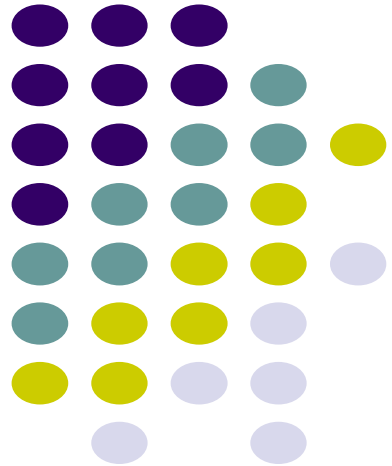
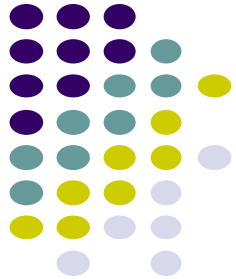
CS 528 Mobile and Ubiquitous Computing

Lecture 3a: View Bindings, Android Components, Saving State & Rotation

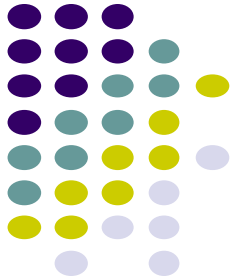
Emmanuel Agu





Kotlin Introduction

Kotlin Introduction



- Since you can program….
- Easiest way to learn kotlin is to go through
  - Kotlin Bootcamp for programmers
  - https://developer.android.com/courses/kotlin-bootcamp/overview
  - Free!!
  - Should take about 1 hour to go through
- Can type code in bootcamp in Android studio or try web interpreter at
  - https://try.kotlinlang.org/

Kotlin Introduction

- Kotlin program to print “Hello World”

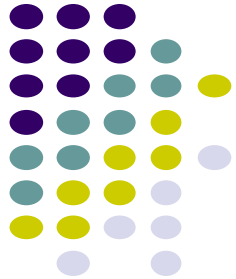
```
fun printHello() {
    println ("Hello World")
}

printHello()
```

- Two types of kotlin variables:
  - **var:** can be changed
  - **val:** cannot be changed

```
var fish = 1
fish = 2
val aquarium = 1
aquarium = 2
```

```
⇒ error: val cannot be reassigned
```

Kotlin Strings



- Use “ at beginning and end of string
- + to concatenate strings

```
val numberOfFish = 5
val numberOfPlants = 12
"I have $numberOfFish fish" + " and $numberOfPlants plants"
```

```
⇒ res20: kotlin.String = I have 5 fish and 12 plants
```

- Can use expression in a string template

```
"I have ${numberOfFish + numberOfPlants} fish and plants"
```

```
⇒ res21: kotlin.String = I have 17 fish and plants
```

Compare Conditions and Booleans

- Check if a value is within a range

```
val fish = 50
if (fish in 1..100) {
    println(fish)
}
```

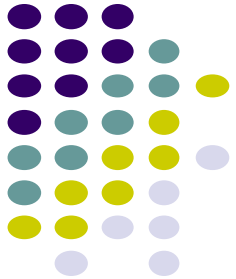
- Another example:

```
val numberOfFish = 50
```

```
if (numberOfFish == 0) {
    println("Empty tank")
} else if (numberOfFish < 40) {
    println("Got fish!")
} else {
    println("That's a lot of fish!")
}
```

```
⇒ That's a lot of fish!
```

Nullability



- Can declare variables as nullable (cannot be set to null), to reduce errors
- By default variables cannot be null

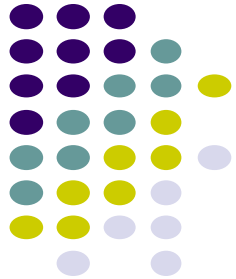
```
var rocks: Int = null
```

```
⇒ error: null can not be a value of a non-null type Int
```

- Use ? to make a variable null

```
var marbles: Int? = null
```

Lists and Arrays



- Use **listOf** to declare a list

```
val school = listOf("mackerel", "trout", "halibut")
println(school)
```

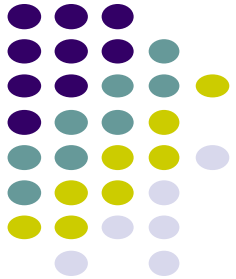
```
⇒ [mackerel, trout, halibut]
```

- Use **mutableListOf** to declare list that can be changed (items removed)

```
val myList = mutableListOf("tuna", "salmon", "shark")
myList.remove("shark")
```

```
⇒ res36: kotlin.Boolean = true
```

Nullability



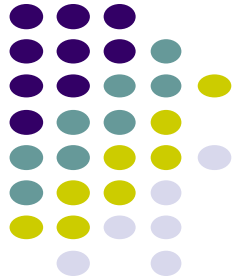
- if **fishFoodTreats** is not null, then decrement it

```
var fishFoodTreats = 6
if (fishFoodTreats != null) {
    fishFoodTreats = fishFoodTreats.dec()
}
```

- Can chain null tests using ?:
- E.g. if **fishFoodTreats** is not null, decrement and use it. Otherwise use value after ?:

```
fishFoodTreats = fishFoodTreats?.dec() ?: 0
```

- **Important:** Go through bootcamp! Best 1 hour you will invest
  - https://developer.android.com/courses/kotlin-bootcamp/overview

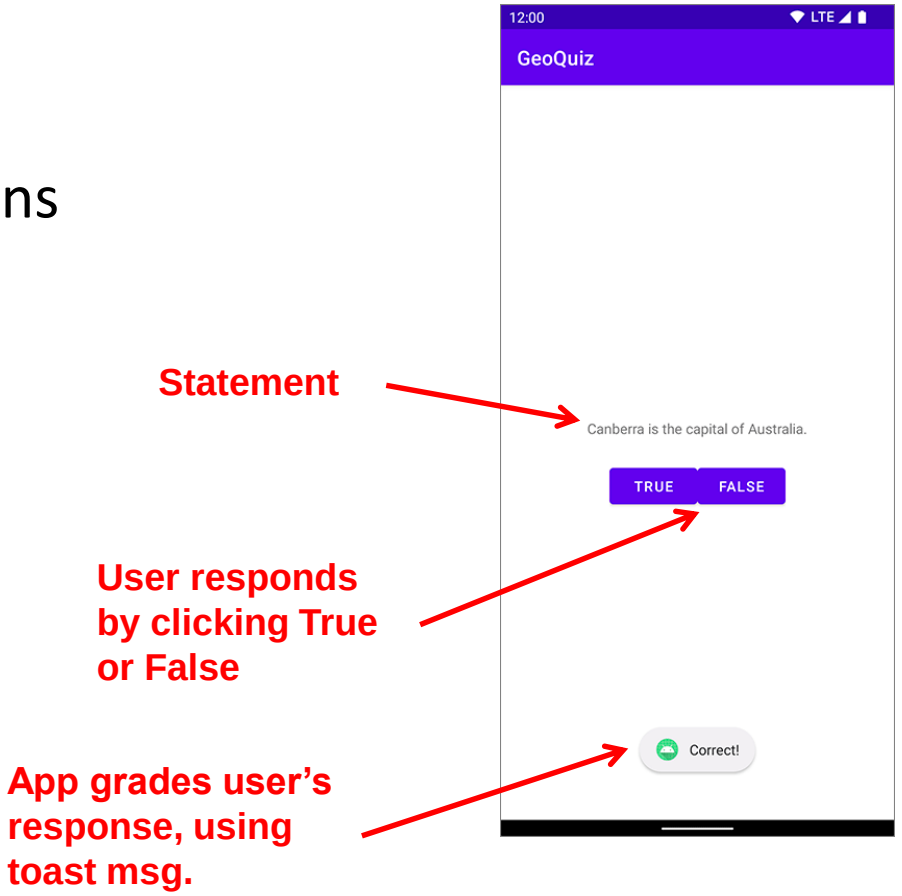


Add more questions to GeoQuiz

Recall: GeoQuiz App

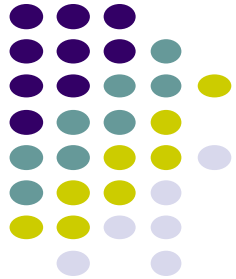
Ref: Android Nerd Ranch (5th edition), Ch. 1, pgs. 1-32

- App makes statements about geography, with goal to test user's knowledge of geography
- User answers by pressing **True** or **False** buttons
- Hard-coded 1 question



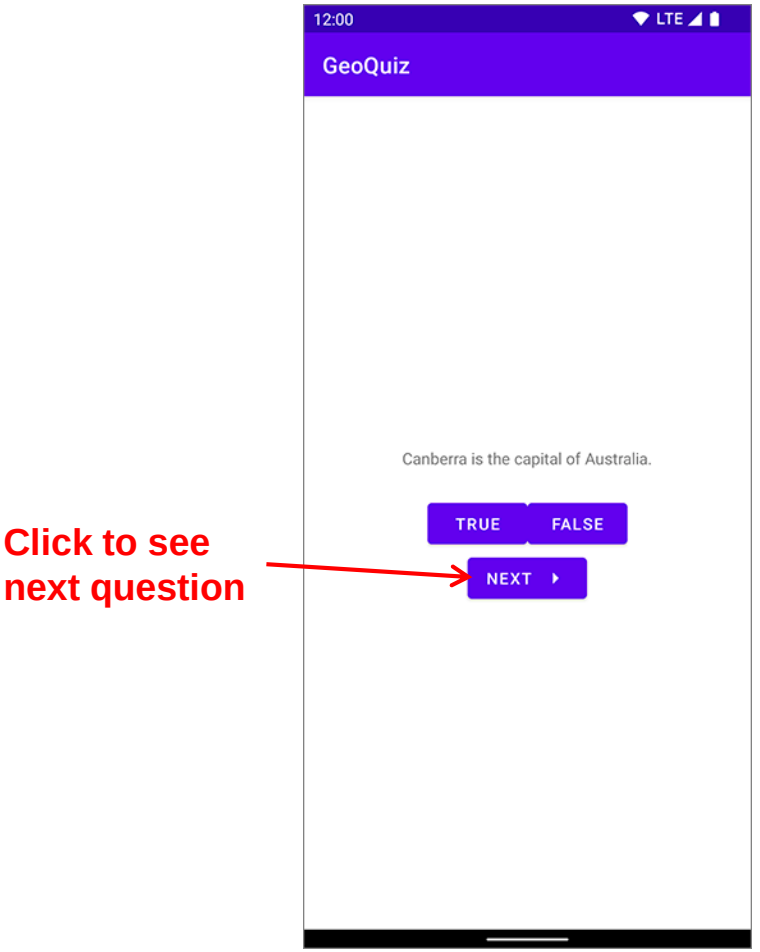
GeoQuiz: Creating Interactive Question Interface

ANR (5th edition) Chapter 2



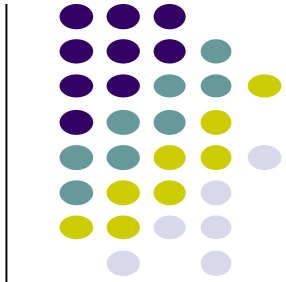
- **Goal:** Create list of questions (not just one)

- List of questions/statements:
  - Canberra is the capital of Australia
  - The Pacific Ocean is larger than the Atlantic Ocean
  - The Suez Canal Connects the Red Sea and the Indian Ocean
  - The source of the Nile River is in Egypt
  - The Amazon is the longest river in the Americas
  - Lake Baikal is the world's oldest and deepest freshwater lake

