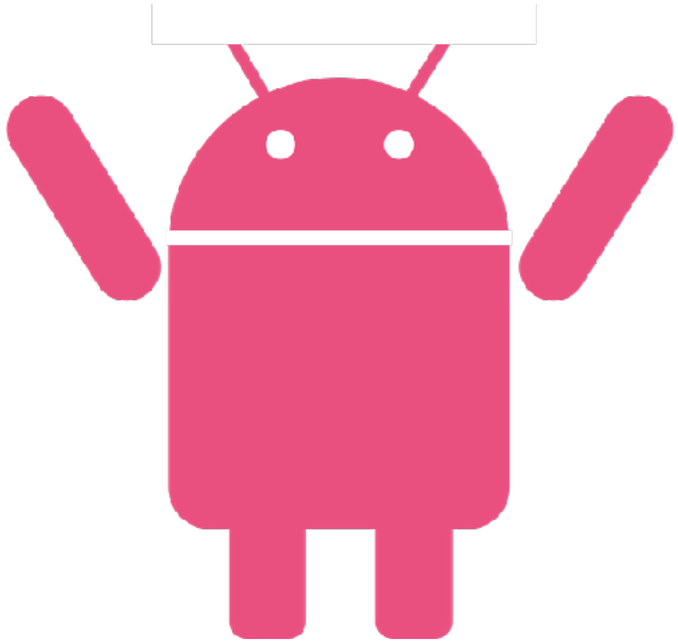
Effective Android Data Binding

Eric Maxwell
Android Developer
@emmax

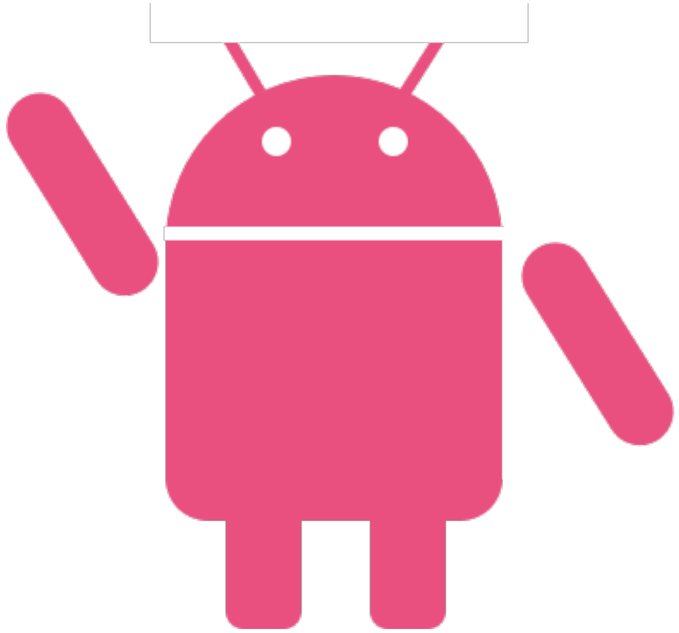


What are you going to learn?

- Lab 1: Data Binding Introduction
- Lab 2: Variables
- Lab 3: Adapters + Converters
- Lab 4: (Optional) AC ViewModels + LiveData

Questions?

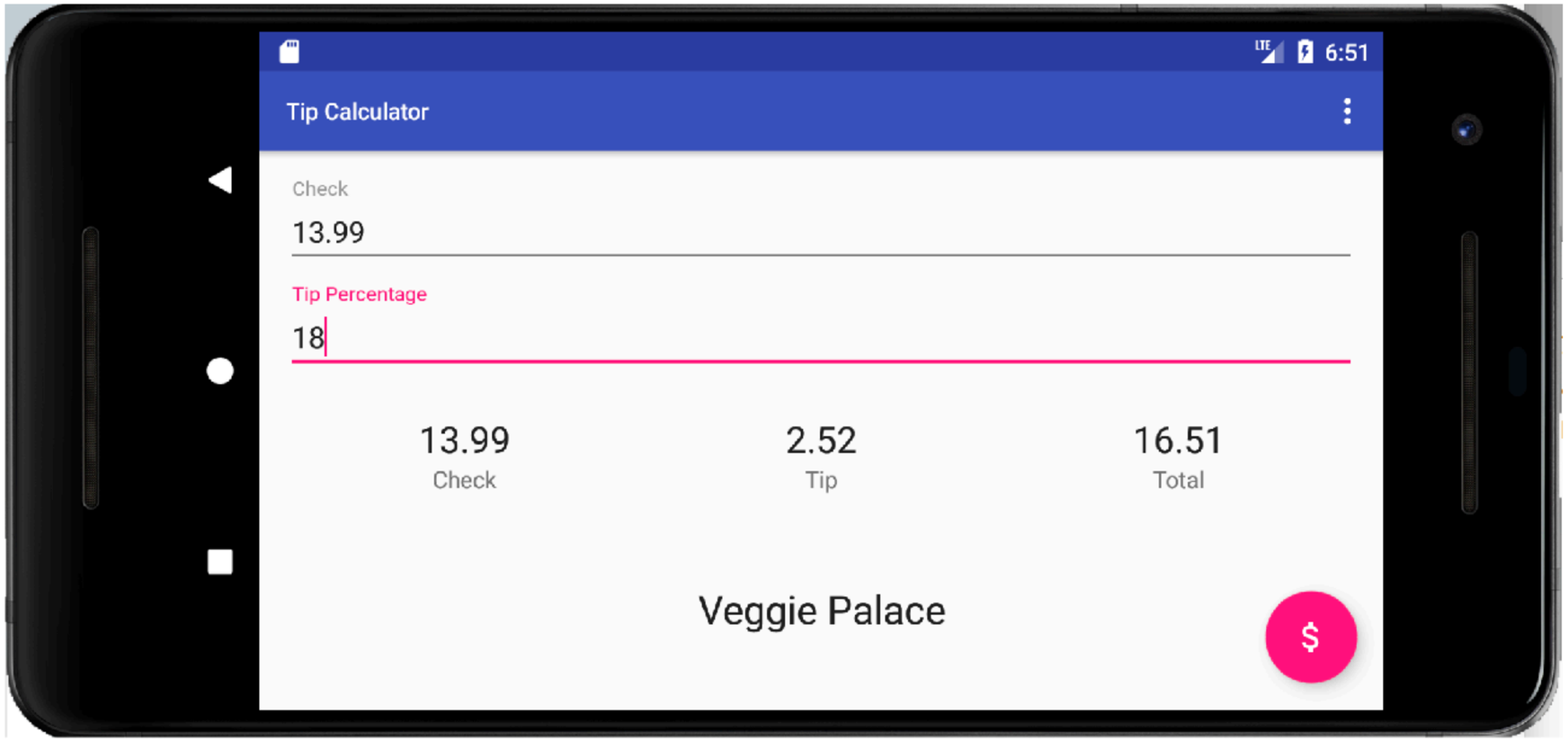
- Feel free to ask questions throughout



Learn Something?

