
형변환
메서드 호출
필드 액세스
배열 액세스: [ ]
삼항 연산자: ?:

27.8. Observable 모델

변경된 데이터 반영

27.8.1. Observable 객체

Observable 인터페이스를 구현한 android.databinding. BaseObservable 클래스를 상속받아 모델 클래스를 만드는 방법
변경된 내용을 적용하려면 notifyPropertyChanged( ) 함수를 호출

```
public class User8 extends BaseObservable {
@Bindable
public String firstName = "gildong";
@Bindable
public String lastName = "hong";
private String email = "a@a.com";
@Bindable
public String getEmail() {
return email;
}
public void setEmail(String email) {
this.email = email;
notifyPropertyChanged(BR.email);
}
public void onClick(Context context){
firstName = "hyeonjin";
lastName = "ryu";
email ="b@b.com";
notifyPropertyChanged(BR.email);
}
}
```

27.8. Observable 모델

27.8.2. ObservableField
변경사항을 적 용하기 위해 함수를 호출하지 않아도 변경사항이 자동으로 적용

```
public class User8ObservableField {
public final ObservableField<String> firstName = new ObservableField<>("gildong");
public final ObservableField<String> lastName = new ObservableField<>("hong");
public final ObservableInt age = new ObservableInt(30);
public String email = "a@a.com";
public void onClick(Context context){
Toast.makeText(context, "onClick call...", Toast.LENGTH_SHORT).show();
email = "b@b.com";
firstName.set("hyeonjin3");
lastName.set("ryu3");
age.set(20);
email = "c@c.com";
}
public void onClickChangeBindableFieldProperty(){
firstName.set("aaa"); //클래스의 모든 변경된 값까지 적용된다.
}
public void onClickChangeNonBindableFieldProperty(){
email="d@d.com"; //전혀 변화가 없다.
}
}
```

27.8. Observable 모델

27.8.3. Observable 컬렉션
변경 감지가 필요한 데이터만 따로 모아 컬렉션 타입에 등록 해 이용하는 방법을 제공
이를 위해 android.databinding.ObservableArrayMap, android. databinding.ObservableArrayList 등을 제공

```
ObservableArrayMap<String, Object> user = new ObservableArrayMap<>();
user.put("firstName", "Google");
user.put("lastName", "Inc.");
user.put("age", 17);
binding.setUser(user);
```

27.9. BindingAdapter

BindingAdapter는 XML의 속성을 위한 함수를 실행하는 방법
XML의 뷰에 임의의 이름으로 속성을 명시하고 그 속성에 의해 함수가 실행되게 하는 방법

```
<layout xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:bind="http://schemas.android.com/tools">
<TextView …
bind:items="@{10}"/>
</layout>
```

```
public class Test {
@BindingAdapter("bind:items")
public static void bindItem(TextView textView, int no) {
int sum = 0;
for(int i = 0; i <= no; i++){
sum += i;
}
textView.setText(String.valueOf(sum));
}
}
```

Step by Step 실습 – 27-1. 데이터 바인딩.

- 그레이들 작업
- 파일 복사
- AndroidManifest 작업
- item_main.xml
- MainActivity.java