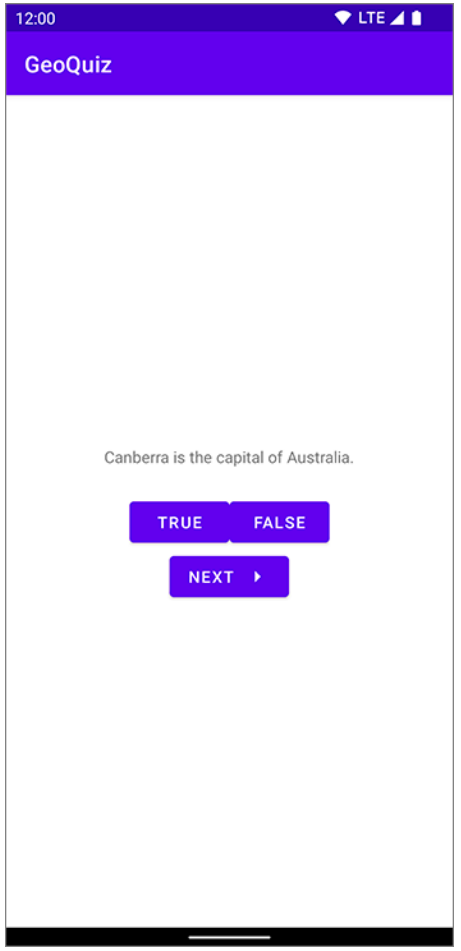


GeoQuiz: Creating Interactive Question Interface

ANR (5th edition) Chapter 2



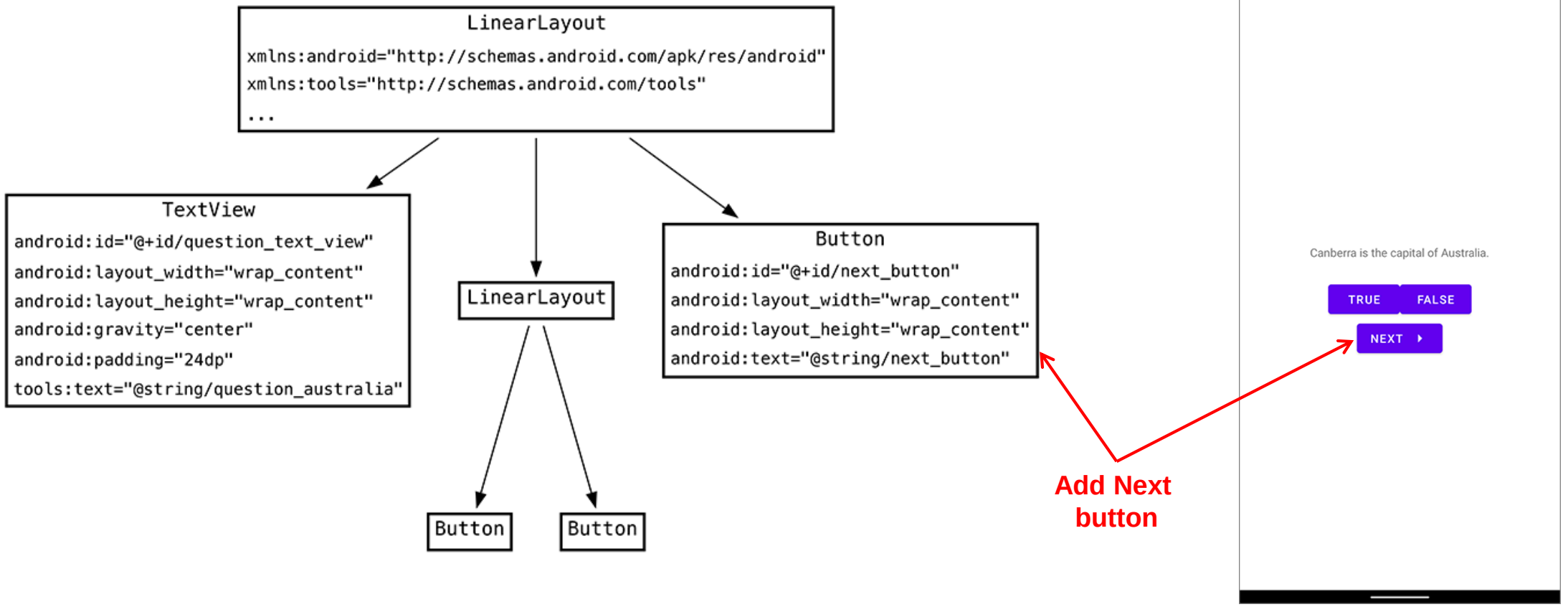
- Create **Question** class to GeoQuiz in new **Question.kt** file
- Instance of **Question** encapsulates single true/false question
- MainActivity will manage a collection of **Question** objects



GeoQuiz: Updating the Layout

ANR (5th edition) Chapter 2

- Add Next button to layout file

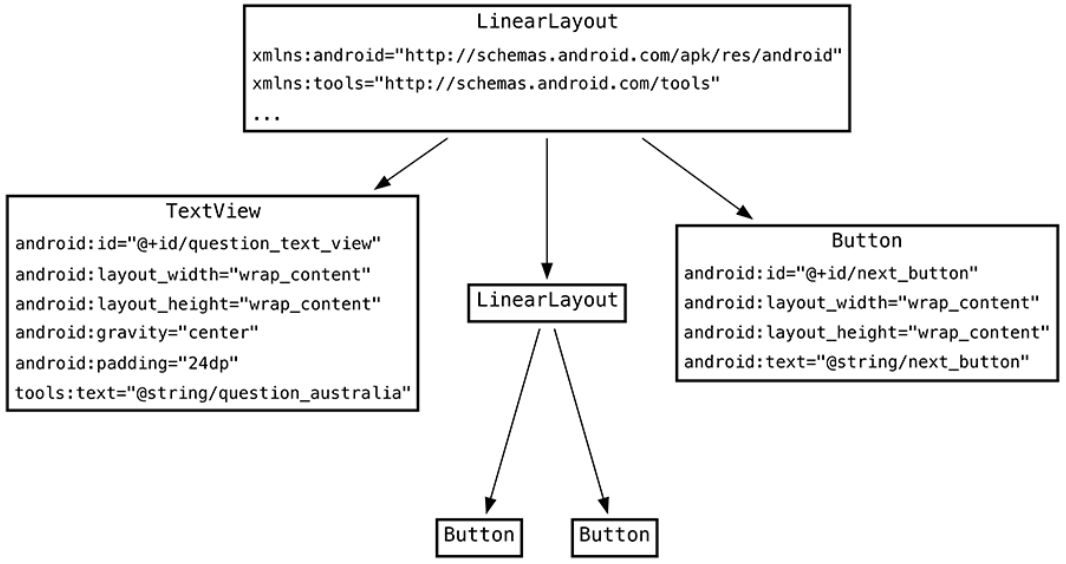


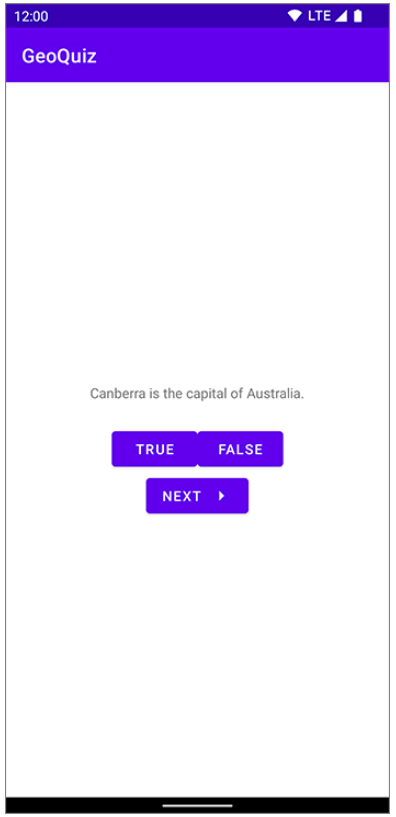
GeoQuiz: Updating the Layout

ANR (5th edition) Chapter 2

- Add Next button to layout file

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        ... >

    <TextView
        android:id="@+id/question_text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:padding="24dp"
        android:text="@string/question_text"
        tools:text="@string/question_australia" />

    <LinearLayout ... >
        ...
    </LinearLayout>

    <Button
        android:id="@+id/next_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/next_button" />

</LinearLayout>
```





GeoQuiz: Updating Strings.xml

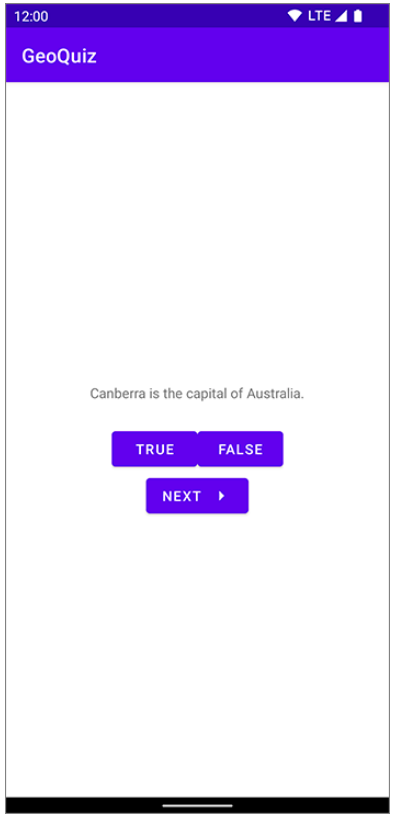
ANR (5th edition) Chapter 2

- Rename **question_text** as **question_Australia** and add string for next button

```
<string name="app_name">GeoQuiz</string>
<string name="question_text">Canberra is the capital of Australia.</string>
<string name="question_australia">Canberra is the capital of Australia.</string>
<string name="true_button">True</string>
<string name="false_button">False</string>
<string name="next_button">Next</string>
```

- Add strings for more questions

```
<string name="question_australia">Canberra is the capital of Australia.</string>
<string name="question_oceans">The Pacific Ocean is larger than
  the Atlantic Ocean.</string>
<string name="question_mideast">The Suez Canal connects the Red Sea
  and the Indian Ocean.</string>
<string name="question_africa">The source of the Nile River is in Egypt.</string>
<string name="question_americas">The Amazon River is the longest river
  in the Americas.</string>
<string name="question_asia">Lake Baikal is the world\'s oldest and deepest
  freshwater lake.</string>
```



GeoQuiz: Write Code in MainActivity.kt

ANR (5th edition) Chapter 2

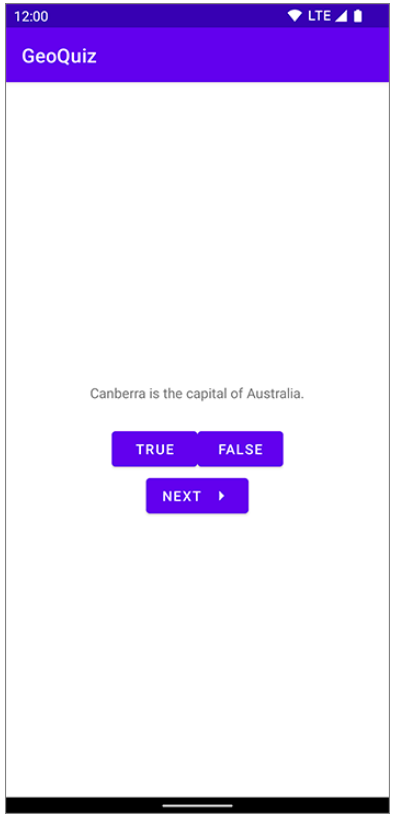
- Create list of **Question** objects, and an index for the list

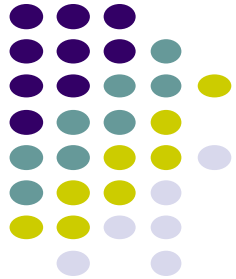
```
class MainActivity : AppCompatActivity() {

    private lateinit var trueButton: Button
    private lateinit var falseButton: Button

    private val questionBank = listOf(
            Question(R.string.question_australia, true),
            Question(R.string.question_oceans, true),
            Question(R.string.question_mideast, false),
            Question(R.string.question_africa, false),
            Question(R.string.question_americas, true),
            Question(R.string.question_asia, true))

    private var currentIndex = 0
    ...
}
```

Index of question

question

Correct answer (True or false)



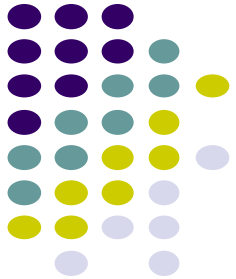


View Bindings Overview

HFAD (3rd edition)

View Bindings

Ref: HFAD (3rd edition), pgs 403-416



- Previously used findViewById( ) to get reference to view (buttons, etc) declared in XML file. E.g.

```
val startButton = findViewById<Button>(R.id.start_button)
startButton.setOnClickListener {
    //Code that does something
}
```

This is the approach you've been using in the book so far.

- **Problem:** findViewById( ) is inefficient (Android searches entire view hierarchy for matching ID)
- **New way:** View binding (part of Android Jetpack libraries, makes coding easier)
- Interact with button using binding object's startButton property
- **Note:** properties are variables defined inside a class but outside a method

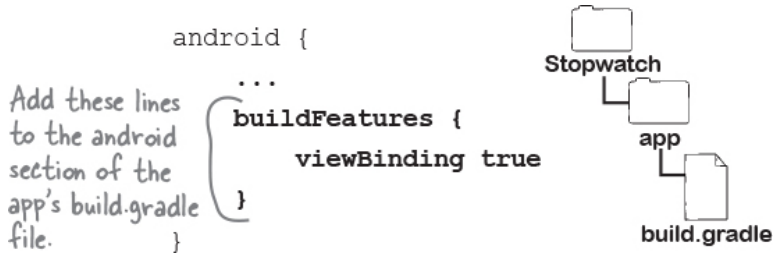
```
binding.startButton.setOnClickListener {
    //Code that does something
}
```

With view binding, you can interact with the button using the binding object's startButton property. This property holds a reference to the view with an ID of start_button.

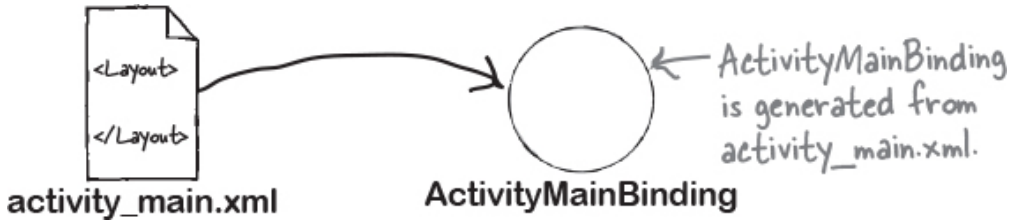
View Bindings

Ref: HFAD (3rd edition), pgs 403-416

- First enable view binding in build.gradle file



- Enabling view binding automatically generates a binding class for each of the layout files in app.
- E.g. for XML file **activity_main.xml**, a binding class named **ActivityMainBinding** is auto-generated

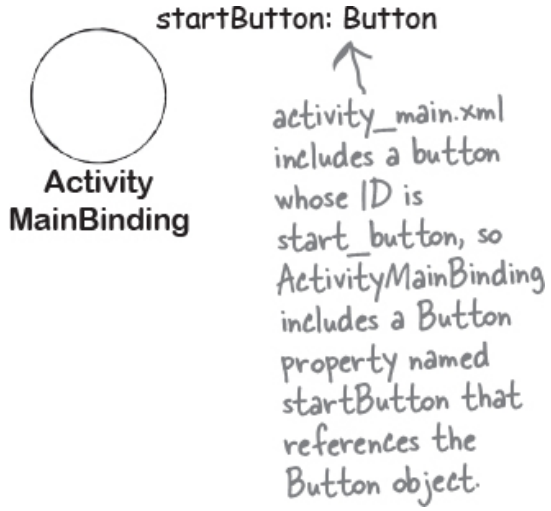


- Binding class (e.g. ActivityMainBinding) includes a property (variable) for each view or widget (e.g. buttons, Textview, etc) that has an ID

View Bindings

Ref: HFAD (3rd edition), pgs 403-416

- E.g. **activity_main.xml** contains a button, **ActivityMainBinding** contains property **StartButton**
- Instead of calling findViewByID( ) to reference button, just interact with button property in binding class



View Bindings

Ref: HFAD (3rd edition), pgs 403-416

- Code snippet showing main steps for using view binding