- Tweet about it and tag #DroidconBos
- Show me at the breaks for SWAG

Data Binding



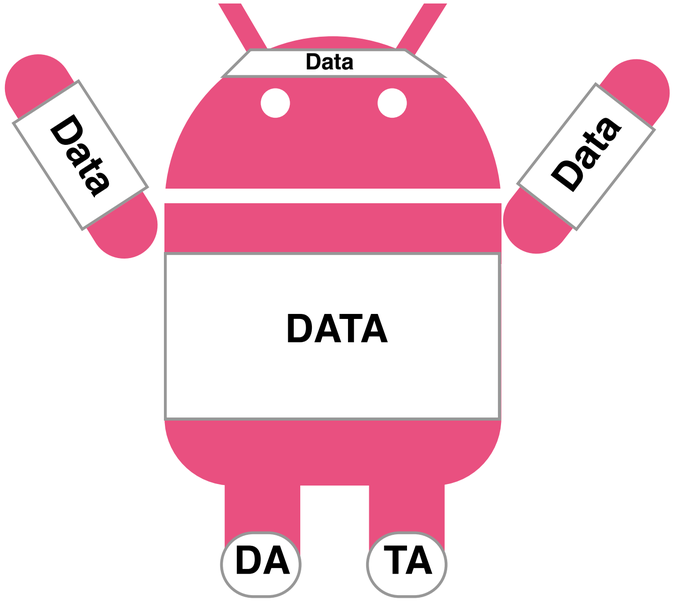
https://github.com/ericmaxwell2003/TipCalculator

Lab 0: Project Setup

- Overview of project
- Review Branches
- Clone, open and run the starter project

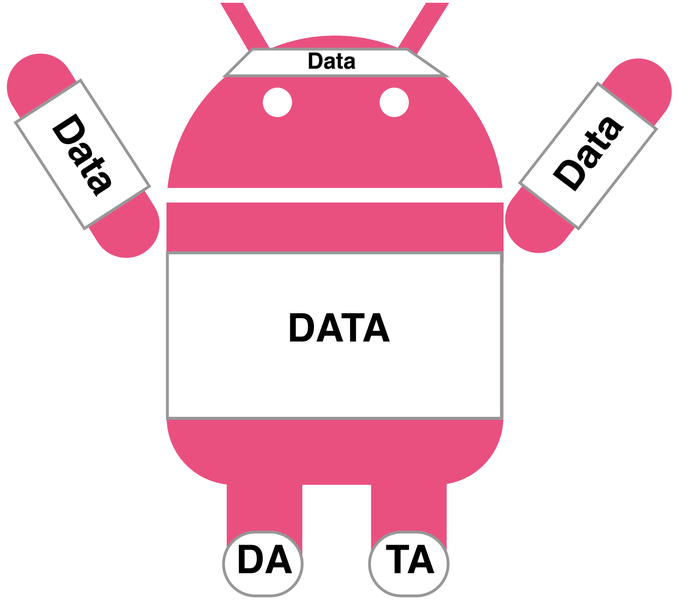
  https://github.com/ericmaxwell2003/TipCalculator
- Download Lab Instructions (**https://goo.gl/PMm6xT**)

Introduction



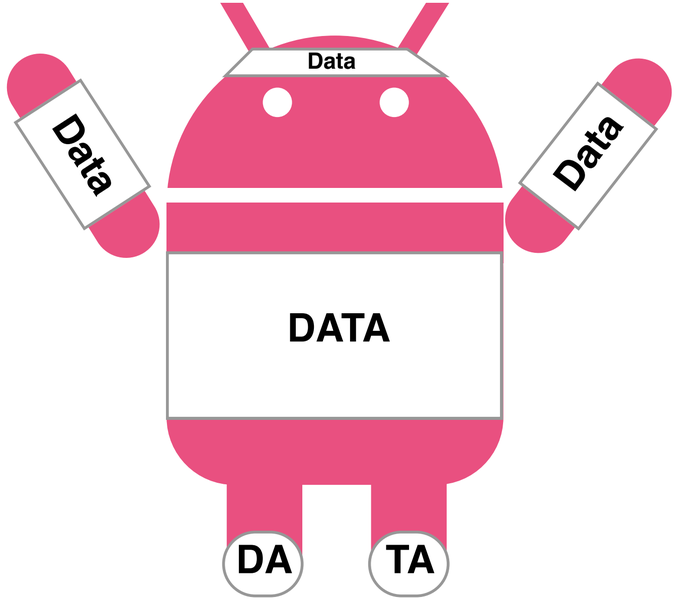
Data Binding Intro

- Support Library to **bind data** directly **to** your **views**



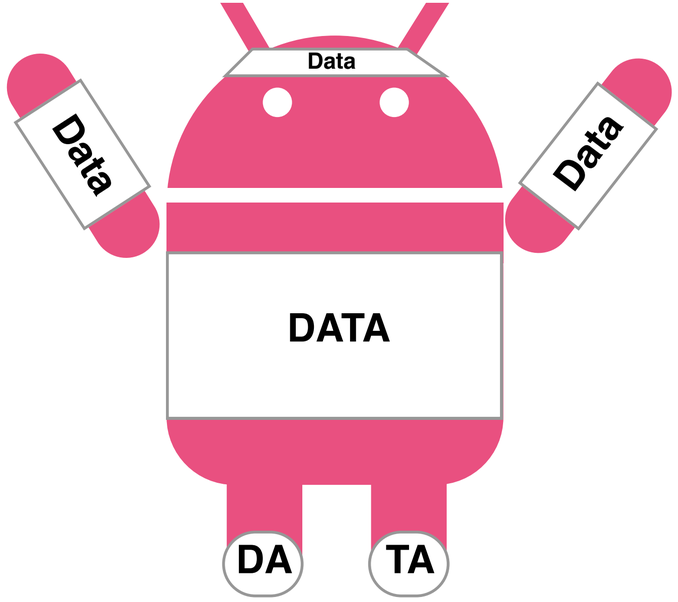
Data Binding Intro

- Support Library to **bind data** directly **to** your **views**
- **Views** can automatically **update when** underlying **data changes**



