Mobile Applications Development
CSCI 448
Lecture 01

My First App: Hello Android!

Previously in CSCI 448

- Syllabus overview
- Versions / History of Android
- Device & Version Fragmentation

Learning Outcomes For Today

- Describe the different high-level components of an app
- Discuss the evolution of Android development and architecture
- List the components of MVVM

- Create your first app

On Tap For Today

- Final Project
- Modern Android
  - Java → Kotlin
  - MVC & View → MVVM & Compose
- Practice
  - Lab00A

On Tap For Today

- Final Project
- Modern Android
  - Java → Kotlin
  - MVC & View → MVVM & Compose
- Practice
  - Lab00A

Final Project

- It starts...now!
- https://cs-courses.mines.edu/csci448/homework/fp.html

- Step 1: Elevator Pitch

On Tap For Today

- Final Project
- Modern Android
  - Java → Kotlin
  - MVC & View → MVVM & Compose
- Practice
  - Lab00A

Version History

- KitKat – “Ok, Google”
- Lollipop – Material Design
- Marshmallow – Split Screen
- Nougat – Quick Switch Apps (& emojis)
- Oreo – Picture in Picture (more emojis)
- Pie – Dark Mode (more emojis)
- 10 – Improved permission options
- 11 – Native screen recording
- 12 – UI Updates
- 13 – MaterialYou

On Tap For Today

- Final Project
- Modern Android
  - Java → Kotlin
  - MVC & View → MVVM & Compose
- Practice
  - Lab00A

On Tap For Today

- Final Project
- Modern Android
  - Java → Kotlin
  - MVC & View → MVVM & Compose
- Practice
  - Lab00A

History

- Created in 2011 for in house work at JetBrains
- Public release in 2012
- Built on top of JVM
  - Fully interoperable with Java

- In 2017, Google adopted Kotlin for Android development

Some Benefits of Kotlin

- Semi-colons are optional
- Can infer data type
- Null safe
- Lambda expressions replace Single Abstract Method classes (SAMs)
  - Reduces boiler plate!!

- Will learn more in the wonderful PKS!
  (Paone Kotlin Series)

Java Class

```
01 class Product {
02   private Integer mId;
03   private String mName;
04   Product( Integer id, String name ) {
05     mId = id;
06     mName = name;
07   }
08   public Integer getId() { return mId; }
09   public void setId( Integer id ) { mId = id; }
10   public String getName() { return mName; }
11   public void setName( String name ) { mName = name; }
12   public boolean equals( Product p ) { ... }
13   public Integer hashCode() { ... }
14   public String toString() { ... }
15   public Product copy(...) { ... }
16 }
```

Kotlin Data Classes

- Equivalent Kotlin class

```
01 data class Product( var id : Int, var name : String )
```

SAMs

```java
// Java
nextButton.setOnClickListener( new View.OnClickListener() {
  @Override
  public void onClick( View view ) {
    // do something
  }
});

// Kotlin
nextButton.setOnClickListener {
  // do something
}
```

On Tap For Today

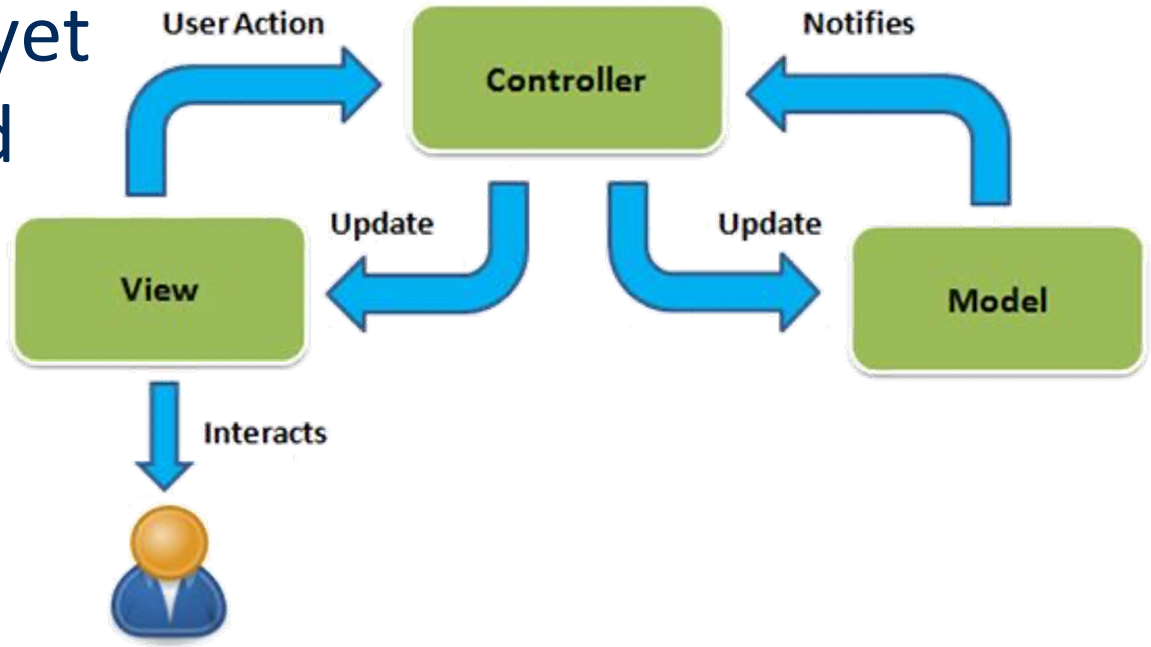
- Final Project
- Modern Android
  - Java → Kotlin
  - MVC & View → MVVM & Compose
- Practice
  - Lab00A

Three Tier Architecture

- Presentation Layer (UI Logic)
- Domain Layer (Business Logic)
- Model Layer (Data Logic)