```
package com.hfad.stopwatch

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import com.hfad.stopwatch.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        binding = ActivityMainBinding.inflate(layoutInflater)
        val view = binding.root
        setContentView(view)
    }
}
```

Import the binding class, in this example ActivityMainBinding.

Add this binding property.

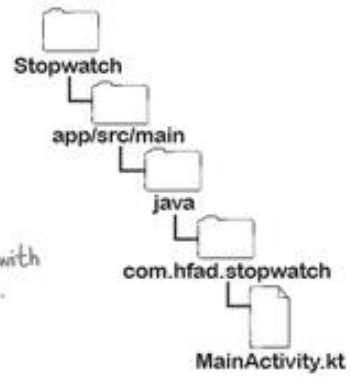
Replace this line with the code beneath.

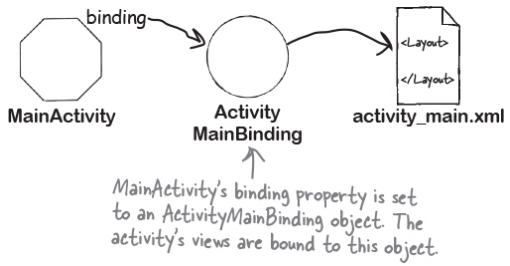
Set the property.

This creates an ActivityMainBinding object that's linked to the layout

Set view to the root view.

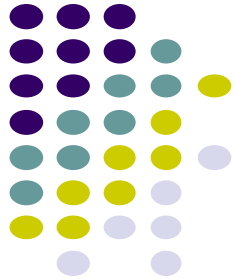
Pass the root view to setContentView().





private: only visible inside this class

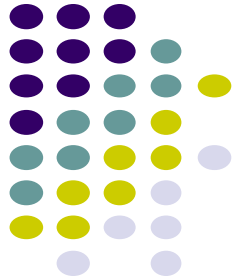
lateinit: Variable is not initialized when object is created. Initialized in future



Switching GeoQuiz to use View Bindings

GeoQuiz: Switching to View Bindings

ANR (5th edition) Chapter 2

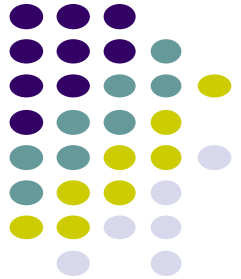


- Initialize **ActivityMainBinding** in **MainActivity.kt**

```
import com.bignerdranch.android.geoquiz.databinding.ActivityMainBinding

class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    private lateinit var trueButton: Button
    private lateinit var falseButton: Button
    ...
    override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      binding = ActivityMainBinding.inflate(layoutInflater)
      setContentView(binding.root)
      ...
    }
    ...
}
```

GeoQuiz: Switching to View Bindings

ANR (5th edition) Chapter 2



- Can then access TRUE and FALSE buttons as properties (binding class variables)

```
binding.trueButton.setOnClickListener { view: View ->
    ...
}
```

```
binding.falseButton.setOnClickListener { view: View ->
    ...
}
```

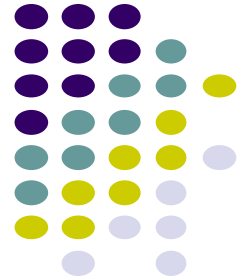
- Also set the text of the TextView that displays the question as a property

```
val questionTextResId = questionBank[currentIndex].textResId
binding.questionTextView.setText(questionTextResId)
```

← Get the question corresponding to currIndex

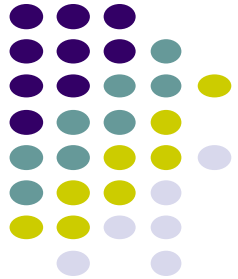
← Set the text of the TextView to question

See ANR Chapter 2 for rest of the code



Android App Components

Android App Components



- Typical Java program starts from main( )

```java
class SillyApp {
    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

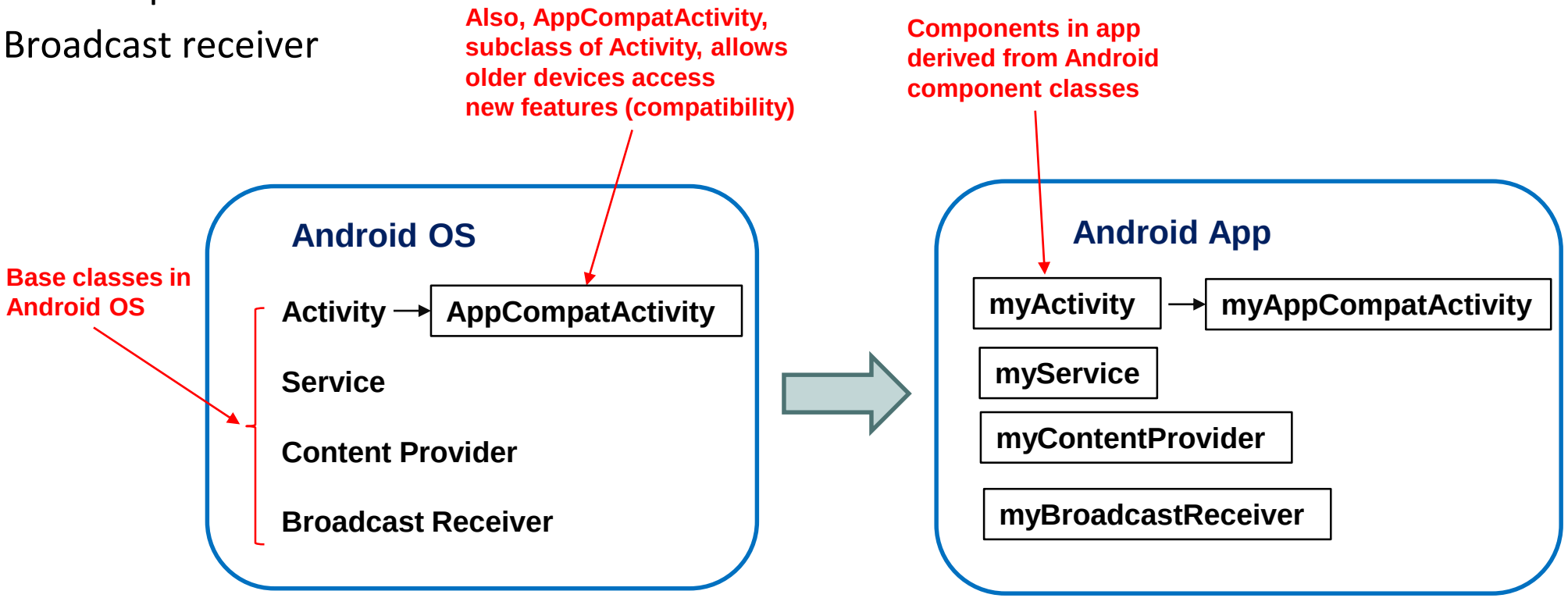
- Similarly, in Kotlin

```
fun main(args: Array<String>) {
        println("Hello, World!")
                                    }
```

- Android app: No need to write a main
- Just define app components derived from base classes already defined in Android

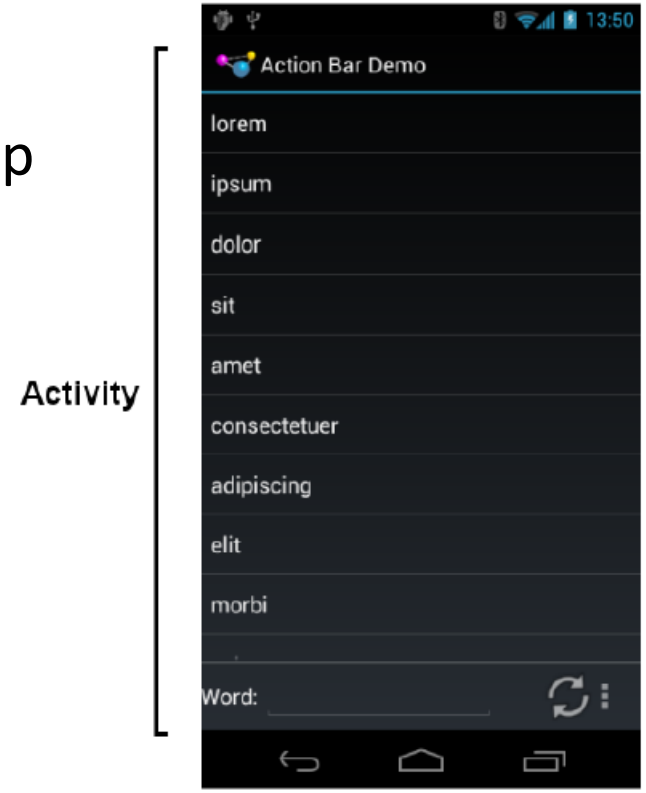
Android App Components

- App components derived from base classes already defined in Android
- 4 main types of Android app components:
  - Activity (already seen this), or AppCompatActivity
  - Service
  - Content provider
  - Broadcast receiver



Recall: Android Activity

- Activity: main building block of Android UI
- Analogous to a window or dialog box in a desktop application
- Apps
  - have at least 1 activity that deals with UI
  - Entry point of app similar to **main( )** in C
  - typically have multiple activities
- Example: A camera app
  - **Activity 1:** to focus, take photo, start activity 2
  - **Activity 2:** to present photo for viewing, save it
- Example: Deriving an App's Activity



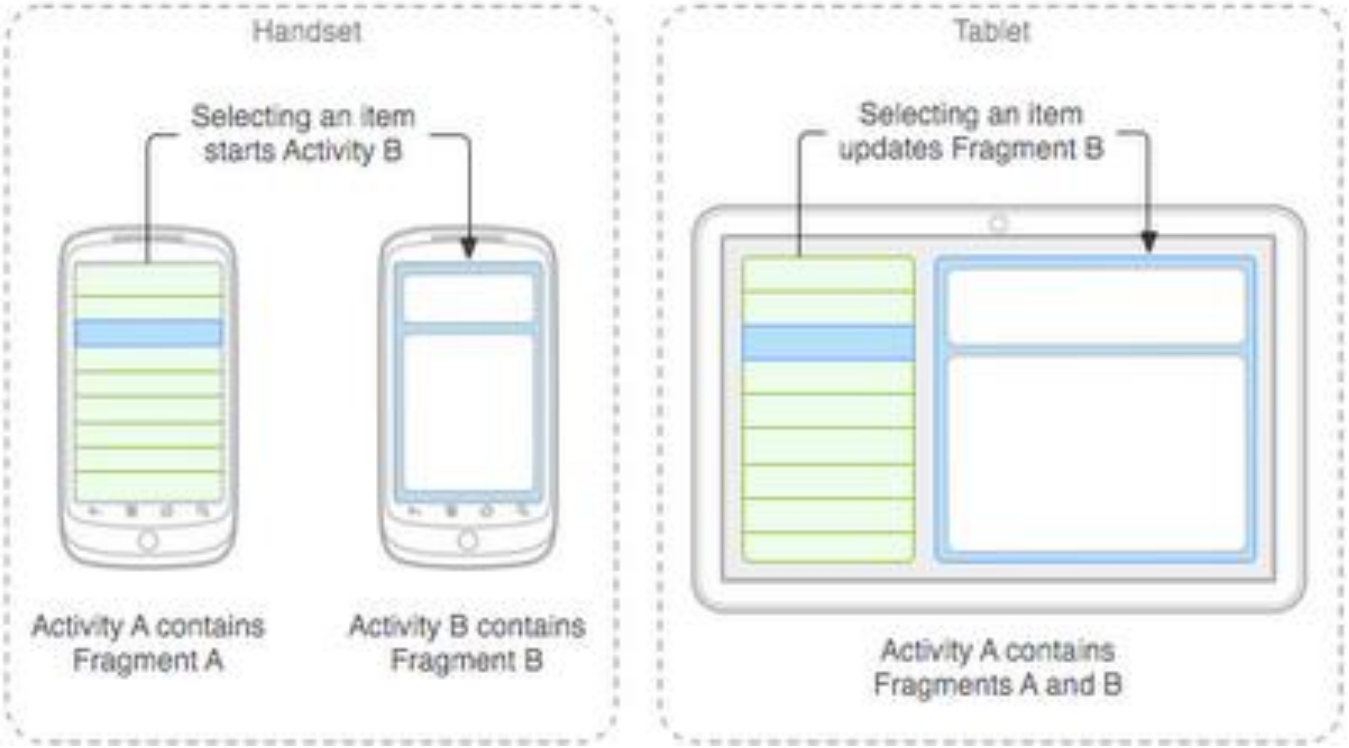

```
class MainActivity : AppCompatActivity() {
```