Data Binding Intro

- Support Library to **bind data** directly **to** your **views**
- **Views** can automatically **update when** underlying **data changes**
- **Views** can **connect** directly to **actions**



Enable Data Binding (Java)

To enable data binding, add this to your app module **build.gradle** file…

```
android {
    ...

    dataBinding {
        enabled = true
    }

}
```

Enable Data Binding (Kotlin)

To enable data binding, add this to your app module **build.gradle** file…

```
android {
    ...

    dataBinding {
        enabled = true
    }

}

dependencies {
    ...
     kapt “com.android.databinding:compiler:$version”
    ...

}
```

Traditional Layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   xmlns:android=“http://schemas.android.com/apk/res/android"
   ...>

         <EditText
             android:id="@+id/userNameEditText" />

         <Button
            android:onClick=‘sayHiTo'/>

</LinearLayout>
```

Data binding Layout

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

    <data>
        <variable name="viewModel" type="com.acme.ViewModel" />
    </data>

    <LinearLayout...

        <EditText
            android:id="@+id/userNameEditText"
            android:text="@{viewModel.initialUserName}" />

        <Button
            android:text="@string/say_hi"
            android:onClick='@{() -> viewModel.sayHiTo(userNameEditText)}'
            />

    </LinearLayout>
</layout>
```

Converting Existing Layouts

```
<?l version="1.0" encoding="utf-8"?>
<LinearLayout
    Convert to data binding layout
    Override Resource in Other Configuration...
    mas.android.com/apk/res/android"

        <EditText
            android:id="@+id/userNameEditText" />

        <Button
            android:onClick='sayHiTo'/>

</LinearLayout>
```

Data binding Layout

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

    <LinearLayout...

        <EditText
            android:id="@+id/userNameEditText"
            android:text="@{viewModel.initialUserName}" />

        <Button
            android:text="@string/say_hi"
            android:onClick='@{() -> viewModel.sayHiTo(userNameEditText)}'
            />

    </LinearLayout>
</layout>
```

Project

- TipCalculator ~/Projects/TipCalculator
  - .gradle
  - .idea
  - app
    - build
      - generated
        - assets
        - res
        - source
          - aidl
          - apt
          - buildConfig
          - dataBinding
          - kapt
            - debug
              - android.databinding
                - generated.callback
                - DataBinderMapper
                - DataBindingComponent
                - DynamicUtil
              - com
                - acme.tipcalculator
                - databinding
                - ActivityTipCalculatorBinding
                - ContentTipCalculatorBinding
                - SavedTipCalculationsListItemBindi
                - SayHelloBinding
                - BR

app | say_hello.xml | SayHelloBinding.java

Files under the "build" folder are generated and should not be edited.

SayHelloBinding

```
1   package com.acme.tipcalculator.databinding;
2   import ...
7   /unchecked/
8   public class SayHelloBinding extends android.databi
9
10      @Nullable
11      private static final android.databinding.ViewDa
12      @Nullable
13      private static final android.util.SparseIntArra
14      static {
15          sIncludes = null;
16          sViewsWithIds = new android.util.SparseIntA
17          sViewsWithIds.put(R.id.userNameEditText, 1)
18      }
19      // views
20      @NonNull
21      private final android.widget.LinearLayout mboun
22      @NonNull
23      public final android.widget.EditText userNameEd
24      // variables
25      // values
26      // listeners
27      // Inverse Binding Event Handlers
28
29      public SayHelloBinding(@NonNull android.databin
```

Data binding Layout

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

    <LinearLayout...

        <EditText
            android:id="@+id/userNameEditText"
            android:text="@{viewModel.initialUserName}" />

        <Button
            android:text="@string/say_hi"
            android:onClick='@{() -> viewModel.sayHiTo(userNameEditText)}'
            />

    </LinearLayout>
</layout>
```

say_hello.xml

Data binding Layout

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

    <LinearLayout...

        <EditText
            android:id="@+id/userNameEditText"
            android:text="@{viewModel.initialUserName}" />

        <Button
            android:text="@string/say_hi"
            android:onClick='@{() -> viewModel.sayHiTo(userNameEditText)}'
            />

    </LinearLayout>
</layout>
```

SayHelloBinding

Binding Class - Good Parts

```
public class SayHelloBinding extends android.databinding.ViewDataBinding  {

  // Views - Anything with an id in layout is accessible here.
  public final android.widget.EditText userNameEditText;


  // Root view in layout file
  public View getRoot() { ... }
```

Inside the Activity

```
class SayHello : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        val binding = DataBindingUtil
                .setContentView<SayHelloBinding>(this, R.layout.say_hello)

        val userEditText = findViewById<EditText>(R.id.userNameEditText)

    }

}
```

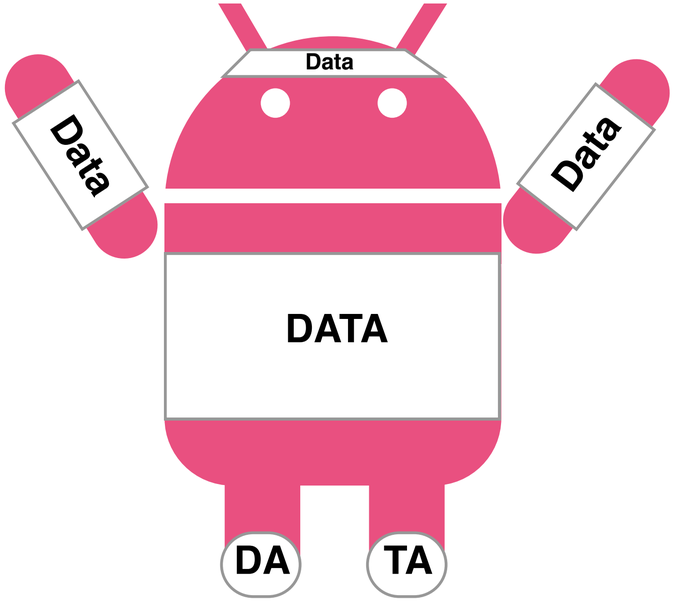
First Benefit

```
class SayHello : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)


        val binding = DataBindingUtil
                .setContentView<SayHelloBinding>(this, R.layout.say_hello)