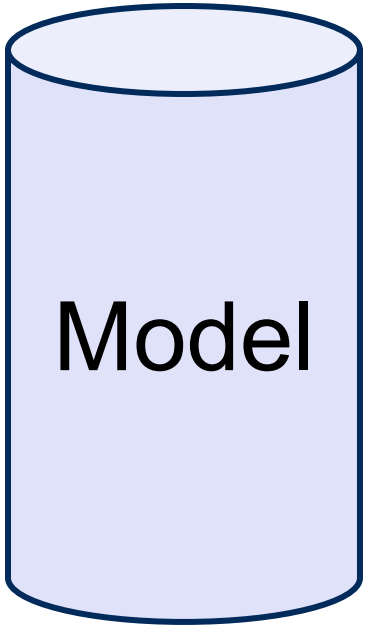
Model-View-Controller (MVC)

- Architecture paradigm for user interfaces
- Three disjoint yet interconnected components
  - Model
  - View
  - Controller



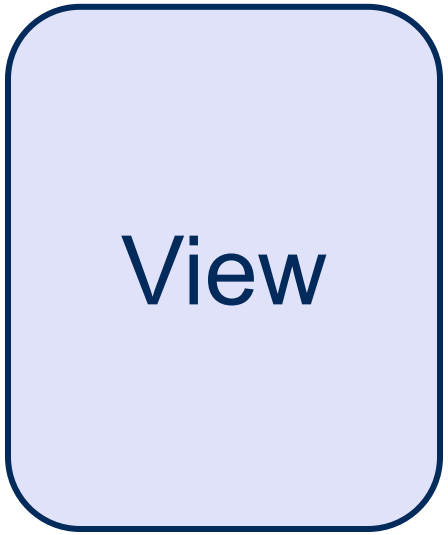
Model

- Entities
  - Manages the data (load/persist)
  - Contains the business logic
  - Dictates rules of operation
- Modified via: Controller
- Updates: Controller



View

- Output / Visual Representation
  - Chart, diagram, form, etc.
  - Multiple views can exist for same information
- Modified via: Controller
- Updates: Controller



Layouts (.xml)

- Contain the XML representation of the corresponding view

activity_main.xml

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

    <TextView
        android:id="@+id/textview_question"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:padding="24dp" />

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/true_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="True" />

        <Button
            android:id="@+id/false_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="False" />
    </LinearLayout>

    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal">

        <Button
            android:id="@+id/previous_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Previous"
            android:drawableLeft="@drawable/arrow_left"
            android:drawablePadding="4dp" />

        <Button
            android:id="@+id/next_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Next"
            android:drawableRight="@drawable/arrow_right"
            android:drawablePadding="4dp" />
    </LinearLayout>
```

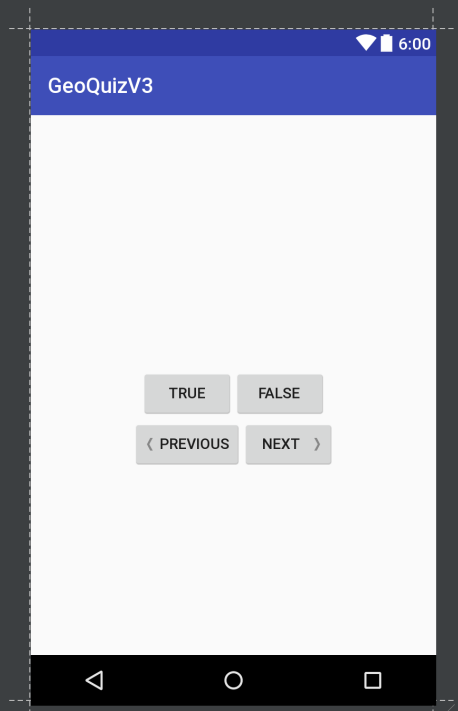
Design Text

Preview

Nexus 4 23 AppTheme Language

Palette

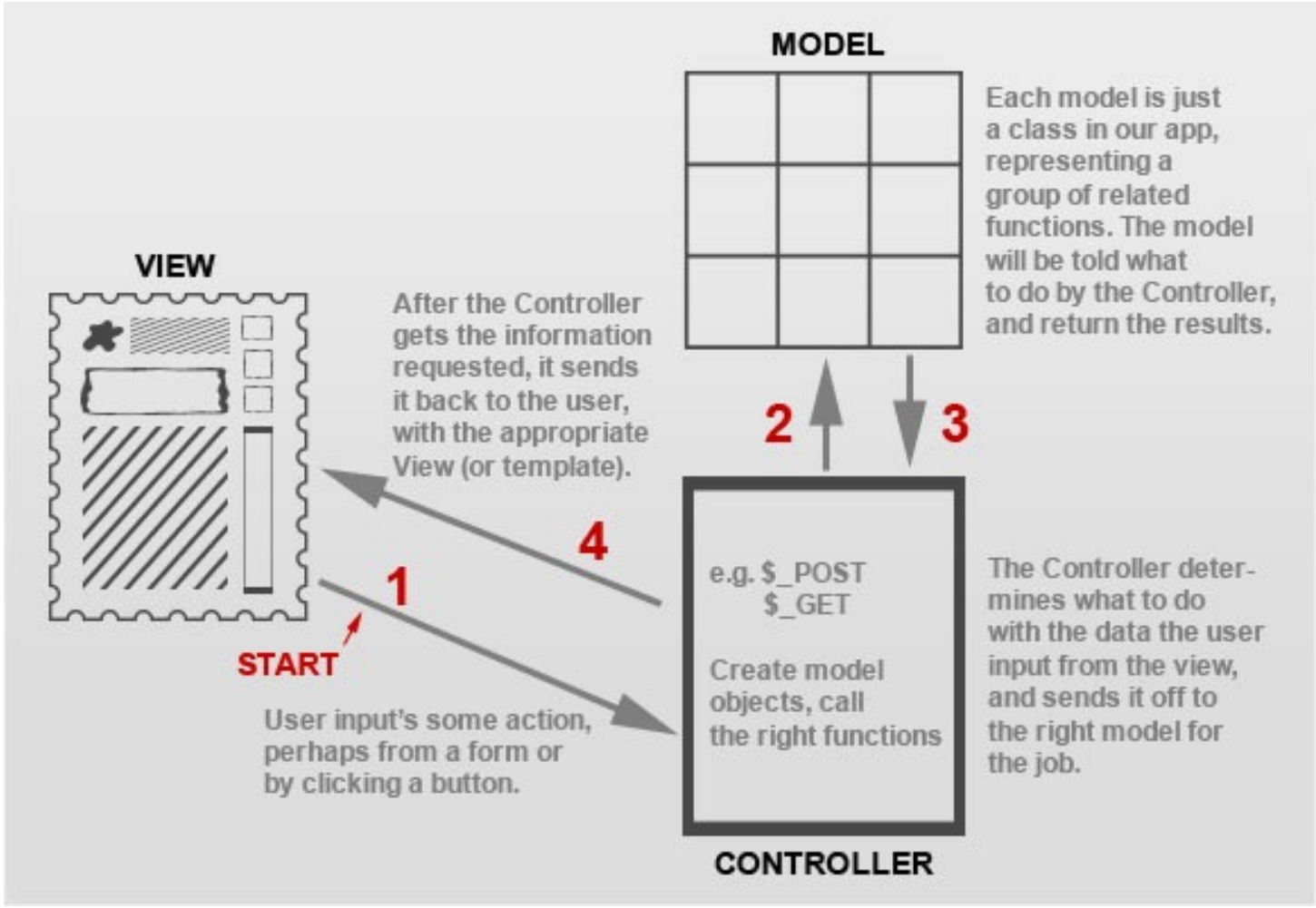
99%



Controller

- Respond to events & converts to application logic
  - Receives user input from view
  - Sends commands to model to update state
  - Receives change of state from model
  - Packages model state for view
  - Sends commands to view to update output
- Modified via: View & Model
- Updates: View & Model

MVC Interactions V2



Activity Class (.kt)

- Sets what layout to use for the activity
- Get reference to View
- Wire up events