App's Activity

Derived from Android's AppCompatActivity class

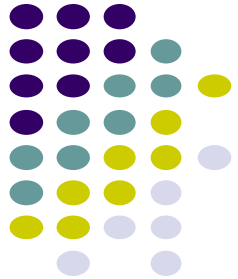
Fragments

- Fragments
  - UI building blocks (pieces), can be re-arranged in Activity in different ways.
  - Enables app to look different on different devices (e.g. phone vs tablet)
- An activity can contain multiple fragments
  - Organize fragments differently on different devices (e.g. for phone vs tablet)
- Parent activity:
  - Hosts fragment
  - Defines rectangle for fragment on screen
  - Swaps fragments in/out dynamically
  - More later



Services

https://developer.android.com/guide/components/services

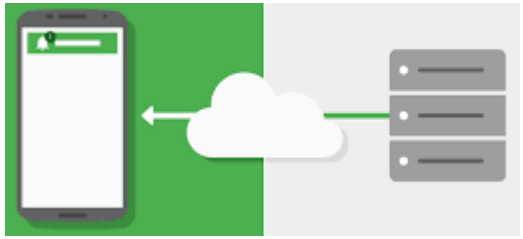


- Activities are short-lived, can be shut down anytime. E.g. when user presses back button
- Services keep running, performs functions, typically in background with no UI
- Similar to Linux/Unix CRON job
- Example uses of services:
  - Periodically check/update device's GPS location
  - Check for updates to RSS feed
- Independent of any activity, minimal interaction
- Typically an activity controls a service -- start it, pause it, get data from it
- Services in an App (e.g. myService) are sub-class of Android's **Service** class

Android Platform Services

- Android Services can either be on:
  - On smartphone or Android device (local)
  - Remote, on Google server/cloud

Android services on smartphone



Android services In Google cloud

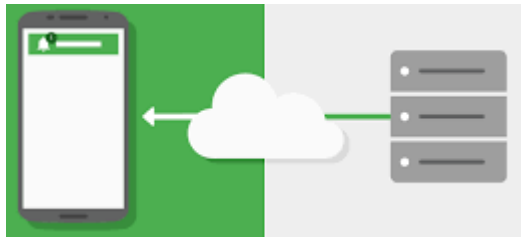
- Android platform local services examples (on smartphone):
  - **LocationManager:** location-based services.
  - **ClipboardManager:** access to device's clipboard, cut-and-paste content
  - **DownloadManager:** manages long-running HTTP downloads in background
  - **FragmentManager:** manages the fragments of an activity.
  - **AudioManager:** provides access to audio and ringer controls.

Google Services (In Google Cloud)

- Maps
- Location-based services
- Game Services
- Authorization APIs
- Google Plus
- Play Services
- In-app Billing
- Google Cloud Messaging
- Google Analytics
- Google AdMob ads

Typically need
Internet connection

Android services
on smartphone

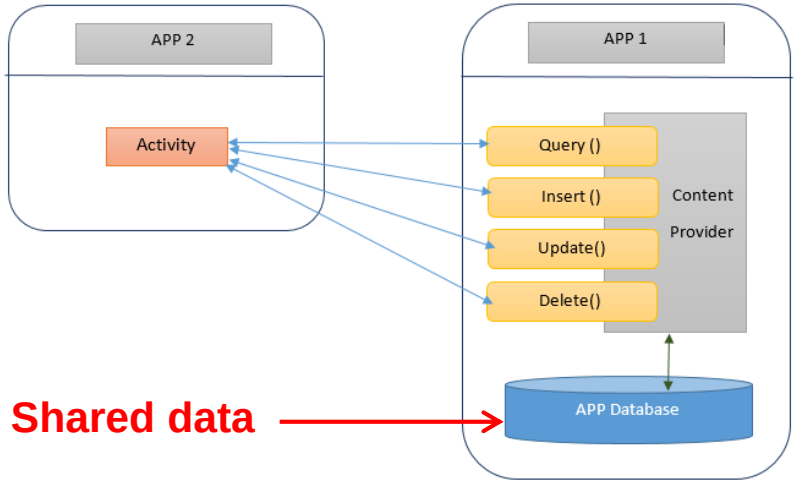


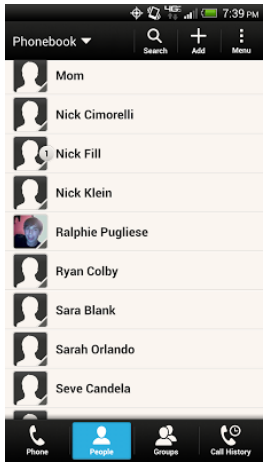
Android services
In Google cloud

Content Providers

https://developer.android.com/guide/topics/providers/content-provider-basics

- Android apps can share data (e.g. User's contacts) as content provider
- Content Provider:
  - Abstracts shareable data, makes it accessible through methods
  - Apps can access shared data by calling methods for the relevant **content provider**
  - E.g. Can query, insert, update, delete shared data (see below)

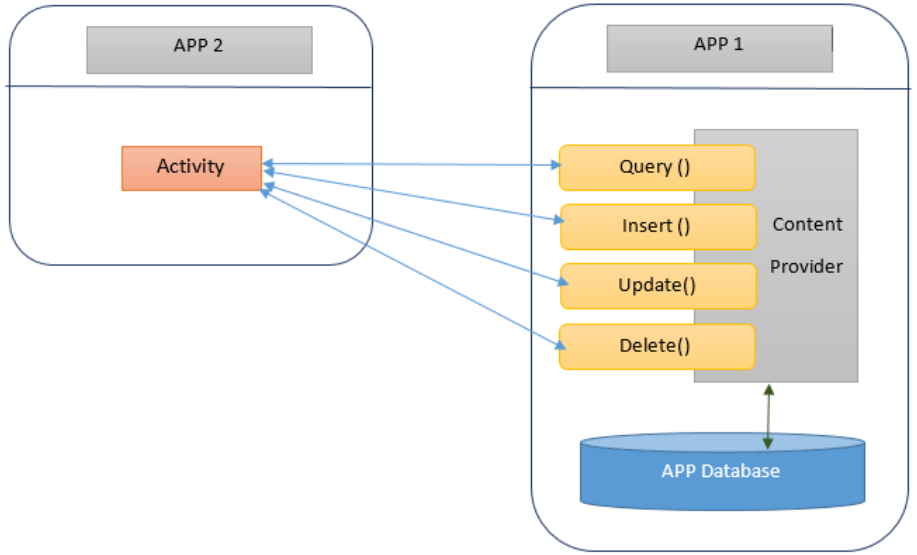


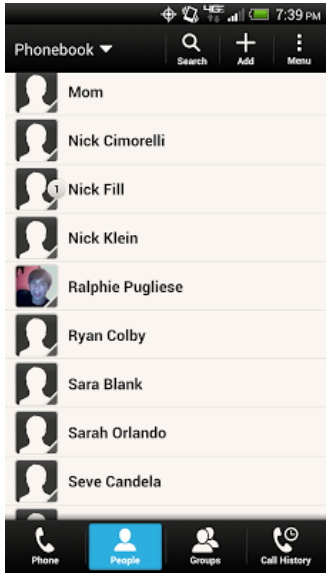


Content Providers

https://developer.android.com/guide/topics/providers/content-provider-basics

- **E.g.** Data stored in Android Contacts app can be accessed by other apps
- **Example:** We can write an app that:
  - Retrieve's contacts list from contact app's content provider
  - Adds contacts to social networking (e.g. Facebook)
- Apps can also **ADD** to data through content provider. E.g. Add contact
- E.g. Our app can also share its data, presented as table(s) similar to relational database
- Content provider in an App are sub-class of Android's **ContentProvider** class

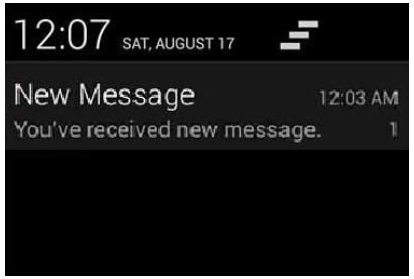




Broadcast Receivers

https://developer.android.com/guide/components/broadcasts

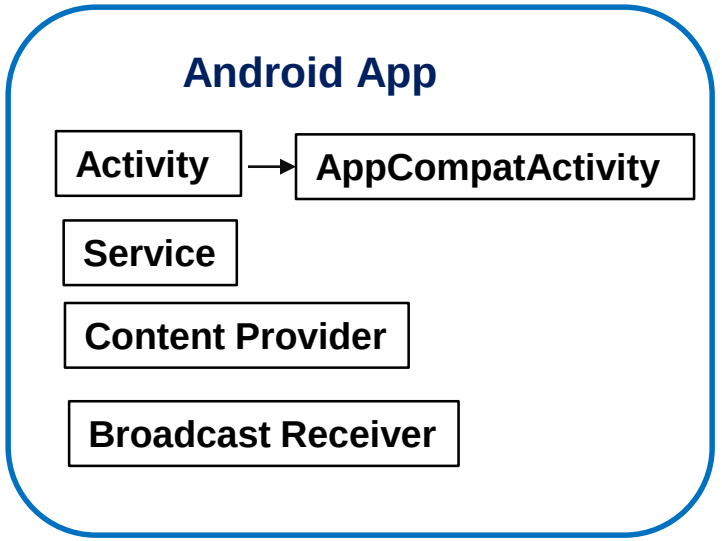
- Android OS (system), or applications, periodically ***broadcasts*** events
- Example broadcasts:
  - Battery getting low
  - Download completed
  - New email arrived
- Any app can create broadcast receiver to listen for broadcasts, respond
- Our app can also initiate broadcasts
- Broadcast receivers typically
  - Doesn't interact with the UI
  - Creates a status bar notification to alert the user when broadcast event occurs
- Broadcast Receiver in an App are sub-class of Android's **BroadcastReceiver** class

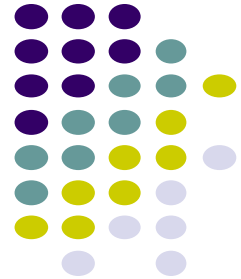


Quiz

- Pedometer App has the following Android components:
  - **Component A:** continuously counts user's steps continuously even when user closes app, does other things on phone (e.g. YouTube, calls)
  - **Component B:** Displays user's step count
  - **Component C:** Can be contacted to retrieve list of user's friends (from contacts list) to text them every day with step totals

- What should component A be declared as?
  - Activity, service, content provider, broadcast receiver?
- What of component B?
- Component C?