        // val userEditText = findViewById<EditText>(R.id.userNameEditText)

        val userEditText = binding.userNameEditText

    }

}
```

Lab 1: Getting Started

- Enable Databinding
- Wrap layouts in layout tag
- Replace findViewById calls with binding Variable References
- Download Lab Instructions (https://goo.gl/**PMm6xT**)

Variables



Layout

```
<?xml version="1.0" encoding="utf-8"?>
<layout ...>

    <android.support.constraint.ConstraintLayout...>


        <TextView
            android:id="@+id/calculation_name"
            tools:text="Veggie Palace" />


        <FloatingActionButton
            android:id="@+id/calculate_fab" />


    </android.support.constraint.ConstraintLayout>

</layout>
```

Layout

```
<?xml version="1.0" encoding="utf-8"?>
<layout ...>

    <data>
        <variable name="vm" type="com.acme.tipcalculator.viewmodel.CalculatorViewModel" />
    </data>

    <android.support.constraint.ConstraintLayout...>


        <TextView
            android:id="@+id/calculation_name"
            android:text="@{vm.tipCalculation.locationName}"
            tools:text="Veggie Palace" />


        <FloatingActionButton
            android:id="@+id/calculate_fab"
            android:onClick="@{() -> vm.calculateTip()}" />


    </android.support.constraint.ConstraintLayout>

</layout>
```

ViewModel

```
class CalculatorViewModel : BaseObservable() {



    @Bindable
    var tipCalculation = TipCalculation()


    fun calculateTip() {
         ...
         tipCalculation = calculator.calculateTip(...)
         notifyPropertyChanged(BR.tipCalculation)
    }

}
```

@Bindable annotation makes this property eligible for binding

Triggers view observing **tipCalculation** to update

ViewModel

```
class CalculatorViewModel : BaseObservable() {




    @Bindable
    var tipCalculation = TipCalculation()


    fun calculateTip() {
         ...
         tipCalculation = calculator.calculateTip(...)
         notifyChange()
    }

}
```

Tells the view that **all** properties have changed

Activity

```
class TipCalculatorActivity : AppCompatActivity() {

    private lateinit var binding: ActivityTipCalculatorBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = DataBindingUtil.setContentView(this, R.layout.activity_tip_calculator)
        setSupportActionBar(binding.toolbar)

        binding.vm = CalculatorViewModel()
    }
    …
```

Connect ViewModel to View

Activity

```
class TipCalculatorActivity : AppCompatActivity() {

    private lateinit var binding: ActivityTipCalculatorBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = DataBindingUtil.setContentView(this, R.layout.activity_tip_calculator)
        setSupportActionBar(binding.toolbar)

        binding.vm = CalculatorViewModel()
    }
    …
```

```
<data>
  <variable name="vm" type="com.acme.counter.CounterViewModel" />
</data>
```

Layout variable declaration defines viewModel **name**

Activity

```
class TipCalculatorActivity : AppCompatActivity() {

    private lateinit var binding: ActivityTipCalculatorBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = DataBindingUtil.setContentView(this, R.layout.activity_tip_calculator)
        setSupportActionBar(binding.toolbar)

        binding.vm = CalculatorViewModel()
    }
    …
```

```
<data>
  <variable name="vm" type="com.acme.counter.CounterViewModel" />
</data>
```

Layout variable declaration defines viewModel **type**

Layout

```
<?xml version="1.0" encoding="utf-8"?>
<layout ...>

    <data>
        <variable name="vm" type="com.acme.tipcalculator.viewmodel.CalculatorViewModel" />
    </data>

    <android.support.constraint.ConstraintLayout...>


        <TextView
            android:id="@+id/calculation_name"
            android:text="@{vm.tipCalculation.locationName}"
            tools:text="Veggie Palace" />


        <FloatingActionButton
            android:id="@+id/calculate_fab"
            android:onClick="@{() -> vm.calculateTip()}" />


    </android.support.constraint.ConstraintLayout>

</layout>
```

Inside <data> Tag

Variables

- Define name and type of variable
- Value set by hosting Activity/Fragment

```
<data>
   <!-- (Define 0..* variables) -->
   <variable name="viewModel" type="com.acme.ViewModel" />

   <!-- (Define 0..* imports) -->
   <import type="android.view.View" />

</data>
```

Imports

- Define imports similar to imports for static classes and functions
- Import static -Utils classes, View, etc.
- Example usages

```
android:visibility="@{someCondition ? View.VISIBLE : View.GONE}"
android:text='@{StringUtils.suffixWithAwesome("Everything is ")}'
```

Layout

```
<?xml version="1.0" encoding="utf-8"?>
<layout ...>

    <data>
        <variable name="vm" type="com.acme.tipcalculator.viewmodel.CalculatorViewModel" />
    </data>

    <android.support.constraint.ConstraintLayout...>


        <TextView
            android:id="@+id/calculation_name"
            android:text="@{vm.tipCalculation.locationName}"
            tools:text="Veggie Palace" />


        <FloatingActionButton
            android:id="@+id/calculate_fab"
            android:onClick="@{() -> vm.calculateTip()}" />


    </android.support.constraint.ConstraintLayout>

</layout>
```

Expression Syntax

Variable Expressions

- “@{viewModel.foo}”
- “@{viewModel.method(`literal`)}”
- “@{Utils.staticMethod()}”

Resources

- “@{@string/stringsRes}”
- “@{@string/dollar_amount(vm.tipCalculation.checkAmount)}”