```
1   package edu.mines.pizzaparty4k
2
3   import ...
7
8   class MainActivity : AppCompatActivity() {
9
10      override fun onCreate(savedInstanceState: Bundle?) {
11          super.onCreate(savedInstanceState)
12          setContentView(R.layout.activity_main)
13
14          val numPizzasTextView: TextView = findViewById( R.id.num_pizzas_text_view )
15          numPizzasTextView.setOnClickListener{ it: View!
16              Toast.makeText( context: this@MainActivity, text: "Make Toast!", Toast.LENGTH_SHORT ).show()
17          }
18      }
19  }
20
```

MVC Tightly Bound

- And it's relative MVP

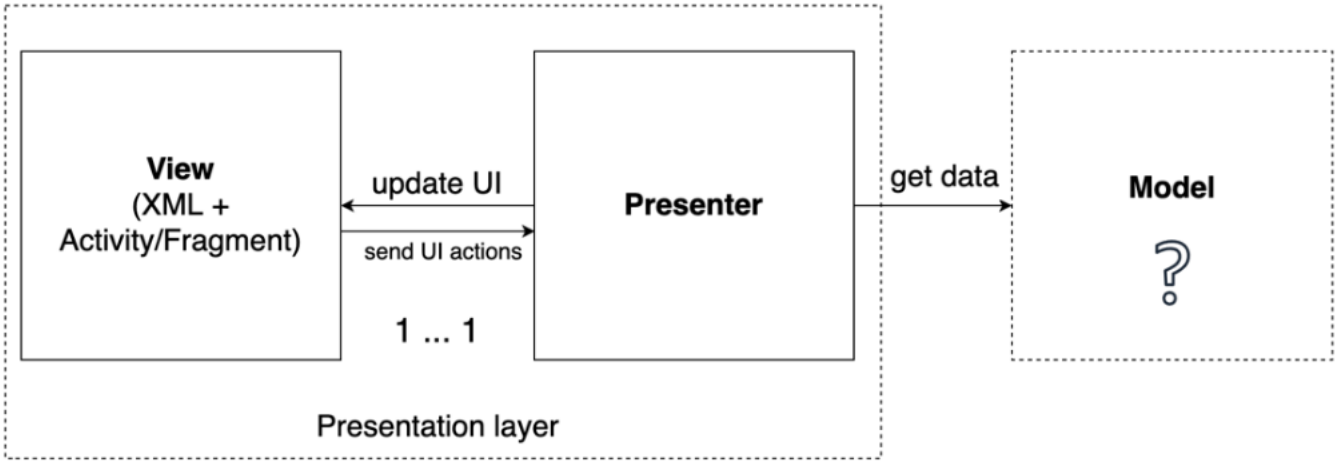


Figure 7.3 – Presentation layer in the MVP pattern

Prior UI Framework

- Android Framework was “old”
  - ~15 years
  - Written in Java
- Fragments 10+ years old
  - Became “micro activities”
  - Activities & Fragments 1:1
- Single Activity design encouraged 2018
- 2019: Fragments should have been views from the beginning

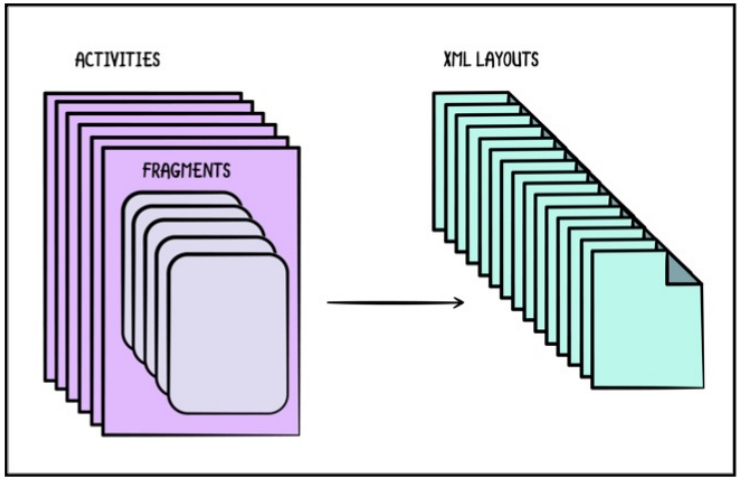
Problems with MVC & View Framework

- Scalability
- Separation of Concerns
- Inheritance instead of composition
- Imperative Programming

Problems with MVC & View Framework

- Scalability
- Separation of Concerns
- Inheritance instead of composition
- Imperative Programming

Non-Scalable View System



Non-scaleable Layout System

