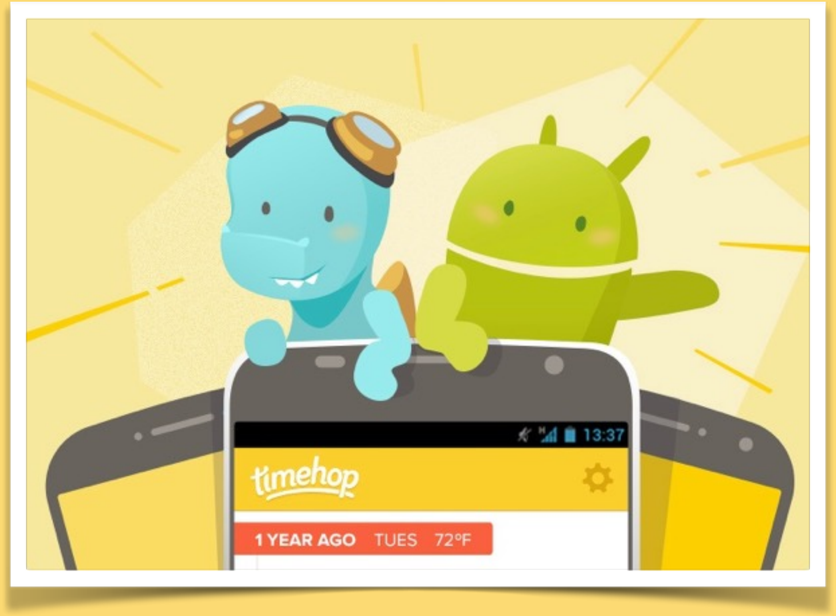
timehop

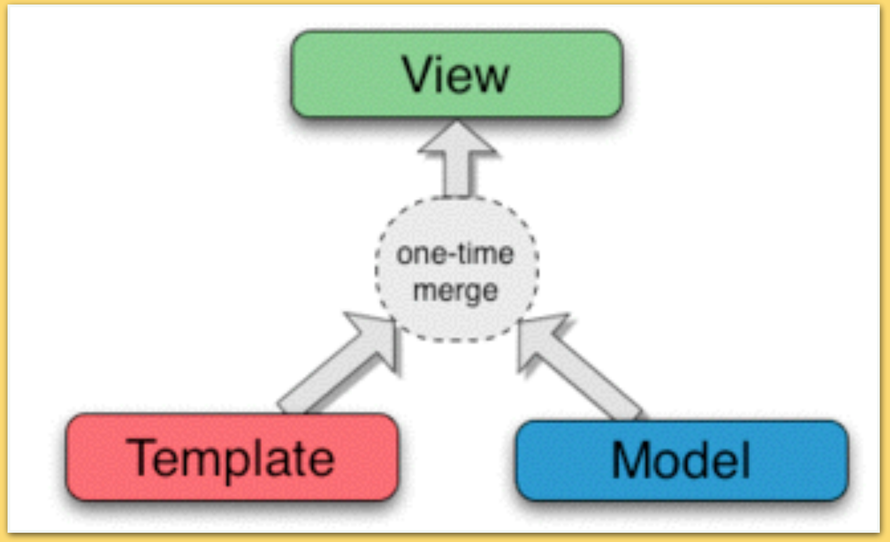
JACOB TABAK
@JACOBTABAK

Data Binding Techniques

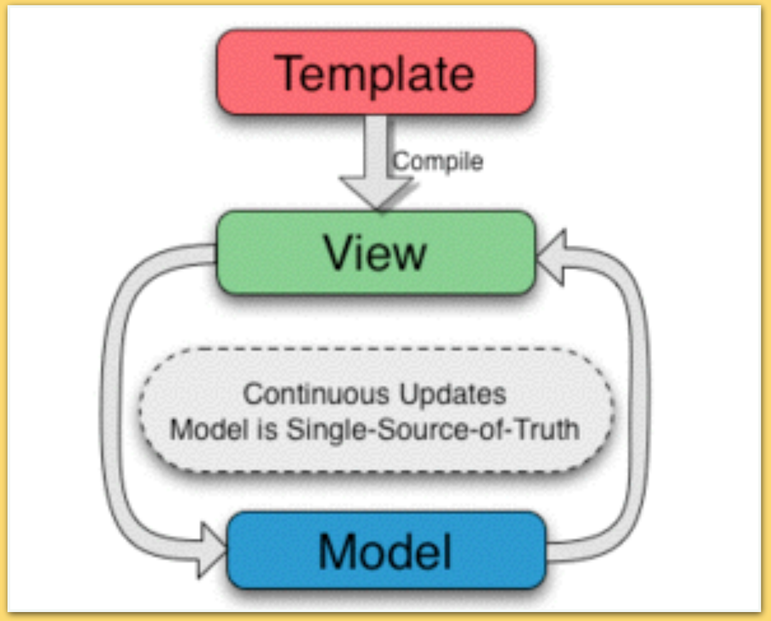


Data binding establishes a connection between your app's UI and data models.

One-way binding



Two-way binding



Diagrams from docs.angularjs.org

Why databinding for Android?

- Removes UI code from Activities & Fragments
- XML becomes single source of truth for UI
- Eliminates the primary need for view IDs
  - and by extension, `findViewById()`

Senseless casting

```androidlayout
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/first_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        tools:text="Bob"/>
    <TextView
        android:id="@+id/last_name"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        tools:text="Smith"/>
</LinearLayout>
```

```androidcode
public class OldWayActivity extends AppCompatActivity {
  private static final Employee employee =
      Employee.newInstance("Bob", "Smith");

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_oldway);
    TextView firstNameView =
        (TextView) findViewById(R.id.first_name);
    TextView lastNameView =
        (TextView) findViewById(R.id.last_name);
    firstNameView.setText(employee.firstName());
    lastNameView.setText(employee.lastName());
  }
}
```

Every view that gets updated in code needs an ID

TextViews have no concept of an employee, just Strings

```
<layout>
    <data>
        <variable
            name="employee"
            type="me.tabak.databinding.model.Employee"/>
    </data>
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{employee.firstName}"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@{employee.lastName}"/>
    </LinearLayout>
</layout>
```

No IDs needed

View properties mapped directly to model

```
public class BindingActivity extends AppCompatActivity {
  private static final Employee employee =