```
<string name=“dollar_amount">%1$.02f</string>
tv.text = getString(R.string.dollar_amount, vm.tipCalculation.checkAmount)
```

Expression Resolution

‘@{viewModel.value}'

Where Value will resolve to (in order of precedence)

- getValue() // Property Accessor
- value() // Method with that name
- value // Public property

Expressions are Null Safe

‘@{viewModel.value.subval}’

Expressions are null safe

- viewModel == null // Okay!
- viewModel.value == null // Bring it on!
- viewModel.value.subval == null // I eat null for breakfast!

Null Coalescing

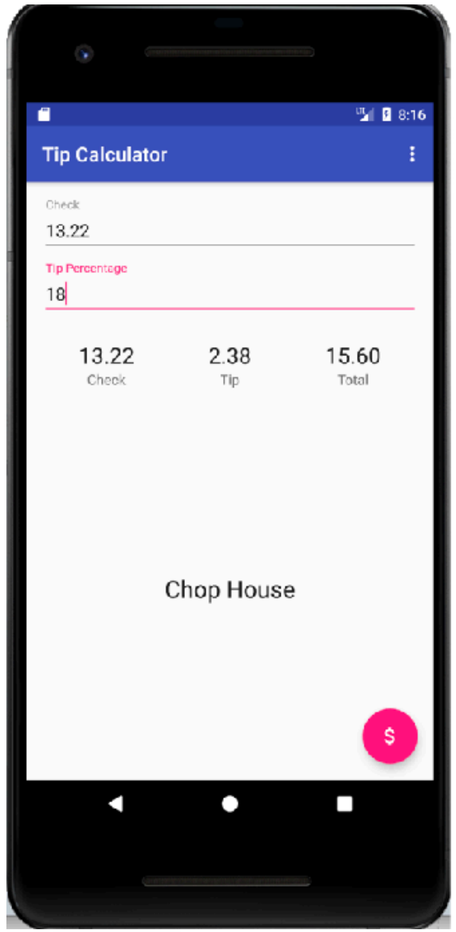
‘@{viewModel.value.subval ?? “Foo”}’

2 Way Data Binding

'@={expressions}'

2 Way Data Binding

```
<?xml version="1.0" encoding="utf-8"?>
<layout ...>

    <data>
        <variable name="vm"
                  type="com.acme.tipcalculator.viewmodel.CalculatorViewModel" />
    </data>

    <android.support.constraint.ConstraintLayout...>


    <EditText
        android:id="@+id/input_check_amount"
        android:text="@={vm.checkAmtInput}" />

    <EditText
        android:id="@+id/input_tip_percentage"
        android:text="@={vm.tipPctInput}" />


    </android.support.constraint.ConstraintLayout>

</layout>
```



Inputs can be populated by the ViewModel

2 Way Data Binding

```
class CalculatorViewModel : BaseObservable() {

    var checkAmtInput = ""
    var tipPctInput = ""

    @Bindable
    var tipCalculation = TipCalculation()

    fun calculateTip() {

        val checkAmt = checkAmtInput.toDoubleOrNull()
        val tipPctAmt = tipPctInput.toIntOrNull()

        if(checkAmt != null && tipPctAmt != null) {
            tipCalculation = calculator.calculateTip(checkAmt, tipPctAmt)
            notifyPropertyChanged(BR.tipCalculation)
        }
    }

    fun loadTipCalculation(tc: TipCalculation) {
        checkAmtInput = tc.checkAmount.toString()
        tipPctInput = tc.tipPct.toString()
        tipCalculation = tc
        notifyChange()
    }
}
```

Using inputs set by the View

Inputs can be populated by the View too

2 Way Data Binding

```
class CalculatorViewModel : BaseObservable() {

    var checkAmtInput = ""
    var tipPctInput = ""

    @Bindable
    var tipCalculation = TipCalculation()

    fun calculateTip() {

        val checkAmt = checkAmtInput.toDoubleOrNull()
        val tipPctAmt = tipPctInput.toIntOrNull()

        if(checkAmt != null && tipPctAmt != null) {
            tipCalculation = calculator.calculateTip(checkAmt, tipPctAmt)
            notifyPropertyChanged(BR.tipCalculation)
        }
    }

    fun loadTipCalculation(tc: TipCalculation) {
        checkAmtInput = tc.checkAmount.toString()
        tipPctInput = tc.tipPct.toString()
        tipCalculation = tc
        notifyChange()
    }
}
```

Set the inputs for the View later

Inputs can be populated by the View too

Normal Recycler View Adapter

```
class MyRecyclerAdapter : RecyclerView.Adapter<MyRecyclerAdapter.MyViewHolder>() {


   override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {

       val inflater = LayoutInflater.from(parent.context)

       val root = inflater.inflate(
                        R.layout.list_item,
                        parent,
                        false)

       return MyViewHolder(root)
    }


    inner class MyViewHolder(val root: View) :
            RecyclerView.ViewHolder(root) {

       fun bind(model: MyModelObject) {

           root.findViewById<TextView>(R.id.textViewFoo)?.text = model.fooValue
           root.findViewById<TextView>(R.id.textViewBar)?.text = model.barValue
           root.findViewById<TextView>(R.id.textViewMarclar)?.text = model.marclarValue
           ...
       }

    }
```

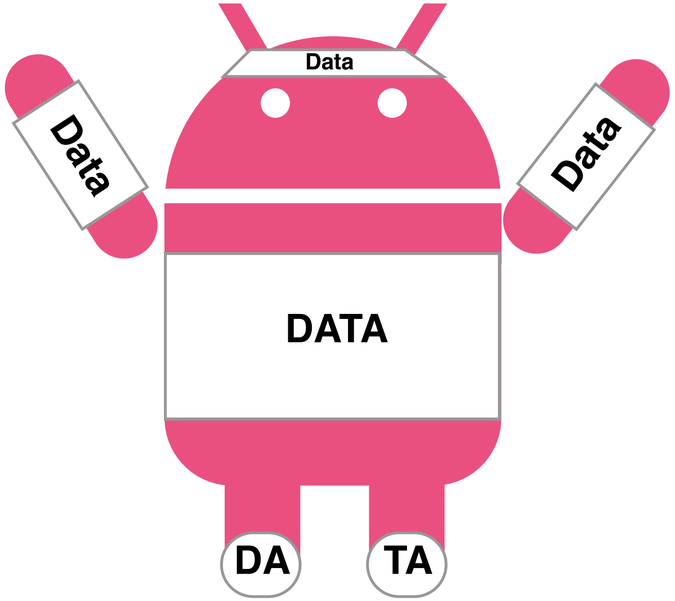
Data Binding Recycler View Adapter