- Hello World in a fragment
  1. MainActivity.kt
  2. activity_main.xml
  3. MyFragment.kt
  4. fragment_my.xml
- Dynamic UI components add even more files!

Problems with MVC & View Framework

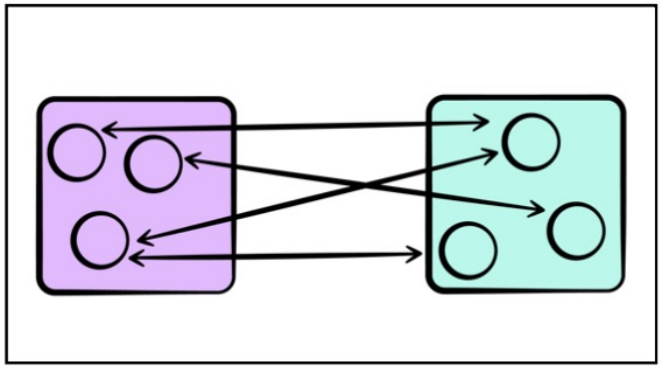
- Scalability
- Separation of Concerns
- Inheritance instead of composition
- Imperative Programming

Separation of Concerns

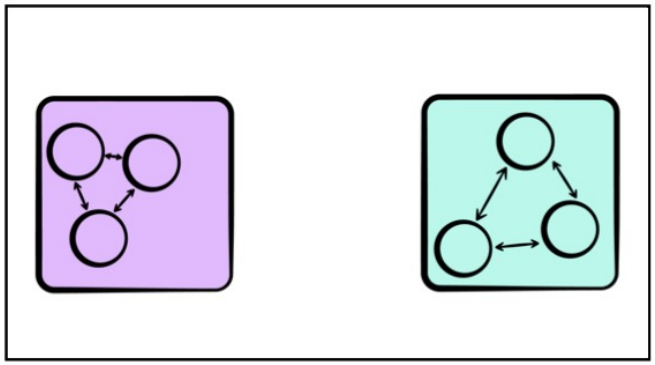
- Distinct sections each addressing a distinct concern
  - Such as separating business logic from UI logic
- App comprised of modules
  - Reduce coupling
  - Increase cohesion

Coupling & Cohesion

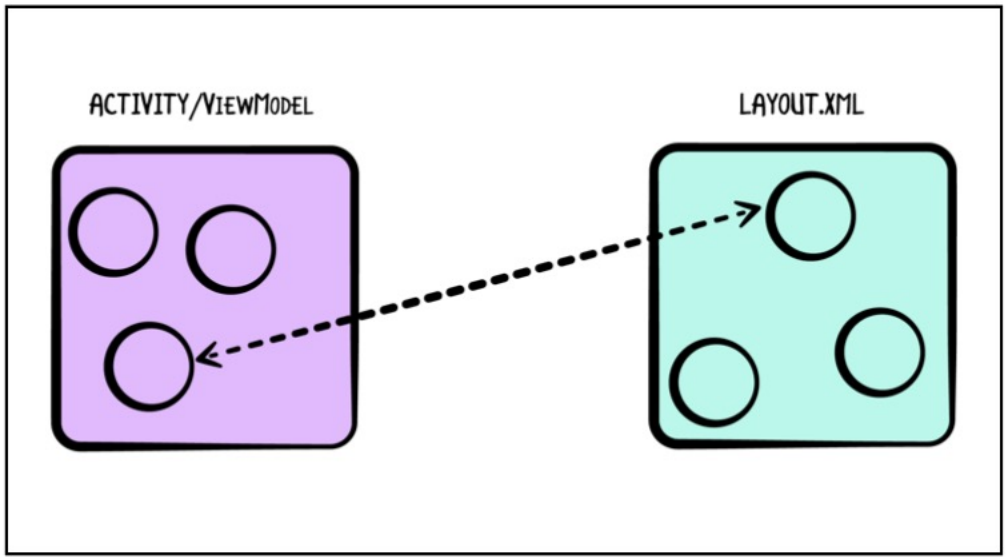
- Coupling: dependencies between modules



- Cohesion: dependencies within a module



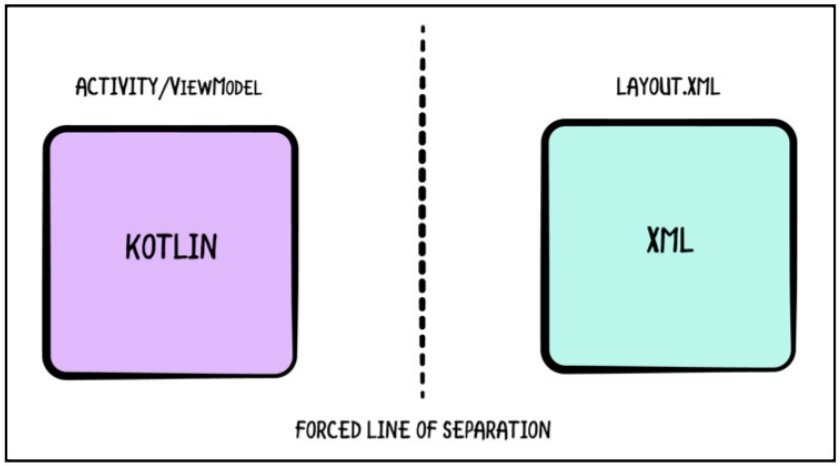
View UI Coupling & Cohesion



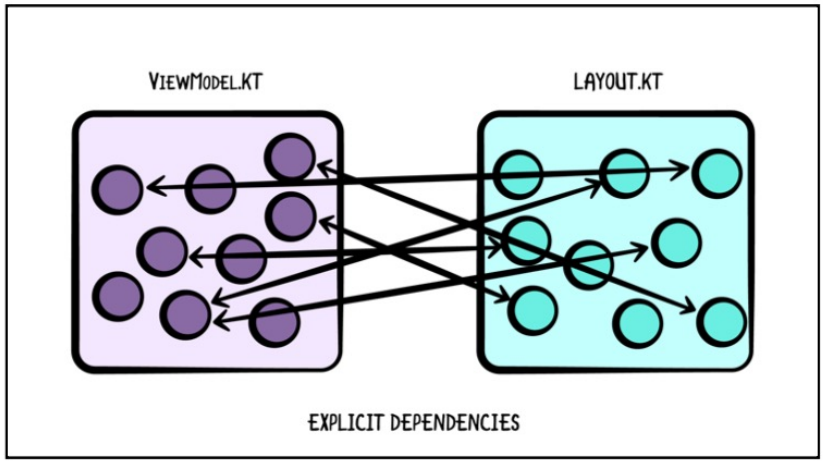
Modules when Implementing Android UI

- Should be cohesive, but can't be due to language difference
  - Dependency is implicit, modules are closely related

View UI Separation of Concern



Forced Line of Separation of Concerns



Problems with MVC & View Framework

- Scalability
- Separation of Concerns
- Inheritance instead of composition
- Imperative Programming

Consider Button Classes

1. Text Buttons

java.lang.Object
↳ android.view.View
↳ android.widget.TextView
↳ android.widget.Button