      Employee.newInstance("Bob", "Smith");

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EmployeeItemBinding binding = DataBindingUtil
        .setContentView(this, R.layout.employee_item);
    binding.setEmployee(employee);
  }
}
```

No casting needed

UI Boilerplate

- More code
- More development time
- More bugs
- Harder to read

Open source to the rescue!

Butter Knife

```
public class FancyFragment extends Fragment {
  @Bind(R.id.button) Button button;

  @Override public View onCreateView(LayoutInflater inflater,
      ViewGroup container, Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fancy_fragment, container, false);
    ButterKnife.bind(this, view);
    button.setText("Click me!");
    return view;
  }

  @OnClick(R.id.button)
  void onButtonClick(View view) {
    // handle click
  }
}
```

Android Annotations

```
@EFragment(R.layout.fancy_fragment)
public class FancyFragment extends Fragment {
  @ViewById(R.id.load_button) Button loadButton;

  @AfterViews public View initUi() {
    loadButton.setText("Click me!");
  }

  @Click(R.id.load_button)
  void onLoadClicked(View view) {
    loadData();
  }

  @Background
  void loadData() {
    setData(database.loadData());
  }

  @UiThread
  void setData(List<String> data) {
    adapter.setData(data);
  }
}
```

RoboBinding

```
<LinearLayout
    xmlns:bind="http://robobinding.org/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <TextView
        bind:text="{hello}"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <Button
        android:text="Say Hello"
        bind:onClick="sayHello"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

```
@org.robobinding.annotation.PresentationModel
public class PresentationModel
  implements HasPresentationModelChangeSupport {

  private String name;

  public String getHello() {
    return name + ": hello!!!";
  }
  ...
  public void sayHello() {
    firePropertyChange("hello");
  }
}
```