```
class MyRecyclerAdapter : RecyclerView.Adapter<MyRecyclerAdapter.MyViewHolder>() {


   override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): MyViewHolder {

        val inflater = LayoutInflater.from(parent.context)

        val binding = DataBindingUtil.inflate<ListItemBinding>(
                          inflater,
                          R.layout.list_item,
                          parent,
                          false)

        return MyViewHolder(binding)
    }


    inner class MyViewHolder(val binding: ListItemBinding) :
            RecyclerView.ViewHolder(binding.root) {

        fun bind(model: MyModelObject) {
            ...
            binding.model = model
            binding.executePendingBindings()
        }

    }
```

Lab 2: Variables

- Bind to Variables + Actions
- 2-Way Data Binding
- Using Formatting Strings in Binding Expressions
- Bind RecyclerViews

Adapters & Converters



Binding Adapters

Binding Adapters allow you to add custom attributes to views

```
@BindingAdapter(value = ["app:imageUrl"])
fun loadImage(view: ImageView, url: String) {
    Picasso.get()
            .load(url)
            .into(view)
}
```

```
<ImageView
    android:id="@+id/imageView"
    tools:src="@drawable/design_time_sample_image"
    app:imageUrl='@{"http://acme.com/images/any.png"}' />
```

Binding Adapters

```
@BindingAdapter(value = ["app:imageUrl"])
fun loadImage(view: ImageView, url: String) {
    Picasso.get()
            .load(url)
            .into(view)
}
```

```
<ImageView
    android:id="@+id/imageView"
    tools:src="@drawable/design_time_sample_image"
    app:imageUrl='@{"http://acme.com/images/any.png"}' />
```

Adds **app:imageUrl** attribute

Binding Adapters

```
@BindingAdapter(value = ["app:imageUrl"])
fun loadImage(view: ImageView, url: String) {
    Picasso.get()
            .load(url)
            .into(view)
}
```

```
<ImageView
    android:id="@+id/imageView"
    tools:src="@drawable/design_time_sample_image"
    app:imageUrl='@{"http://acme.com/images/any.png"}' />
```

Applies to **ImageView**

Binding Adapters

```
@BindingAdapter(value = ["app:imageUrl"])
fun loadImage(view: View, url: String) {
    Picasso.get()
            .load(url)
            .into(view)
}
```

```
<ImageView
    android:id="@+id/imageView"
    tools:src="@drawable/design_time_sample_image"
    app:imageUrl='@{"http://acme.com/images/any.png"}' />
```

Applies to all **Views**

Binding Adapters

Binding Adapters allow you to add custom attributes to views

```
@BindingAdapter(value = ["app:imageUrl"])
fun loadImage(view: TextView, url: String) {



}
```

```
<ImageView
    android:id="@+id/imageView"
    tools:src="@drawable/design_time_sample_image"
    app:imageUrl='@{"http://acme.com/images/any.png"}' />
```

Cannot find the setter for attribute 'app:imageUrl' with parameter type java.lang.String on android.widget.ImageView

Binding Adapters

```
@BindingAdapter(value = ["app:imageUrl"])
fun loadImage(view: View, url: String) {
    Picasso.get()
            .load(url)
            .into(view)
}
```

```
<ImageView
    android:id="@+id/imageView"
    tools:src="@drawable/design_time_sample_image"
    app:imageUrl='@{"http://acme.com/images/any.png"}' />
```

Value of expression
Must be an expression!

Binding Adapters

```
@BindingAdapter(value = ["app:imageUrl", "app:placeholder"])
fun loadImage(view: ImageView, url: String, placeholder: Drawable) {
    Picasso.get()
            .load(url)
            .placeholder(placeholder)
            .into(view)
}

<ImageView
    android:id="@+id/imageView"
    tools:src="@drawable/design_time_sample_image"
    app:imageUrl='@{“http://acme.com/images/any.png"}'
    app:placeHolder="@{@drawable/ic_launcher_background}"
 />
```

Binding Converters

Instead of this…

```
<ImageView
    android:id="@+id/imageView"
    app:imageUrl='@{“http://acme.com/images/any.png"}'
    app:placeHolder="@{@drawable/ic_launcher_background}"
    android:visibility=“@{viewModel.showImage ? View.VISIBLE : View.GONE}” />
```

Converters

…you can do this

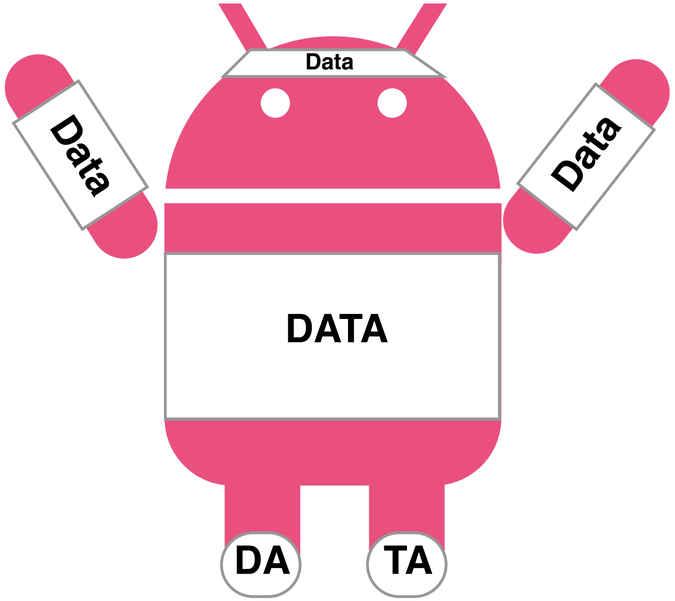
```
<ImageView
    android:id="@+id/imageView"
    app:imageUrl='@{“http://acme.com/images/any.png"}'
    app:placeHolder="@{@drawable/ic_launcher_background}"
    android:visibility="@{viewModel.showImage}" />