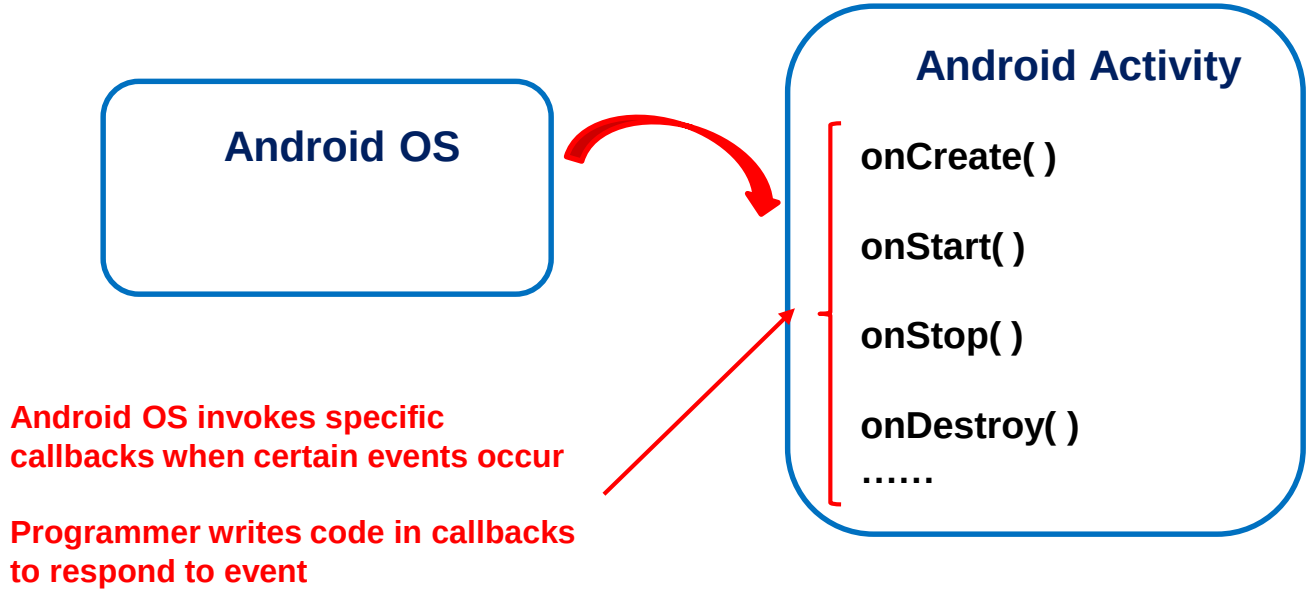




Android Activity LifeCycle

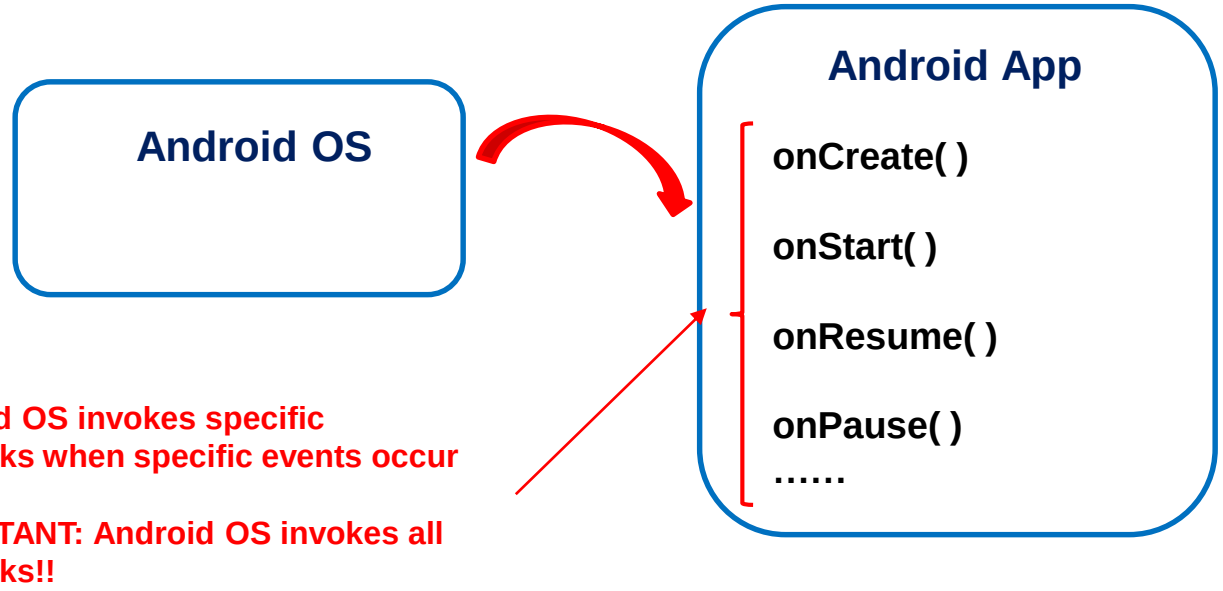
Starting Activities

- Android Activity callbacks invoked corresponding to app state.
- Examples:
  - When activity is created, its **onCreate( )** method invoked (like constructor)
  - When activity is stopped, its **onStop( )** method invoked



Activity Callbacks

- onCreate() ← Already saw this (initially called)
- onStart()
- onResume()
- onPause()
- onStop()
- onDestroy()



OnCreate( )

- Used to initialize activity once created
- **Important:** Android OS calls my apps' OnCreate( ) method
- Operations typically performed in onCreate() method:
  - Inflate (create) widgets and place them on screen (e.g. using layout files specified by setContentView( ))
  - Getting references to inflated widgets (using findViewbyId( ))
  - Setting widget listeners (e.g. onClickListener) to handle user interaction
- E.g.

```
class MainActivity : AppCompatActivity() {

    private lateinit var trueButton: Button
    private lateinit var falseButton: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        trueButton = findViewById(R.id.true_button)
        falseButton = findViewById(R.id.false_button)
    }
}
```

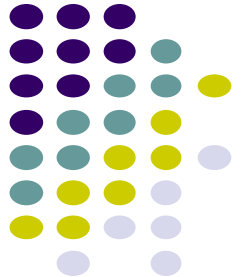
Call parent class onCreate( )

Format this activity (screen) using activity_main.xml

Get references to inflated widgets (using findViewById)

Understanding Android Lifecycle

https://developer.android.com/guide/components/activities/activity-lifecycle.html

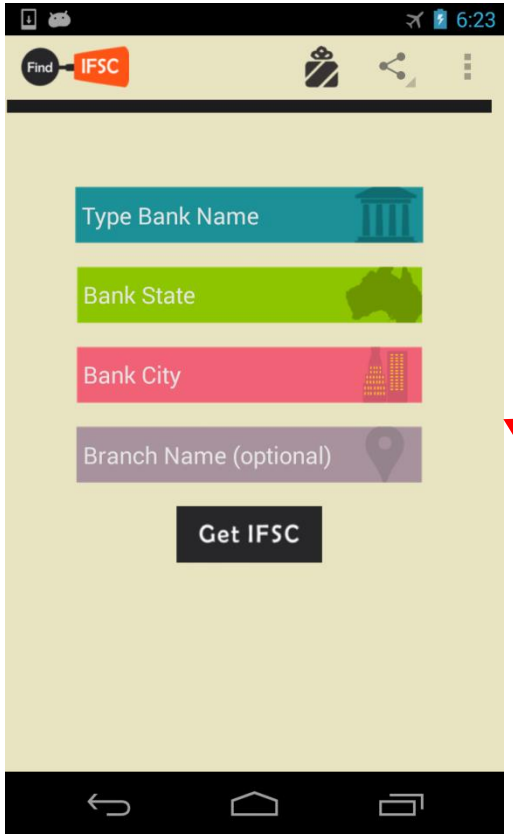


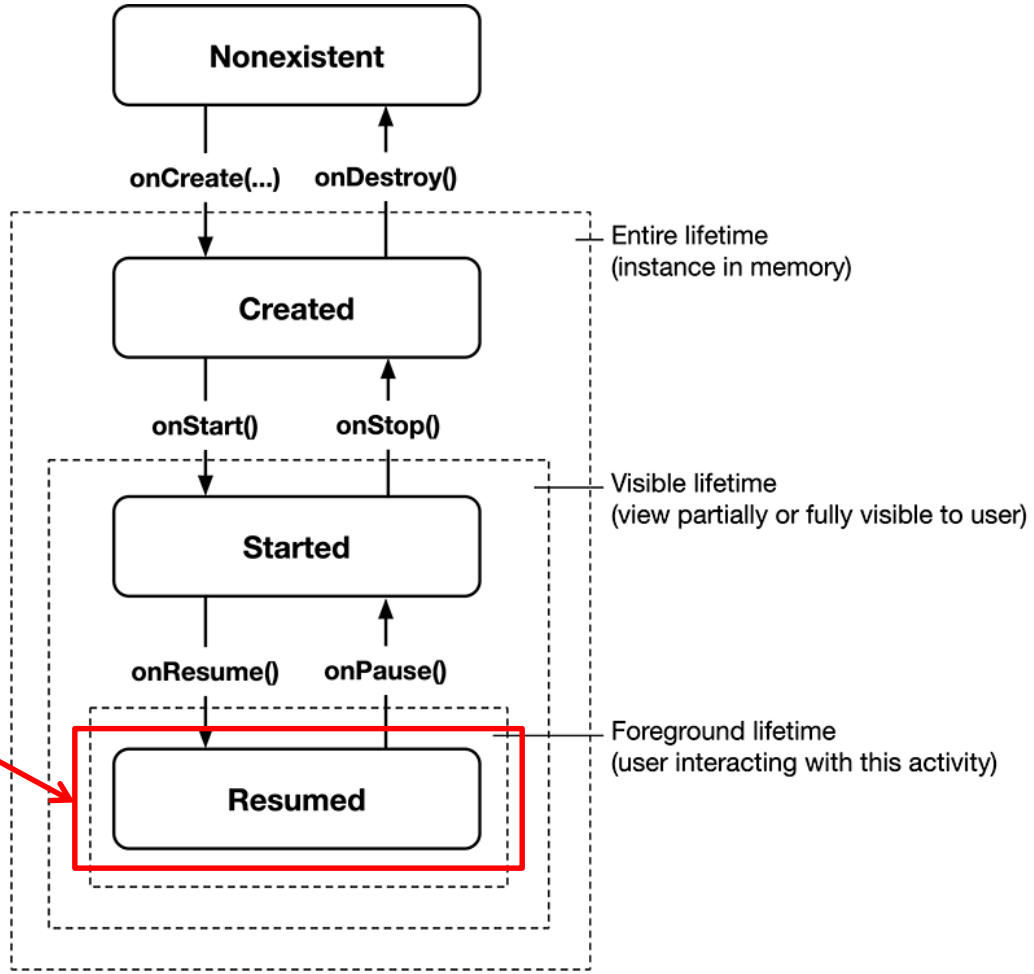
- Many **disruptive** things could happen while app is running. E.g.
  - Incoming call or text message,
  - User closes our app, user switches to another app, etc

- A well-designed app should NOT:
  - Crash if interrupted, or user switches to other app
  - Lose the user's state/progress (e.g state, positions of chess game app) if they leave your app and return later
  - Crash or lose the user's progress when screen rotates between landscape and portrait orientation.
    - E.g. Youtube video should continue at correct point (e.g., 4.56) after rotation
- To handle these situations, appropriate callback methods must be invoked appropriately to “tidy up” before app gets bumped

Resumed (Running) App

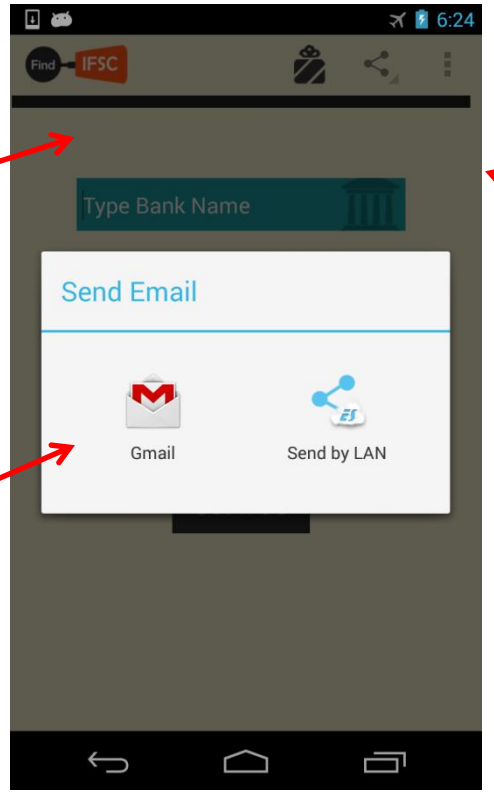
- A resumed app is one that user is currently using or interacting with
  - Visible, in foreground





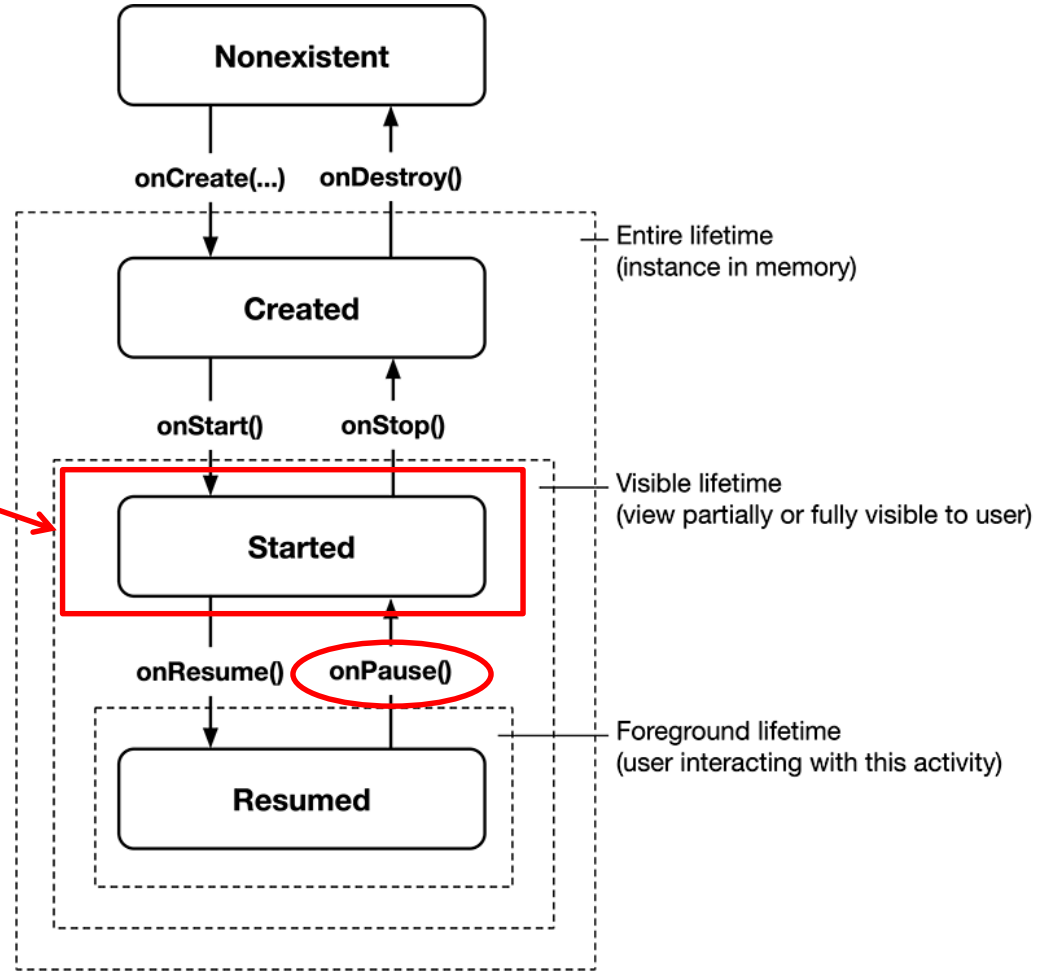
Started (Paused) App

- An app is **started (paused)** if it is **visible but no longer in foreground**
- E.g., blocked by a pop-up dialog box
- App's **onPause( )** method is called during transition from resumed to started (paused) state