```
// in your activity
PresentationModel presentationModel = new PresentationModel();
View rootView = Binders
  .inflateAndBindWithoutPreInitializingViews(this, R.layout.activity_main, presentationModel);
```


Windows
Presentation
Foundation


ANGULARJS
by Google


Knockout.


apache
Flex™

History @ Google

- Originally slated for release with Lollipop
- Postponed because L ended up being huge
- Reconsidered for Android M
- Didn’t get much traction due to performance concerns
- Design doc passed around - gained wide acceptance
- Prototypes built in late 2014, easing concerns

```xml
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

wrap the existing XML in a `<layout>` tag

```
<?xml version="1.0" encoding="utf-8"?>
<layout
    xmlns:android="http://schemas.android.com/apk/res/android">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>
</layout>
```

xmlns declaration goes in the `<layout>` tag

variables can be declared in the `<data>` tag

```
<?xml version="1.0" encoding="utf-8"?>
<layout
    xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable
            name="employee"
            type="com.example.models.Employee"/>
    </data>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@{employee.name}" />
</layout>
```

variable properties can be referenced in XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<layout
    xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable
            name="employee"
            type="com.example.models.Employee"/>
        <import type="android.view.View"/>
    </data>
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@{employee.name}"
        android:visibility="@{employee.fired ? View.GONE : View.VISIBLE}"/>
</layout>
```

classes can be imported for convenience, just like in java

sophisticated expression language

Expression Language

Supports almost everything you can do in Java...
...without code completion / static checks!

Mathematical + - / * %
String concatenation +
Logical && ||
Binary & | ^
Unary + - ! ~
Shift >> >>> <<
Comparison == > < >= <=
instanceof

Grouping ()
Literals
Cast
Method calls
Field access
Array access []
Ternary operator ?:

Null safety

```
// JSON from API
{
  "location": {
    "latitude": “51.5033630”,
    "longitude": “-0.1276250”
  }
}

// Java
if (response != null && response.location != null) {
  latitudeView.setText(response.location.latitude);
}

// expression language
android:text="@{response.location.latitude}"
```

Generated Bindings

activity_main.xml => ActivityMainBinding.java

```
ActivityMainBinding binding =
  ActivityMainBinding.inflate(getLayoutInflater());
```

- or -

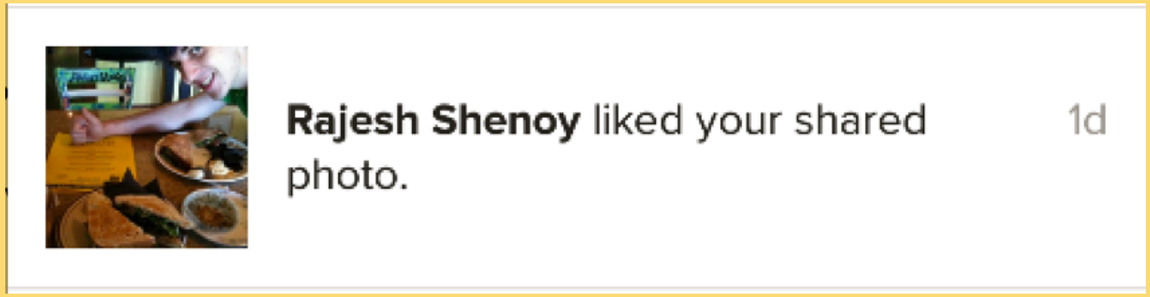
```
View view = getLayoutInflater()
  .inflate(R.layout.activity_main, parent, false)

ActivityMainBinding binding =
  ActivityMainBinding.bind(view);
```

Generic Bindings (Reusable ViewHolder)

```
public class DataBoundViewHolder<T extends ViewDataBinding>
  extends RecyclerView.ViewHolder {

  private T binding;

  public DataBoundViewHolder(T binding) {
    super(binding.getRoot());
    this.binding = binding;
  }

  public T getBinding() {
    return binding;
  }
}
```

Avoid complex expressions

- Limited code completion/formatting/static checks on expressions
- Consider using ViewModels rather than complex expressions
- ViewModels mediate communication between views and models