@BindingConversion
public static int convertBooleanToVisibility(boolean expression) {
    return expression ? View.VISIBLE : View.GONE;
}
```

Applies for all attributes where **boolean expression** is given and **int** is **required**

Lab 3: Adapters *+ Converters*

- Use Binding Adapter on TextView
- Adapter will hide the view when there is no text
- Animate the transition to show or hide

ViewModel + LiveData



Current ViewModel

```
class TipCalculatorActivity : AppCompatActivity() {

    private lateinit var binding: ActivityTipCalculatorBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = DataBindingUtil.setContentView(this, R.layout.activity_tip_calculator)
        setSupportActionBar(binding.toolbar)

        binding.vm = CalculatorViewModel()
    }
    …
```

Connect ViewModel to View

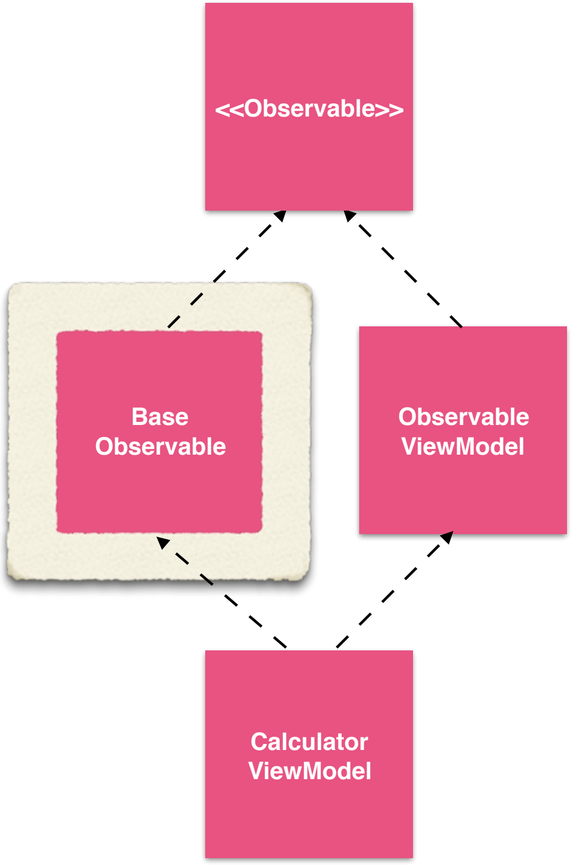
AC ViewModel

```
class TipCalculatorActivity : AppCompatActivity() {

    private lateinit var binding: ActivityTipCalculatorBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = DataBindingUtil.setContentView(this, R.layout.activity_tip_calculator)
        setSupportActionBar(binding.toolbar)

        binding.vm = ViewModelProviders.of(this).get(CalculatorViewModel::class.java)

    }
    …
```

Connect ViewModel to View

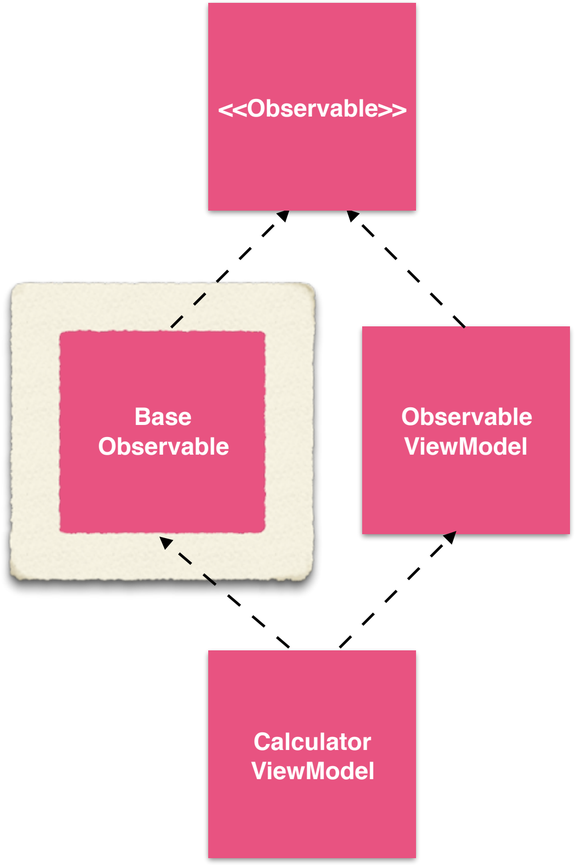
Data Binding Magic Base Class

```
class CalculatorViewModel : BaseObservable() {



    @Bindable
    var tipCalculation = TipCalculation()


    fun calculateTip() {
         ...
         tipCalculation = calculator.calculateTip(...)
         notifyPropertyChanged(BR.tipCalculation)
    }

}
```



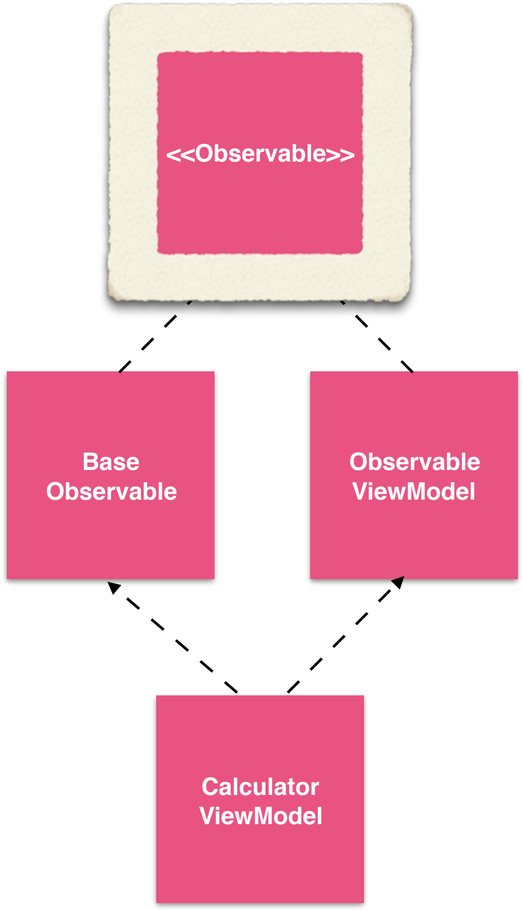
AC Magic Base Class :-(

```
class CalculatorViewModel : ViewModel() BaseObservable() {



    @Bindable
    var tipCalculation = TipCalculation()


    fun calculateTip() {
        ...
        tipCalculation = calculator.calculateTip(...)
        notifyPropertyChanged(BR.tipCalculation)
    }