2. Image Buttons

java.lang.Object
↳ android.view.View
↳ android.widget.ImageView
↳ android.widget.ImageButton

3. Text & Image Button?
   - Which class to extend?

View Interaction

- Views aren't static
- Views expose listeners to handle UI events
- Parent View class handles all possible scenarios
- Current View.java class

Lines of Code in View.java

Problems with MVC & View Framework

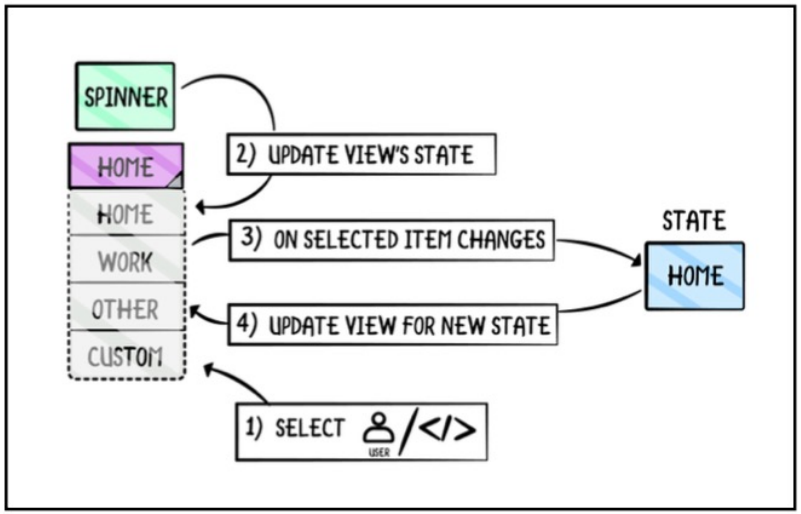
- Scalability
- Separation of Concerns
- Inheritance instead of composition
- Imperative Programming

View Interaction

- Views aren't static
- Views expose listeners to handle UI events
- Concerns with Data Flow:
  1. What is the source of truth?
  2. Who owns the state?
  3. Who updates the state?

Spinner Example

1. onSelectedItemChanged() is called after the value changes
2. Spinner updates its state and tells you its state has changed
3. You update the model to reflect the new state
4. You update UI to reflect model state change



Spinner and State Management

State Problems

- View cannot represent Model state if UI also owns and manages its own state
  - We capture the event, then update the View & Model to the new state
- Therefore, we build UIs to model how changes should occur
  - This programing style is known as **Imperative Programming**

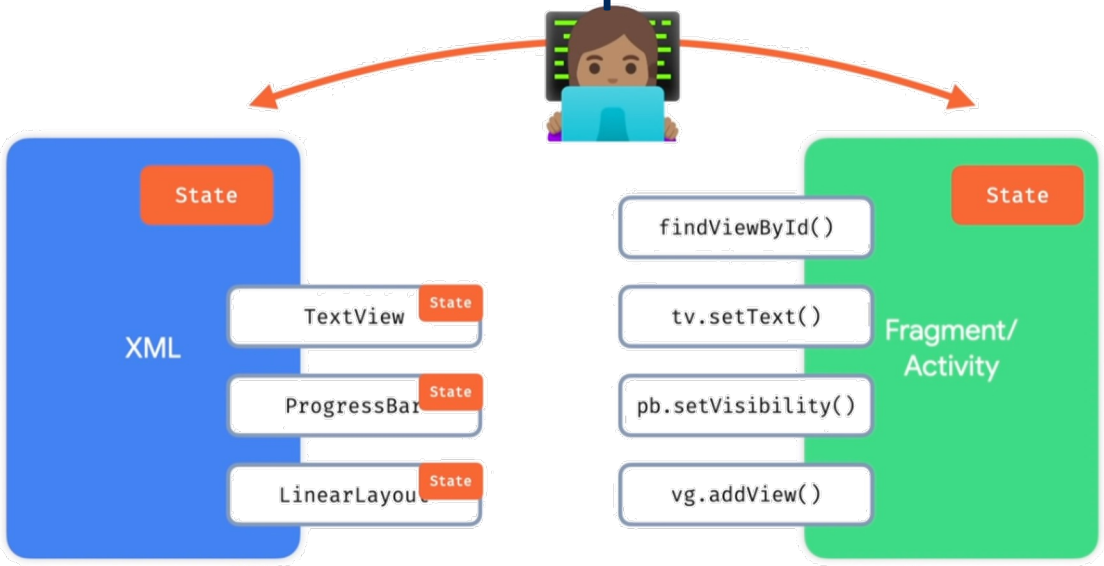