```
public CharSequence formattedText() {
  return spanFactory.bold(
      resources.getString(
        R.string.notification_format,
        notification.participantName(),
        notification.text()),
      0,
      notification.participantName().length(),
      Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
}
```

Automatic Setters

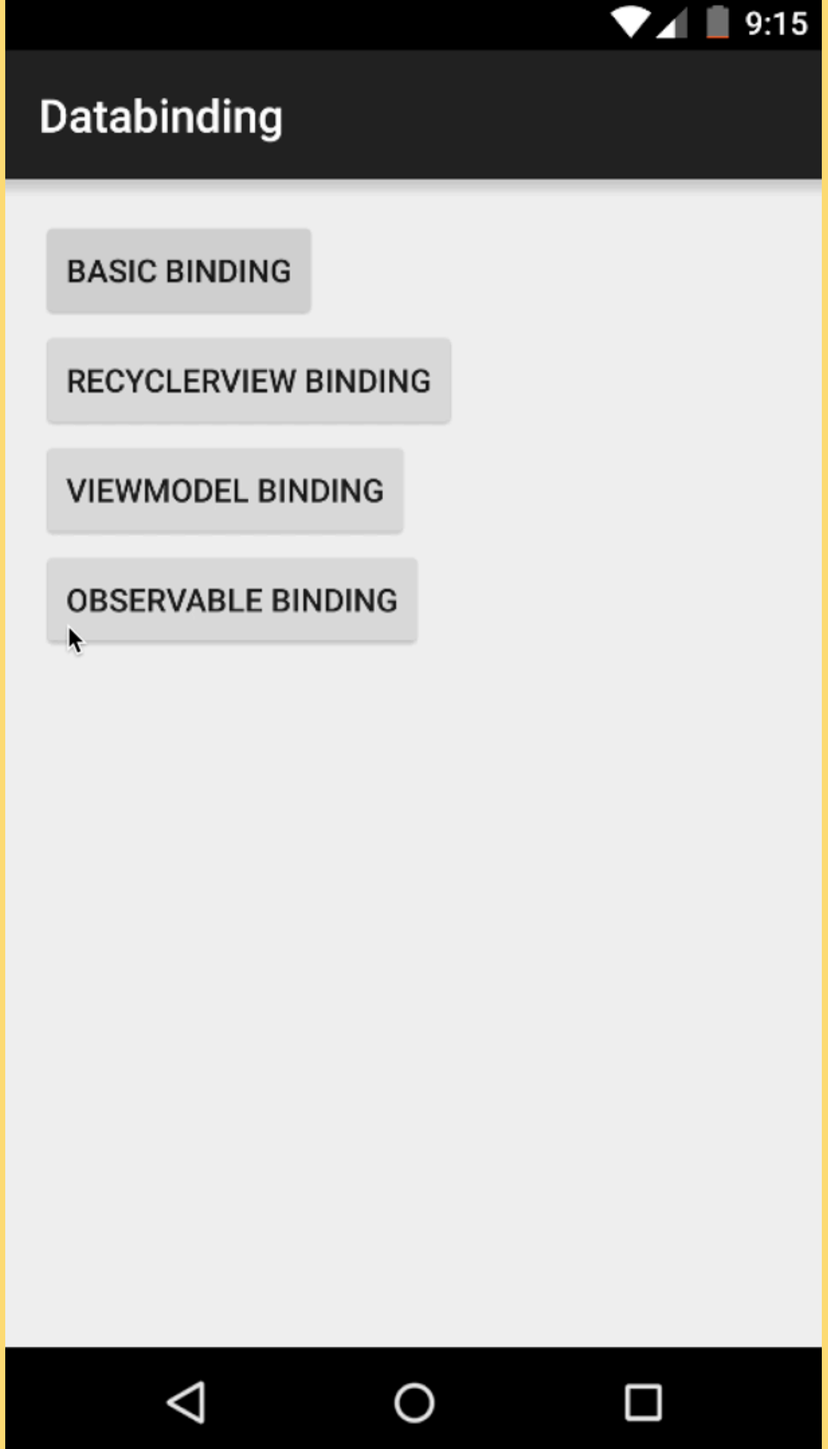
Automatically calls
CompoundButton.setOnCheckedChangeListener()

```
<CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Check me!"
    app:onCheckedChangeListener="@{viewModel.myCheckListener}"/>
```