}
```



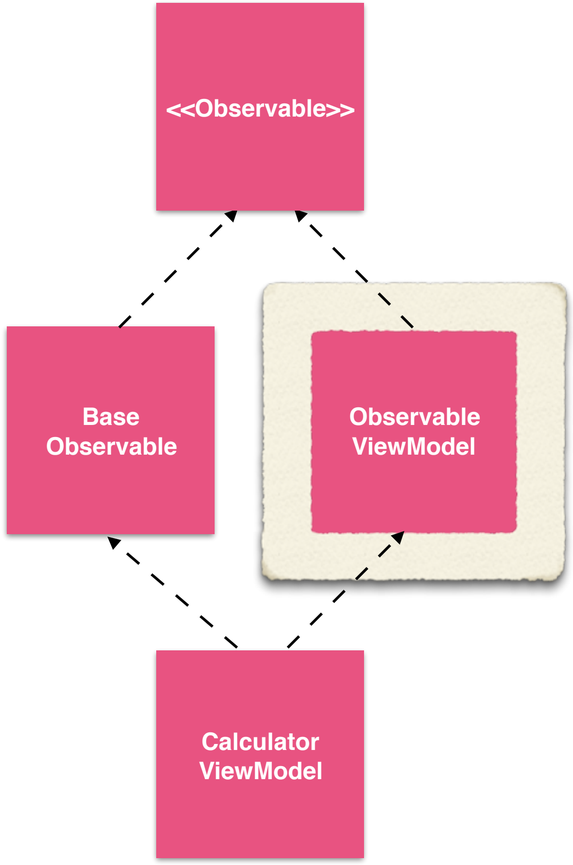
Making Data Observable

```
public interface Observable {
    void addOnPropertyChangedCallback(...);
    void removeOnPropertyChangedCallback(...);
    public abstract static class OnPropertyChangedCallback {
        public abstract void onPropertyChanged(...);
    }
}
```



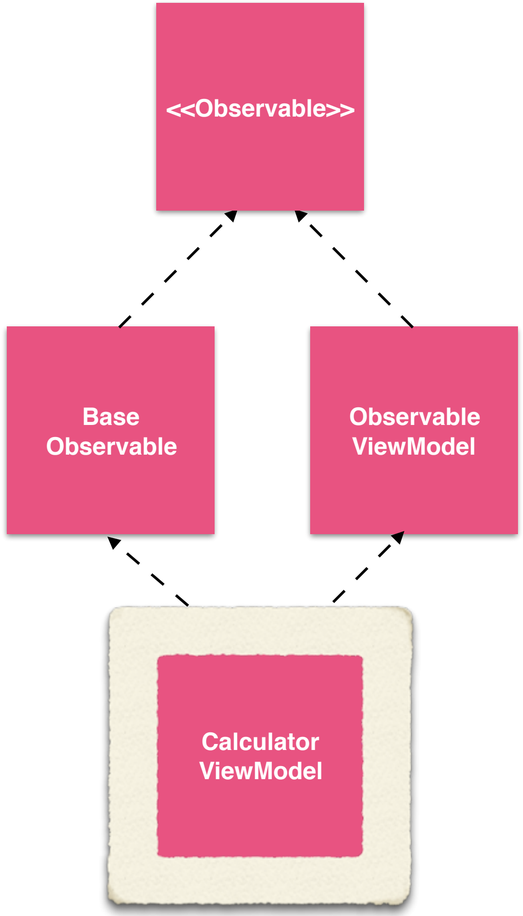
How to make an Observable ViewModel

```
/**
 * Bridge class which does everything android.databinding.BaseObservable
does plus
 * extends Android Architecture Components ViewModel
 */
public abstract class ObservableViewModel
              extends ViewModel
              implements Observable {

   ...

}
```



How to make an Observable ViewModel

```
class CalculatorViewModel : BaseObservableViewModel() {



    @Bindable
    var tipCalculation = TipCalculation()


    fun calculateTip() {
        ...
        tipCalculation = calculator.calculateTip(...)
        notifyPropertyChanged(BR.tipCalculation)
    }

}
```



Observing Live Data

```
// Inside Calculator View Model
fun loadSavedTipCalculations() : LiveData<List<TipCalculation>> {
    return repository.loadSavedTipCalculations()
}
```

Observing Live Data

```
// Inside Calculator View Model
fun loadSavedTipCalculations() : LiveData<List<TipCalculation>> {
    return repository.loadSavedTipCalculations()
}

// Inside Fragment/Activity hosting a RecyclerView
calculatorViewModel.loadSavedTipCalculations().observe(this, Observer { tips ->
    if(tips != null) {
        recyclerAdapter.updateList(tips)
    }
})
```

Observing Live Data

```
// Inside Calculator View Model
fun loadSavedTipCalculations() : LiveData<List<TipCalculation>> {
    return repository.loadSavedTipCalculations()
}

// Inside Fragment/Activity hosting a RecyclerView
calculatorViewModel.loadSavedTipCalculations().observe(this, Observer { tips ->
    if(tips != null) {
        recyclerAdapter.updateList(tips)
    }
})

// Inside RecyclerAdapter
fun updateList(updates: List<TipCalculation>) {
    calculations.clear()
    calculations.addAll(updates)
    notifyDataSetChanged()
}

override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    holder.bind(savedTipCalculations[position])
}

inner class ViewHolder(val binding: ItemBinding) :RecyclerView.ViewHolder(binding.root) {
    fun bind(tipCalc: TipCalculation) {
        binding.tipCalculation = tipCalc
        binding.executePendingBindings()
    }
}
```

Lab 4: ViewModel + LiveData

- Create an ObservableViewModel Base Class
- Use in the Activities + Fragments

Thank you!

Eric Maxwell
emaxwell@bignerdranch.com
github.com/ericmaxwell2003
@emmax