ToiletPaper #84

Android Data Binding

Autor: Tobias Schröpf / Senior Software Architect / Business Division Automotive World

✕ Problem

In vielen Android-Apps besteht UI-Code aus Aufrufen von findViewById, um Views aus dem Layout eines Screens zu finden, anschließenden Downcasts dieser Views in den gewünschten Typ (z.B. TextView) und schließlich Aufrufen auf den Views, um sie zu manipulieren (z.B. setText).
Dieser (unnötige) „Glue Code“, welcher die UI an die Models bindet, befindet sich in der Regel in den meist ohnehin komplexen Activities und Fragmenten.

✓ Lösung

Mit Hilfe der Android Data Binding Library von Google lässt sich dieser Boilerplate-Code eliminieren. ModelKlassen (z.B. POJOs) können direkt aus den Layout-XMLs referenziert werden, um z.B. Drawables und StringRessourcen direkt in die Views zu setzen.
Mit Hilfe von ObservableFields bzw. BaseObservables kann das Layout sich sogar an Properties der Models **binden** und Änderungen automatisch anwenden.

➔ Beispiel

```xml
<?xml version="1.0" encoding="utf-8"?>
<!-- Wrap your Layout into <layout></layout> tags -->
<layout [...]>

    <!-- define variables and imports you want to use -->
    <data>
        <variable name="model" type="de.schroepf.databinding.model.User" />
    </data>

    <LinearLayout [...]>

        <!-- Surround any data binding expression with @{...} -->
        <ImageView [...] app:imageResource='@{model.photo}' />
        <TextView [...] android:text='@{model.firstName + " " + model.lastName}' />
    </LinearLayout>
</layout>
```

```
// The fields of this model class can be accessed in the layout.xml file
// Extend BaseObservable or use ObservableFields to allow the layout to respond to changes.
public class User {
    public final String firstName;
    public final String lastName;
    @DrawableRes
    public final int photo;

    public User(String firstName, String lastName, @DrawableRes int photo) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.photo = photo;
    }
}
```

```
public class MainActivity extends Activity {
    // represents the state we want to display in this activity
    private User myModel = new User("Coffee", "Toffee", R.drawable.coffee_toffee);

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Make sure to use DataBindingUtil to setContentView/inflate your view and retrieve the binding
        ActivityMainBinding binding = DataBindingUtil.setContentView(this, R.layout.activity_main);

        // use the binding to access the layout's view hierarchy (typesafe!) and set variables
        binding.setModel(myModel);
    }
}
```

✚ Weiterführende Aspekte

- Mit Hilfe der generierten Binding-Klassen kann programmatisch, typsicher (kein Downcast!) und ohne findViewById auf die Views in den Layouts zugegriffen werden
- „Method References“ und „Listener Bindings“ ermöglichen das Binden von UI-Events an Java-Code ebenfalls direkt im Layout XML
- Google Dokumentation: https://developer.android.com/topic/libraries/data-binding/index.html