```
public class ViewModel {
   public CompoundButton.OnCheckedChangeListener myCheckListener() {
     return new CompoundButton.OnCheckedChangeListener() {
       @Override
       public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
         // do something
       }
     };
   }
 }
```

```
public class ViewModel {
  public void myCheckListener(CompoundButton buttonView, boolean isChecked) {
    // do something
  }
}
```

Observable Objects


9:15
Databinding
BASIC BINDING
RECYCLERVIEW BINDING
VIEWMODEL BINDING
OBSERVABLE BINDING

```xml
<?xml version="1.0" encoding="utf-8"?>
<layout
    xmlns:android="http://schemas.android.com/apk/res/android">
    <data>
        <variable
            name="person"
            type="me.tabak.databinding.model.ObservablePerson"/>
    </data>
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="First Name"
            android:addTextChangedListener="@{person.firstNameChanged}"/>
        <EditText
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Last Name"
            android:addTextChangedListener="@{person.lastNameChanged}"/>
        <TextView
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@{person.formattedName}"/>
    </LinearLayout>
</layout>
```

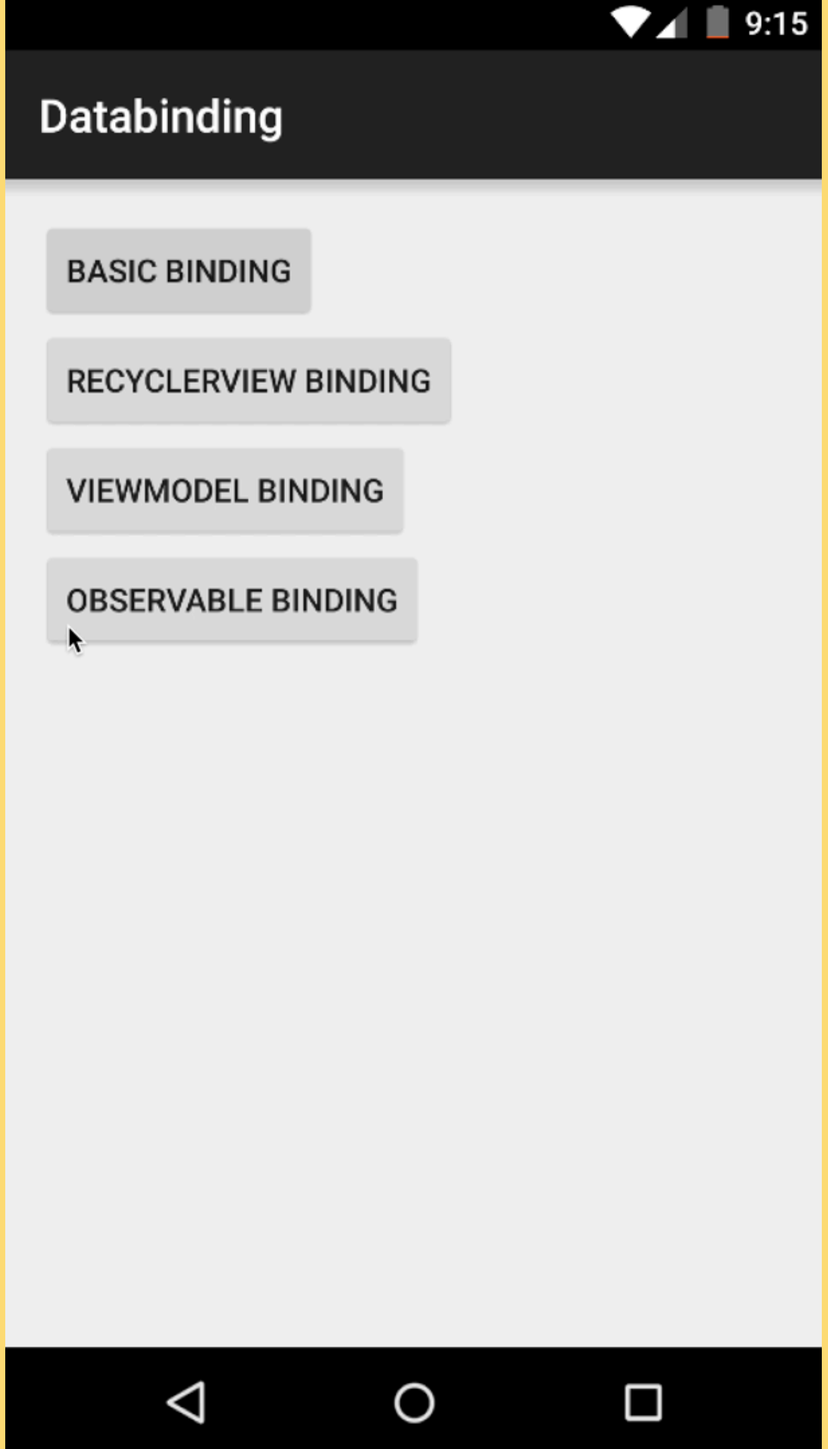
Assigning text watchers to edittexts