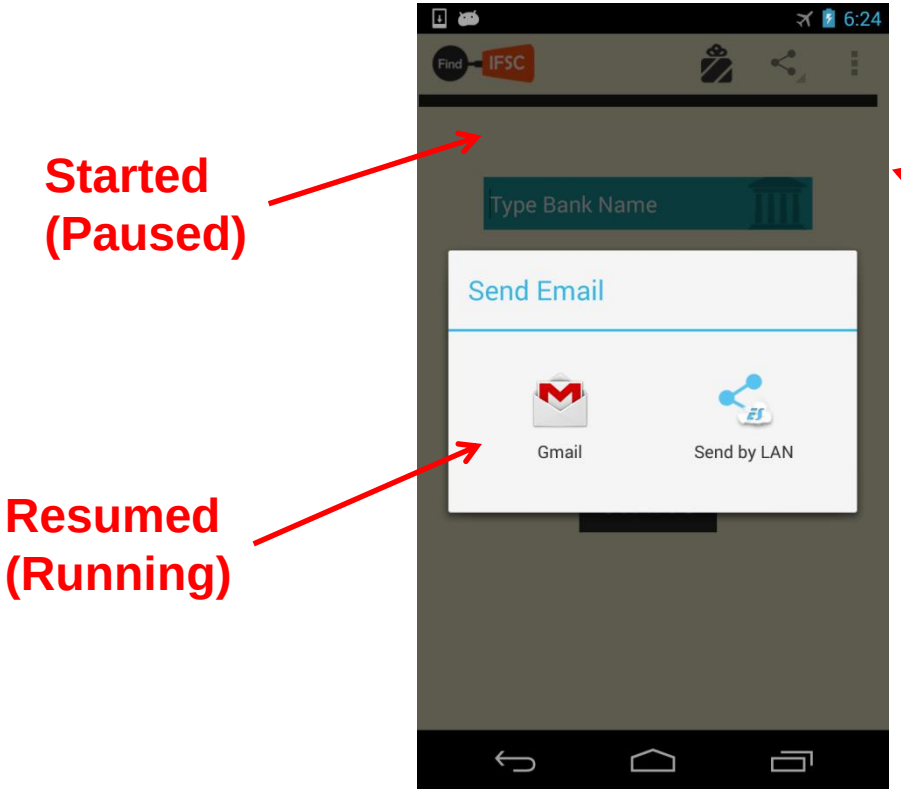
Started (Paused)

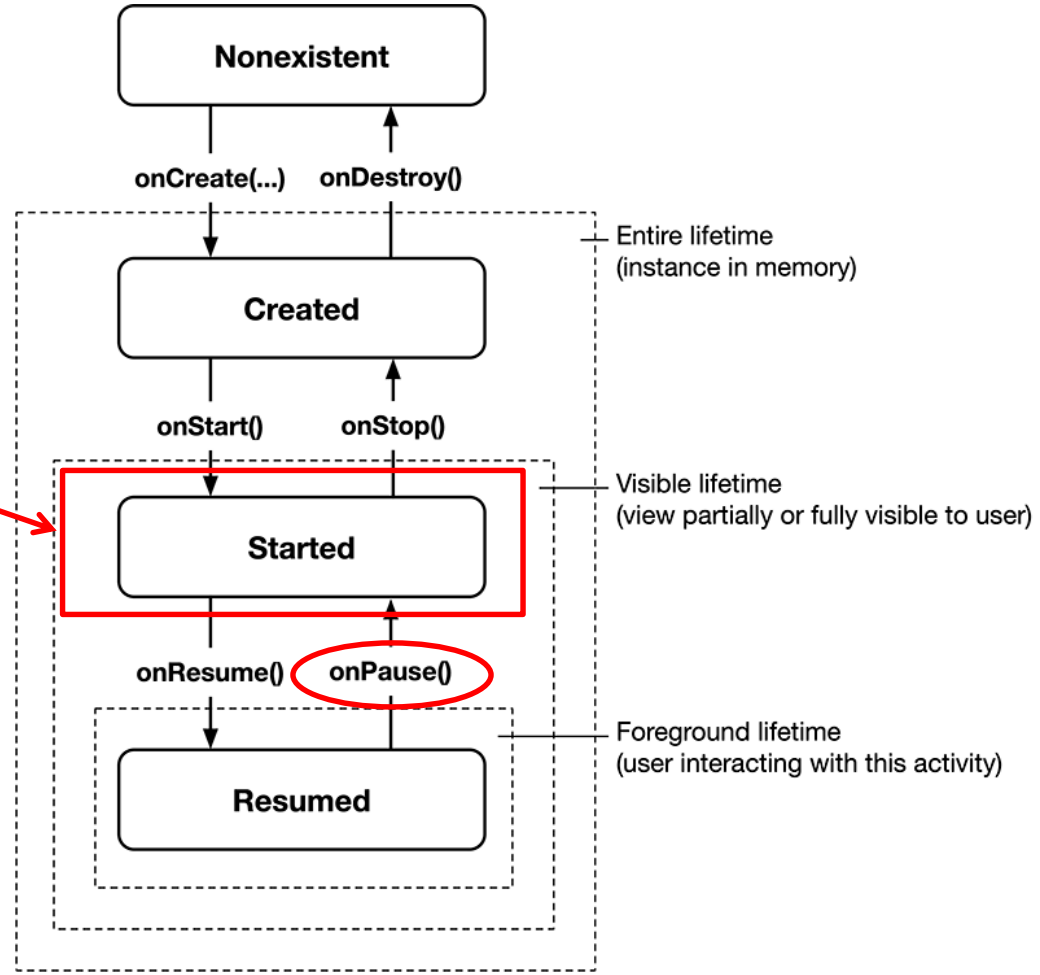
Resumed (Running)



onPause( ) Method

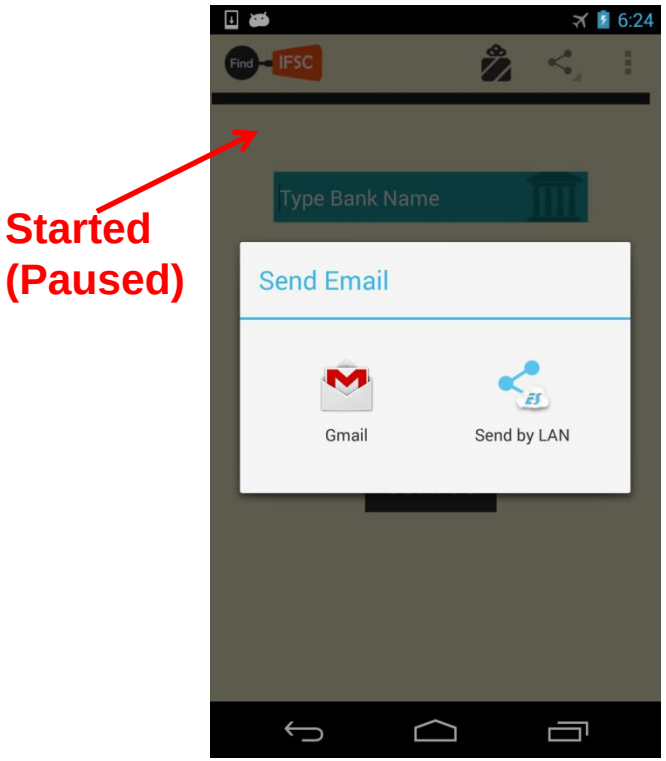
- Idea: release resources, hardware if app going from foreground to partially visible
- Typical actions taken in onPause( ) method
  - Stop listening for GPS, release handles to sensors (e.g GPS, camera)
  - Stop audio and video
  - Stop CPU intensive tasks

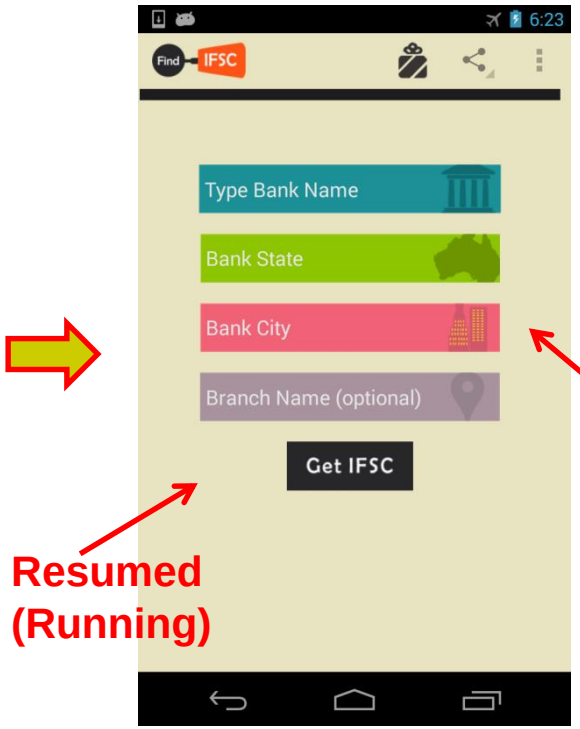


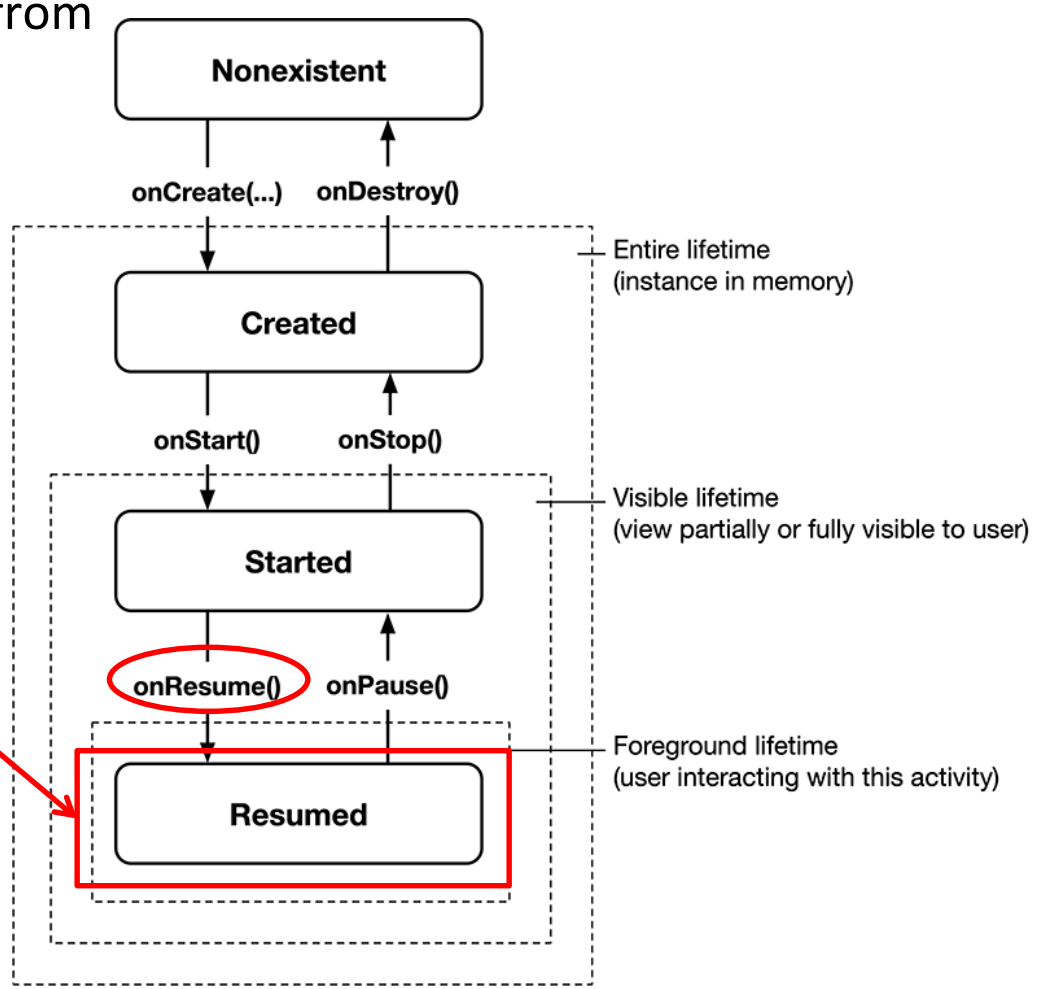


onResume( ): Resuming Paused App

- A **paused** app goes back to **resumed** if it becomes fully visible and in foreground
  - E.g. pop-up dialog box blocking it goes away
- App's **onResume( )** method is called during transition from **started (paused)** to **resumed (running)** state
  - Restart videos, GPS checking, etc