Imperative Programming

- Uses statements in the form of functions to change a program's state
- Focuses on describing **how** a program should operate

- Procedural Programming is a type of Imperative Programming
  - Object-Oriented Programming typically follows the Procedural style as well

Problem of View System

- Imperative UI System
- UI Views are mutable
- Data flow
- Favors inheritance instead of composition
- Doesn't scale



MVVM Architecture

- View Model prepares Model data for View
- V M decoupled from V
  - VM has no reference to V
  - V observes VM state

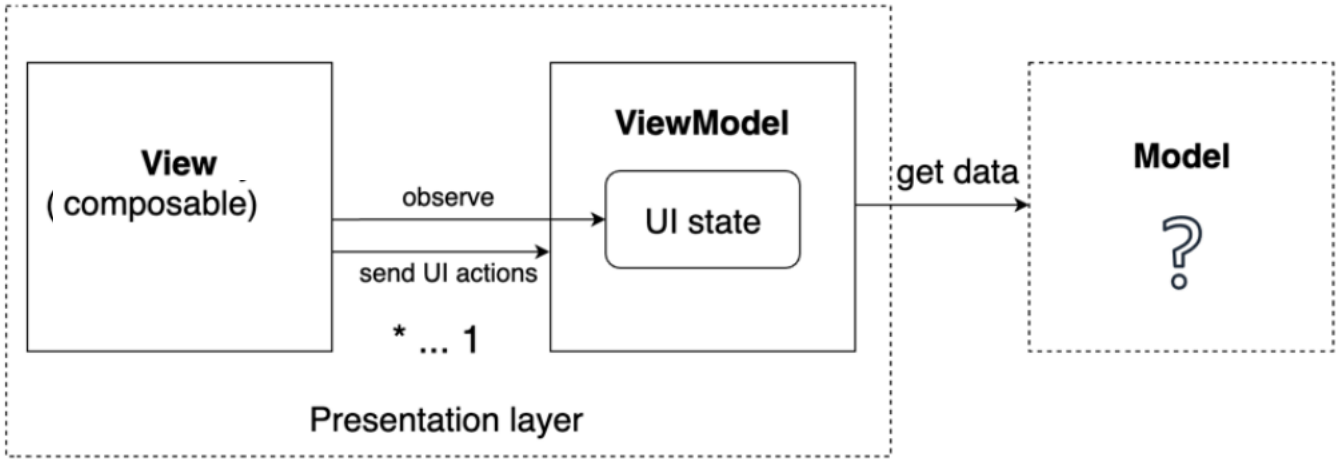


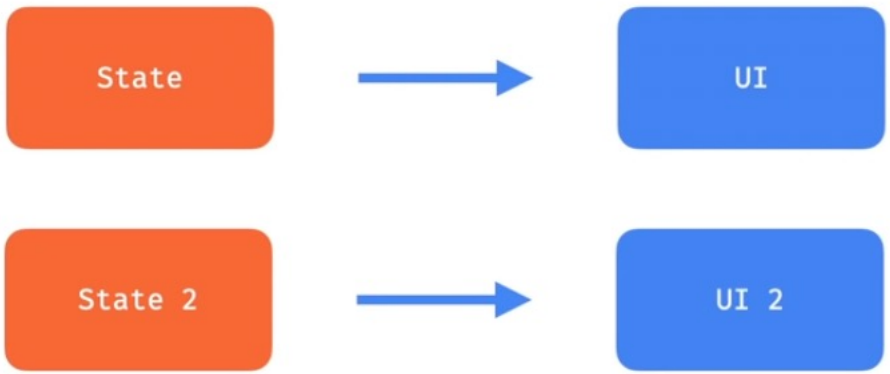
Figure 7.4 – Presentation layer in the MVVM pattern

M - V - V M

- V M observes M for data changes
  - M notifies V M when the data changes
- V observes V M for data changes
  - VM notifies the V when the M changes
- V notifies V M when events occur
  - V M notifies the M when V event occurs
- V M (data producer) has no knowledge of the V (data consumer)

Compose

- UI is immutable: there are no objects
  - But UI is dynamic
  - Different inputs → Different UI
- UI is **idempotent**

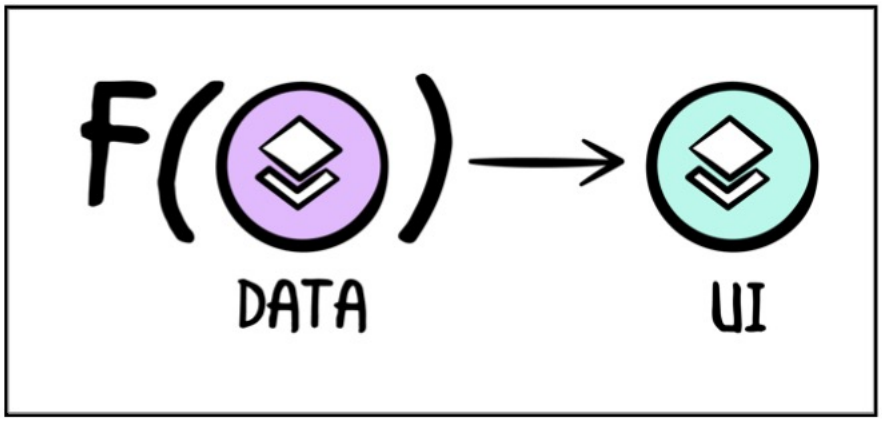


Composables

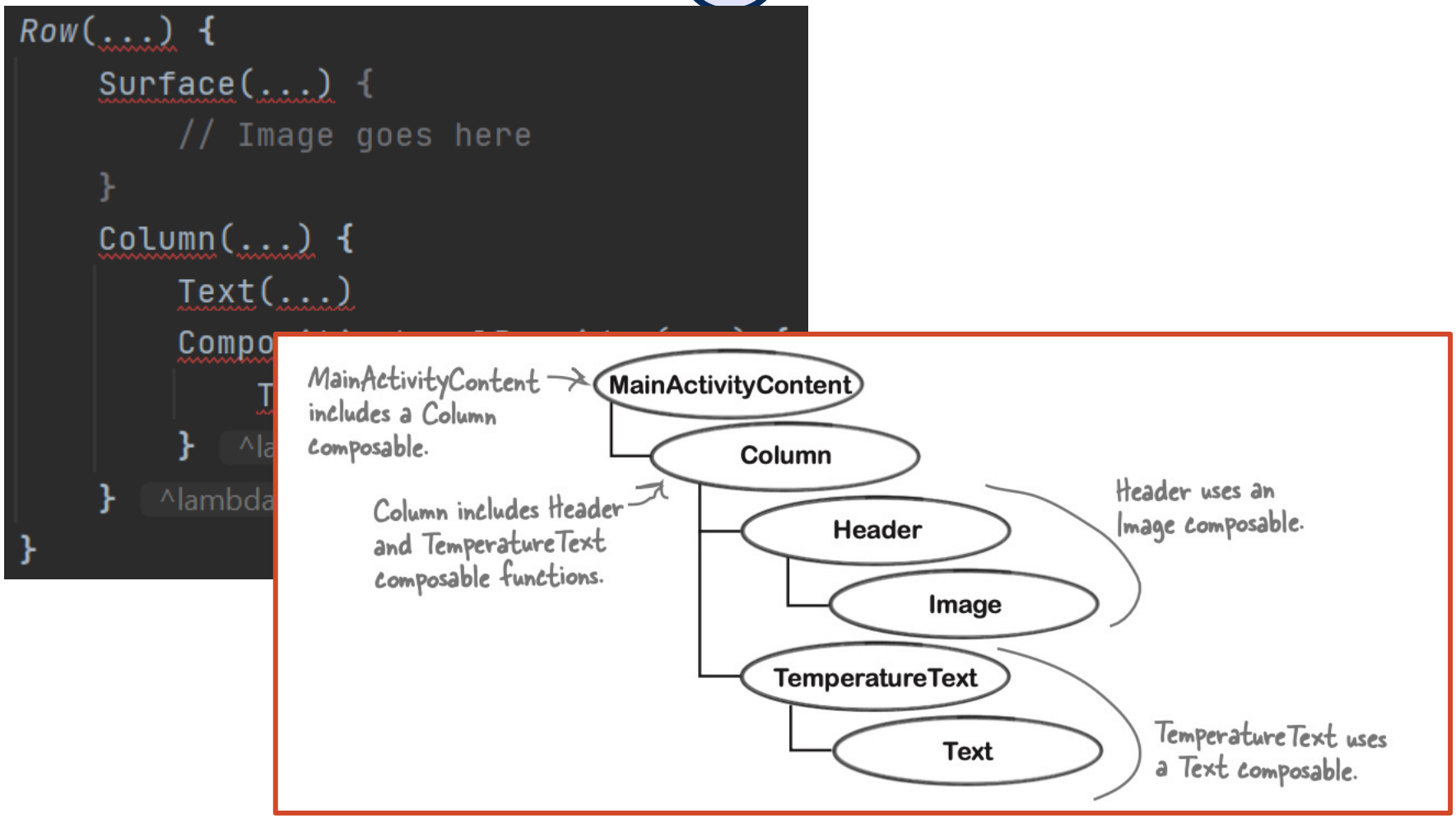
- Functions that take data (state) as parameters and emit UI
  - Composable is immutable