```java
public class ObservablePerson extends BaseObservable {
  private String firstName;
  private String lastName;

  @Bindable
  public String getFormattedName() {
    return lastName + ", " + firstName;
  }

  public final TextWatcher firstNameChanged = new SimpleTextWatcher() {
    @Override
    public void afterTextChanged(Editable s) {
      firstName = s.toString();
      notifyPropertyChanged(BR.formattedName);
    }
  };

  public final TextWatcher lastNameChanged = new SimpleTextWatcher() {
    public void afterTextChanged(Editable s) {
      lastName = s.toString();
      notifyPropertyChanged(BR.formattedName);
    }
  };
}
```

Base class for observable binding

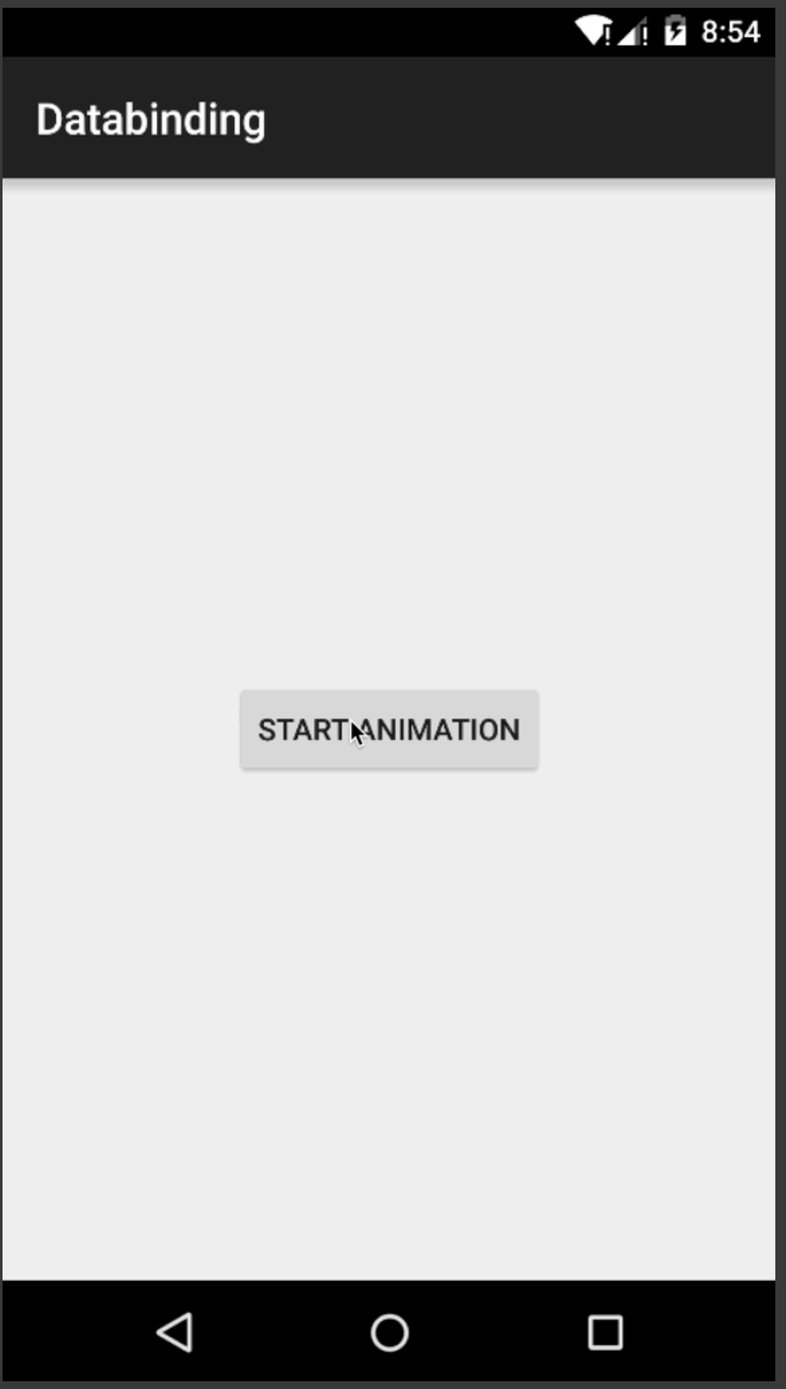
Generated class like R.java