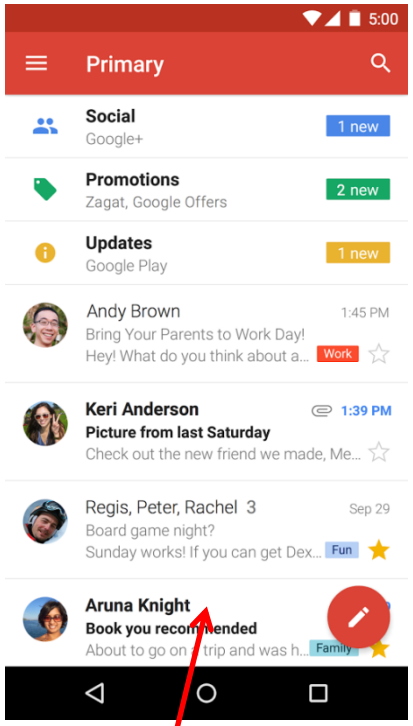




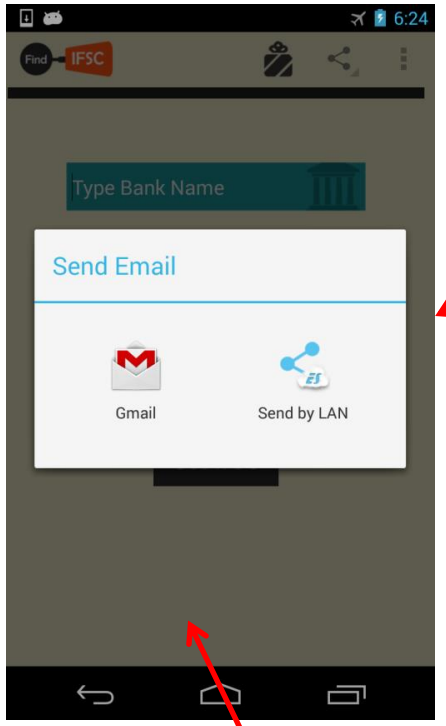


Created (Stopped) App

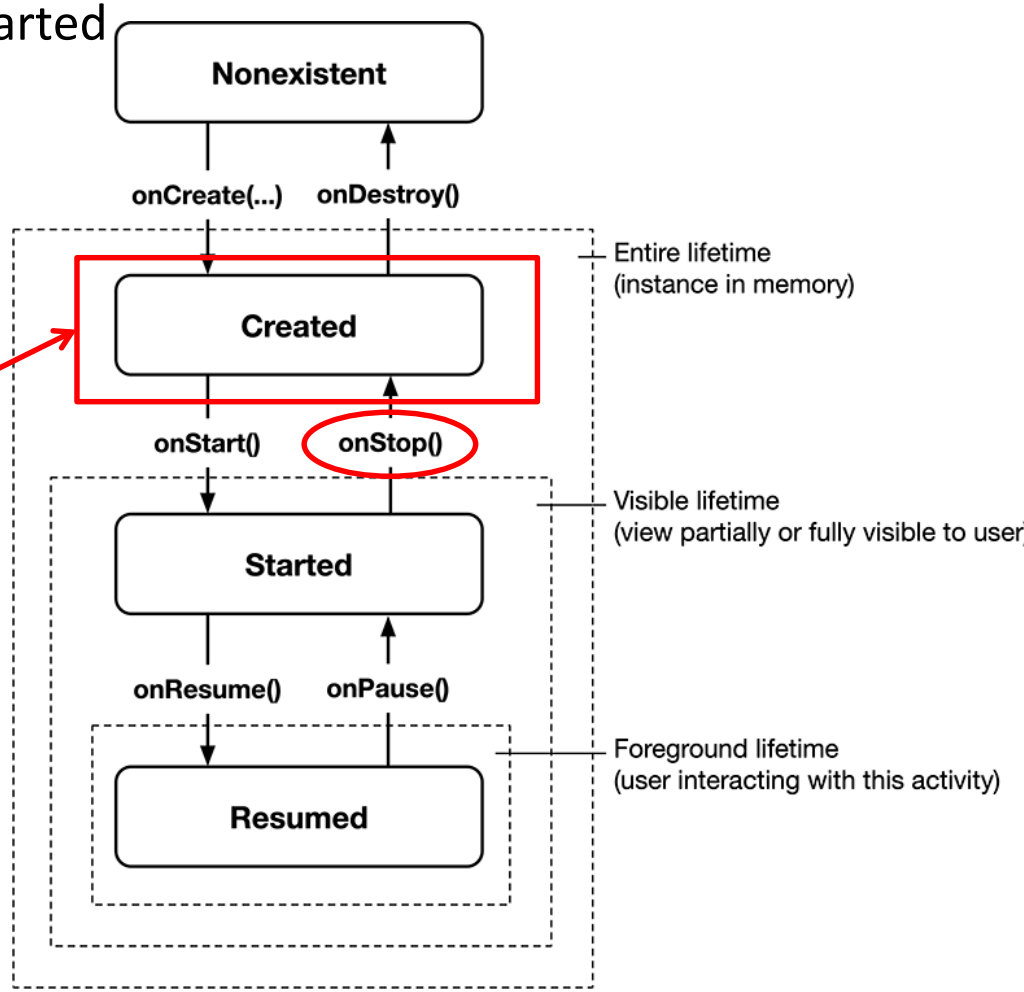
- An app is **stopped** if it's **no longer visible + no longer in foreground.** E.g.
- E.g., user starts using another app
- App's **onStop( )** method is called during transition from started (paused) to created (stopped) state



Now Running



Was Paused -> Stopped

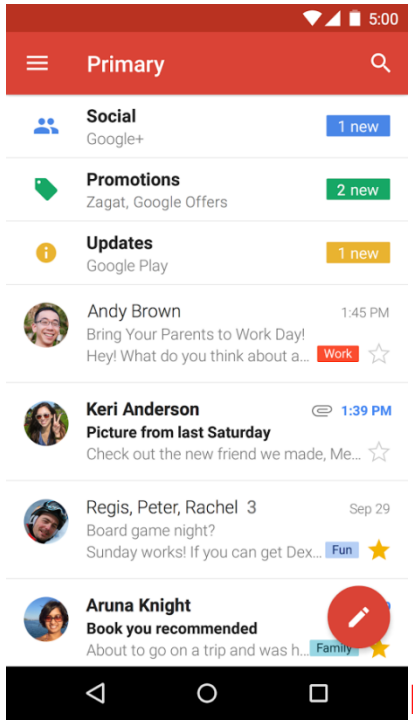


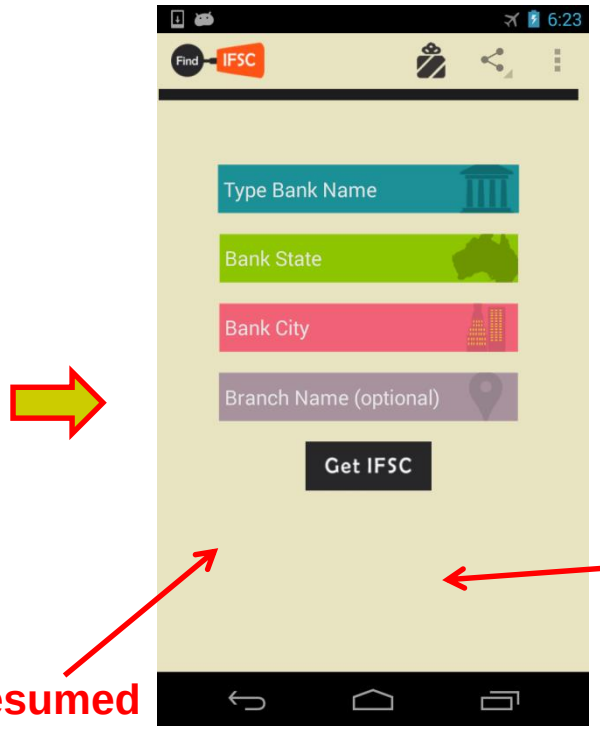
onStop() Method

- An activity is stopped when:
  - User receives phone call
  - User starts another app
- Activity instance and variables of stopped app are retained but no code is being executed by the activity
- If activity is stopped, in onStop( ) method, well behaved apps should
  - save progress to enable seamless restart later
  - Release all resources, save info (persistence)

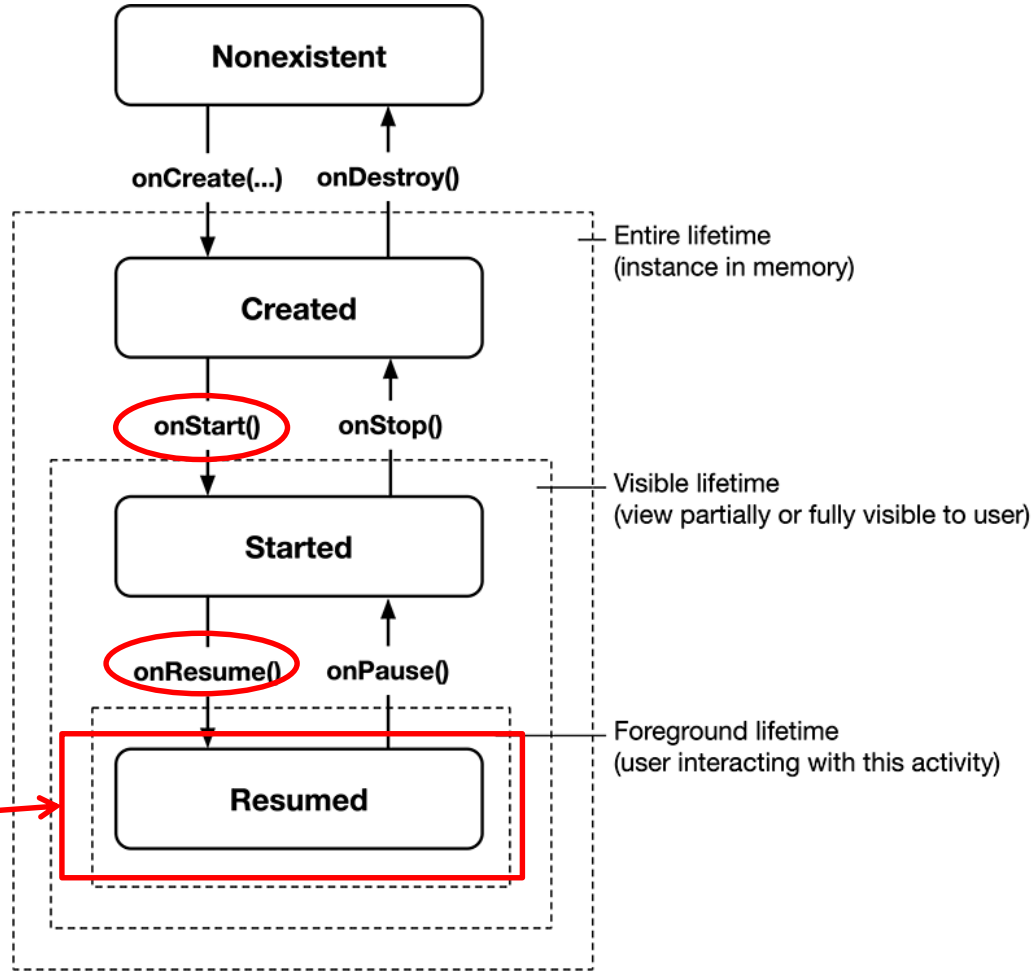
Resuming Created (Stopped) App

- A **created (stopped)** app can go back into **resumed (running)** state if it becomes visible and in foreground (user starts using it again)
- App's **onStart( )** and **onResume( )** methods called to transition from **created (stopped)** to **resumed (running)** state



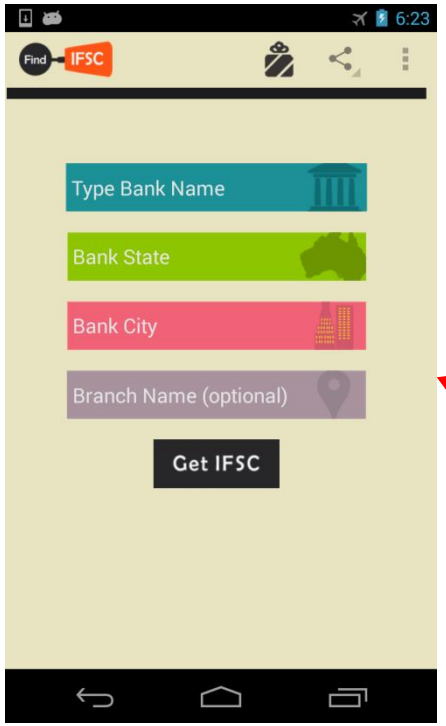


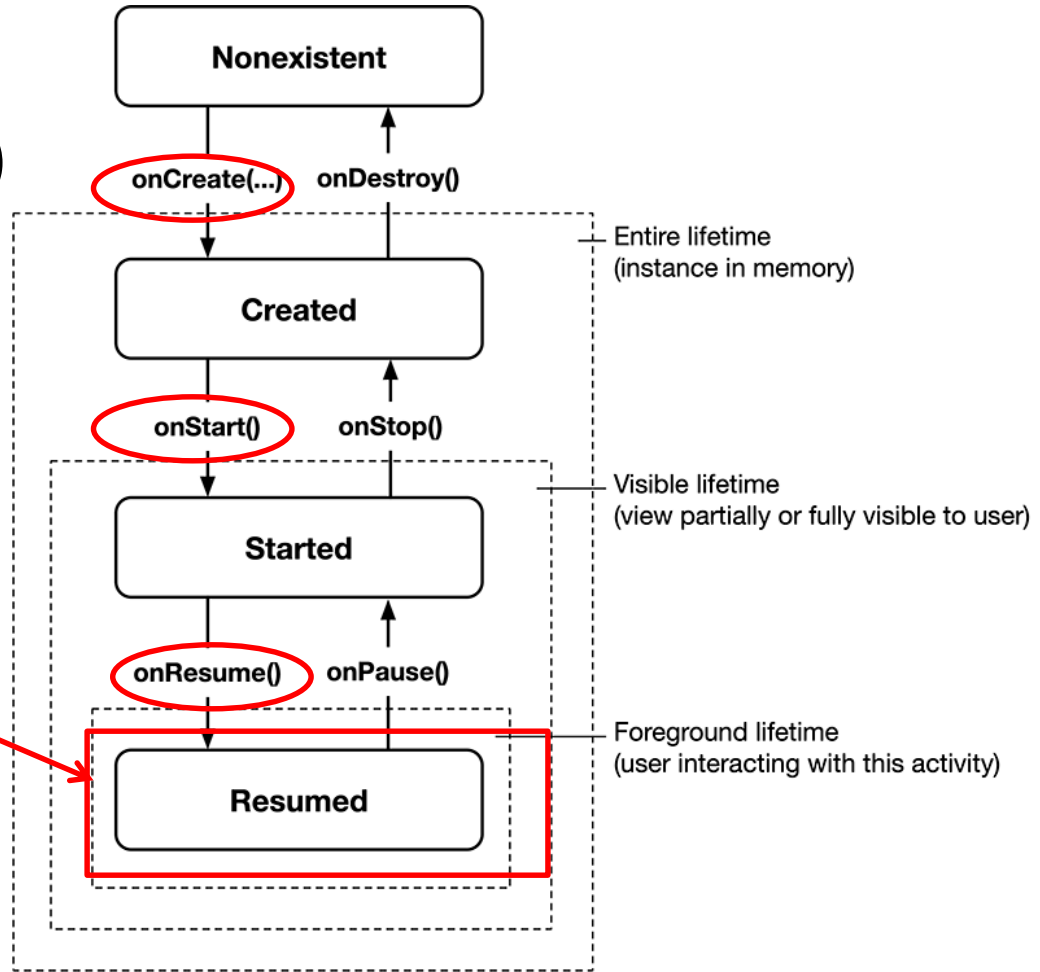
Resumed (running)