```
@Composable
fun Greeting(name: String) {
  Text(text = "Hello $name!")
}
```

  - Function is idempotent without side effects (no global variables)
  - Functions run in parallel – need to be thread safe



Style Structure



Design Principle #1: Composition Over Inheritance

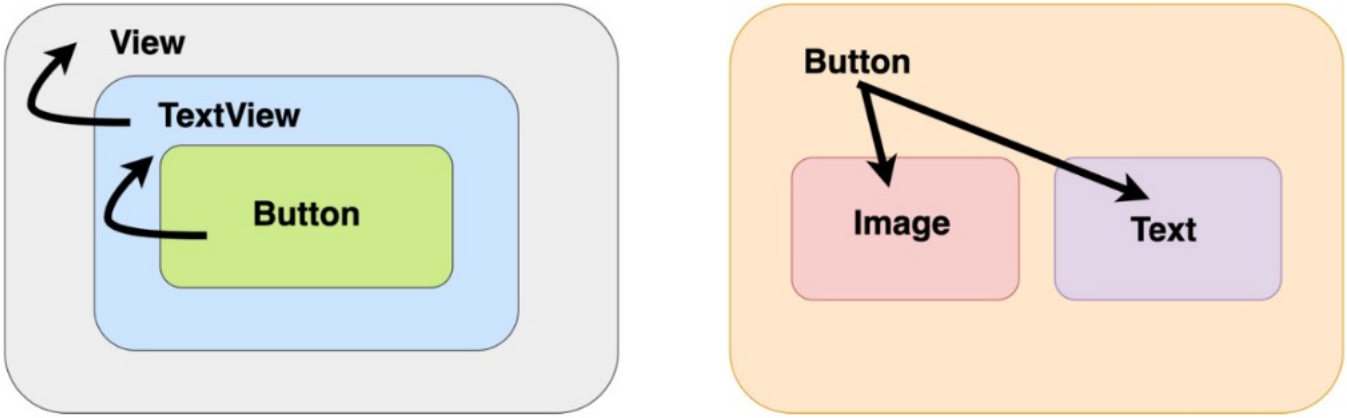


Figure 1.6 – Inheritance versus composition

- *Favor composition over Inheritance*: classes should achieve polymorphic behavior and code reuse by their composition (contain instances of other classes that contain the desired functionality) rather than inherit from a base class

Declarative Programming

- Focuses on describing **what** a program should accomplish
  - Not how it should be done (as with Imperative Programing)
- Functional Programming is a type of Declarative Programming

Imperative vs. Declarative

- *Imperative thinking*: display a list, then collapse it
  - Function that changes how it is displayed

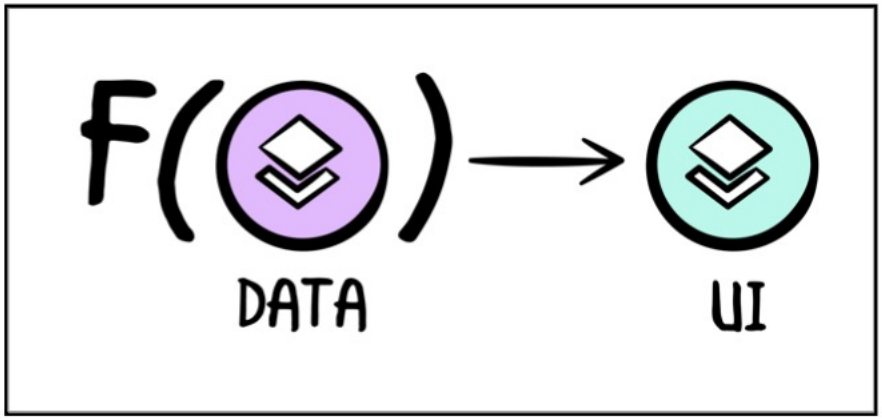
```
// Handle guest list visibility
if (event.guests.size > 0 && !hasGuestList()) {
  addGuestList()
} else if (event.guests.size == 0 && hasGuestList()) {
  removeGuestList()
}