```java
public class ObservableBindingActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    MyObservableBinding binding =
        DataBindingUtil.setContentView(this, R.layout.activity_observable_binding);
    binding.setPerson(new ObservablePerson());
  }
}
```


9:15
Databinding
BASIC BINDING
RECYCLERVIEW BINDING
VIEWMODEL BINDING
OBSERVABLE BINDING

Observable Fields

- Self-contained observable objects that contain a single field
- ObservableBoolean, ObservableByte, ObservableChar, ObservableShort, ObservableInt, ObservableLong, ObservableFloat, ObservableDouble, ObservableParcelable
- Serializable & Parcelable observables are useful for maintaining UI state across configuration changes




```java
public class ViewModel {
  public ObservableBoolean isAnimating = new ObservableBoolean();

  public void onStartAnimationClicked(View view) {
    isAnimating.set(true);
    binding.let_me_explain_you_data_binding.animate()
        .translationX(binding.getRoot().getWidth())
        .rotation(360)
        .setDuration(3000)
        .setInterpolator(new LinearInterpolator())
        .setListener(new AnimatorListenerAdapter() {
          @Override
          public void onAnimationEnd(Animator animation) {
            isAnimating.set(false);
          }
        })
        .start();
  }
}
```

Binding Adapters

- Automatic setters take care of most things
- Some properties don’t have setters (e.g. `paddingLeft`)
- What about threading considerations?
- **Binding adapters** solve these problems

```java
@BindingAdapter("android:paddingLeft")
public static void setPaddingLeft(View view, int paddingLeft) {
  view.setPadding(
      paddingLeft,
      view.getPaddingTop(),
      view.getPaddingRight(),
      view.getPaddingBottom());
}
```

*This binding already exists in `android.databinding.adapters.ViewBindingAdapter`

```
<ImageView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    bind:imageUri="@{imageUri}" />
```

```
@BindingAdapter("bind:imageUri")
public static void loadImageFromUri(ImageView view, Uri uri) {
  Picasso.with(view.getContext())
      .load(uri)
      .fit()
      .centerCrop()
      .into(view);
}
```

```
@BindingAdapter({"bind:imageUri", "bind:placeholder"})
public static void loadImageFromUri(
  ImageView view,
  Uri uri,
  Drawable placeholder
) {
  Picasso.with(view.getContext())
      .load(uri)
      .fit()
      .placeholder(placeholder)
      .centerCrop()
      .into(view);
}
```

Performance

- 100% dependent on generated code - no reflection
- `findViewById()` requires a traversal for each view
- Data Binder only has to traverse once for all views
- Data changes are deferred until next frame for batching
- Bitflags are used to track and check invalidation
- Expressions are cached, e.g.

```
a ? (b ? c : d) : y
e ? (b ? c : d) : x
```

Conclusions

Pros

- Removes UI code from Activities/Fragments
- Already in the support library
- Performance rivals the best hand written code
- Declarative XML layouts are SSOT
- API is complete, release is imminent
- Don’t need to worry about main thread for UI

Cons

- IDE integration, IDE integration, IDE integration
- No IDE support for expressions
- Somewhat confusing error messages
- No refactoring support
- Sometimes clean build is required

timehop

JACOB TABAK
@JACOBTABAK

Questions