// Handle case with more than 5 guests
if (event.guests.size > 5 && !isGuestListCollapsed()) {
  collapseGuestList()
} else if (event.guests.size > 0 && isGuestListCollapsed()) {
  expandGuestList()
}

// Handle guest count badge
if (event.guests.size <= 50) {
  setGuestCountText("$count")
} else {
  setGuestCountText("50+")
}
```

- *Declarative thinking*: display a collapsed list
  - Function that declares what to display

```
if (event.guests.size > 0) {
  Guests(collapsed = event.guests.size > 5) {
    if (event.guests.size > 50) {
      Badge(text="50+")
    } else {
      Badge(text="$count")
    }
  }
}
```

Composables are Declarative

- Functions that take data (state) as parameters and emit UI
  - Composable is immutable
  - Function is idempotent without side effects (no global variables)



Kotlin Lives in Both Worlds

- Kotlin is
  - an Object-Oriented Programming Language (Imperative) with Function Programming constructs (Declarative)
  - a Functional Programming Language (Declarative) with Object-Oriented constructs (Imperative)

NEW Android Framework

- All built with Kotlin
  - Model stores state
    - State: Object-Oriented encapsulation (what state is)
  - ViewModel updates Model via events
    - Events: Imperative design (how state changes)
  - View represents state as a set of composables
    - Composables: Declarative design (what state looks like)
    - Input is State & Event Callback

On Tap For Today

- Final Project
- Modern Android
  - Java → Kotlin
  - MVC & View → MVVM & Compose

- Practice
  - Lab00A

For Wednesday 1/18

- Complete Lab00A

- Watch first set of Kotlin videos
  - In class quiz on Kotlin syntax
  - It is open notes, so take notes while watching the video

On Tap For Today

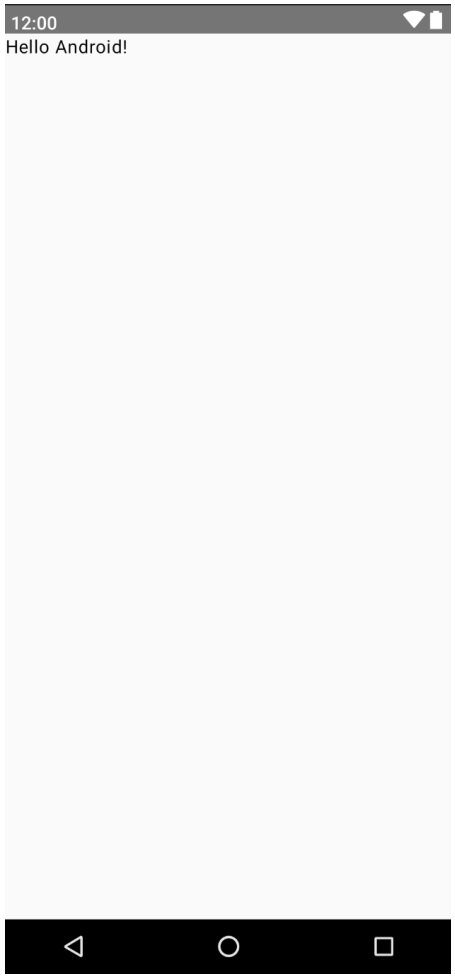
- Final Project
- Modern Android
  - Java → Kotlin
  - MVC → MVVM
  - View Framework → Compose UI
- Practice
  - Lab00A

Lab00A: Hello World!

- Write up online. Three “simple” steps
  - Start Android Studio
  - Create a new project
  - Create an AVD
- There may be special cases we’ll need to work through. Ed Discussion is your friend

Lab00A: Hello Android!

- Android Studio gives you for free!



- New Project
  - Provide Name & Domain
  - Include Kotlin support
  - Minimum SDK: 10.0 (Q) / API 29
  - Empty Compose Activity (Material3)