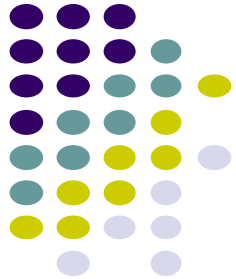


Starting New App

- To launch new (non-existent) app, and get it to resumed (running)
- App's **onCreate( )**, **onStart( )** and **onResume( )** methods are called
- Thereafter, new app is now **resumed (running)**





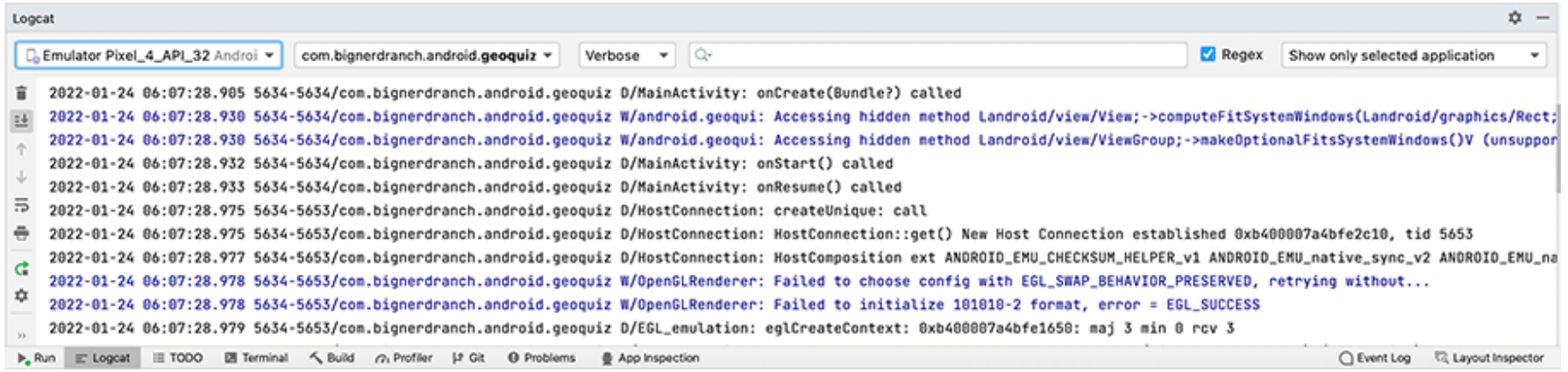


Logging Errors in Android

Logging Errors in Android

Android Nerd Ranch (5th Edition), pages 57-63

- Android Studio can log and display various types of errors/warnings in LogCat Window



- Error logging is in **Log** class of **android.util** package, so need to

  import android.util.Log;

- Turn on logging of different message types by calling appropriate method

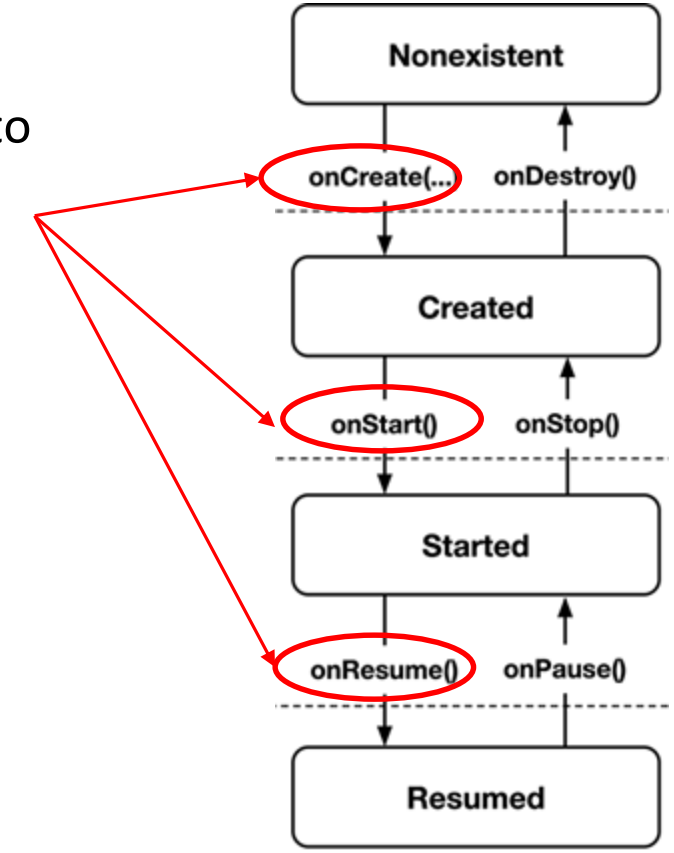
Method	Purpose
`Log.e()`	Log errors
`Log.w()`	Log warnings
`Log.i()`	Log informational messages
`Log.d()`	Log debug messages
`Log.v()`	Log verbose messages

Ref: Introduction to Android Programming, Annuzzi, Darcey & Conder

GeoQuiz MainActivity.kt

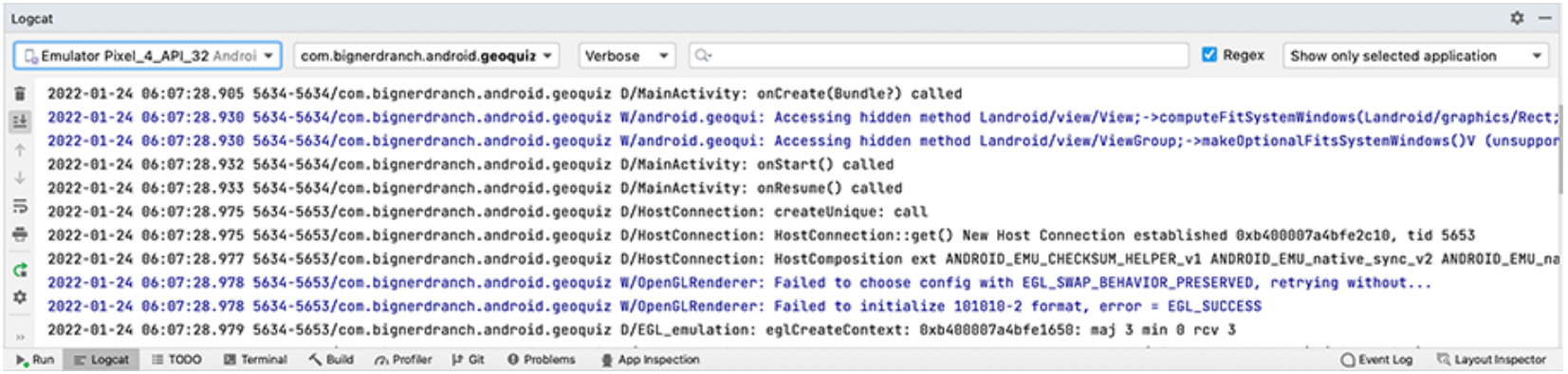
Android Nerd Ranch (5th Edition), pages 57-63

- A good way to understand Android lifecycle methods is to print debug messages in callbacks, displayed when called




```
onCreate(  ){
   … print message “..OnCreate called..”…

}

onStart(  ){
   … print message “..OnStart called..”…

}
```

… etc

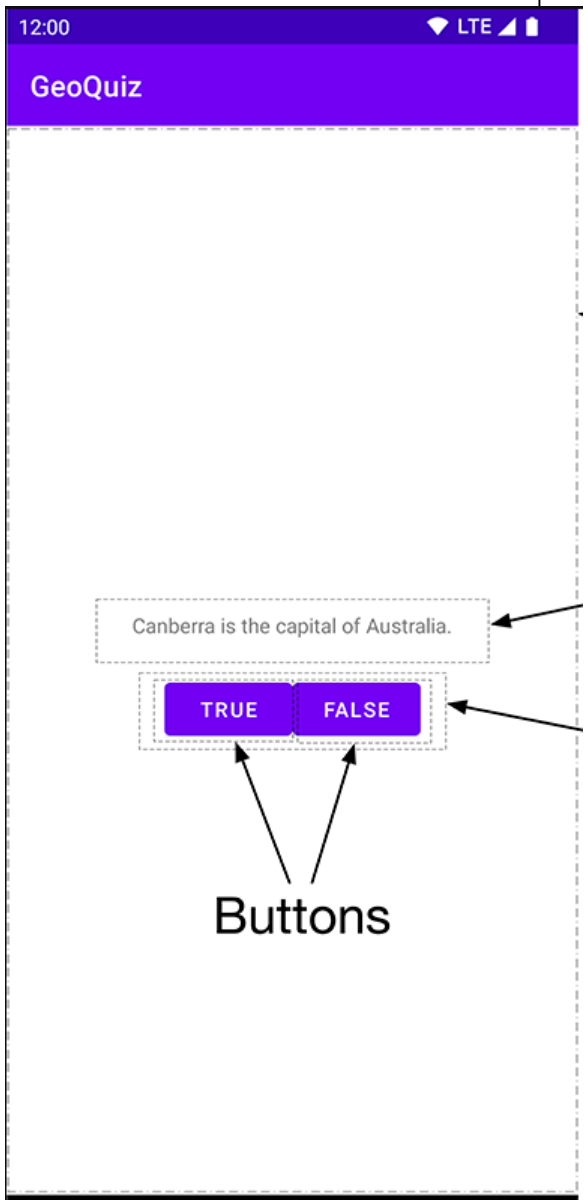


GeoQuiz MainActivity.kt

Android Nerd Ranch (5th Edition), pages 57-63

- Example: print debug message from onCreate method

```
package com.bignerdranch.android.geoquiz

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```



MainActivity.kt

Android Nerd Ranch (5th Edition), pages 57-63

- Debug (d) messages have the form (taking 2 strings)

```
d(tag: String?, msg: String)
```

- E.g.

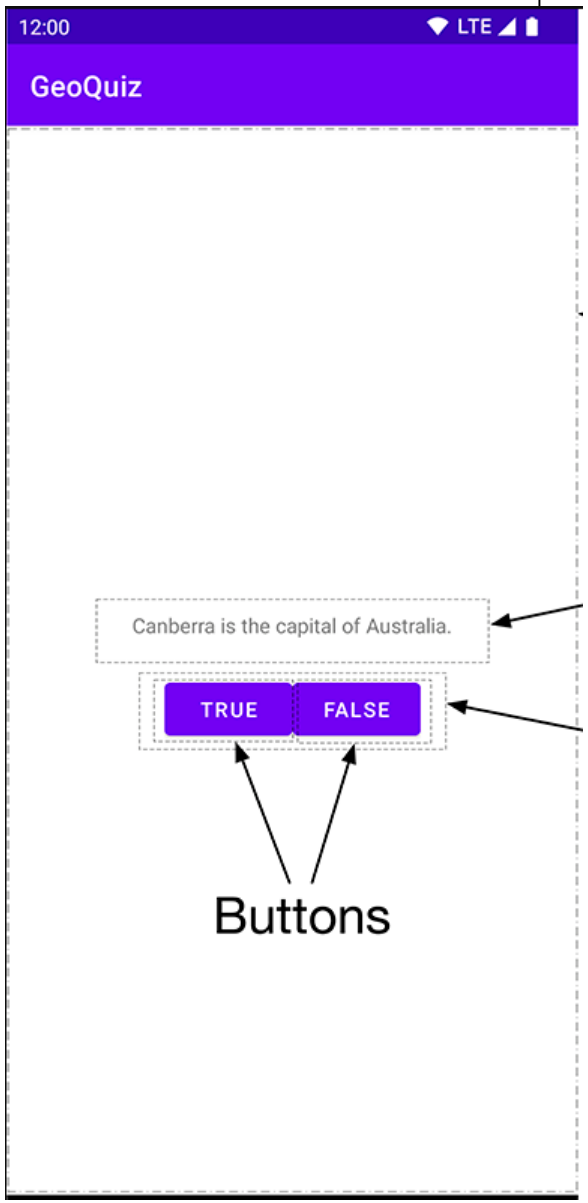
Tag → MainActivity: Message → "onCreate(Bundle) called"

- To declare string for **TAG**

```
private const val TAG = "MainActivity"

class MainActivity : AppCompatActivity() {
    ...
}
```

- Example debug message declaration:

```
Log.d(TAG, "onCreate(Bundle?) called")
```

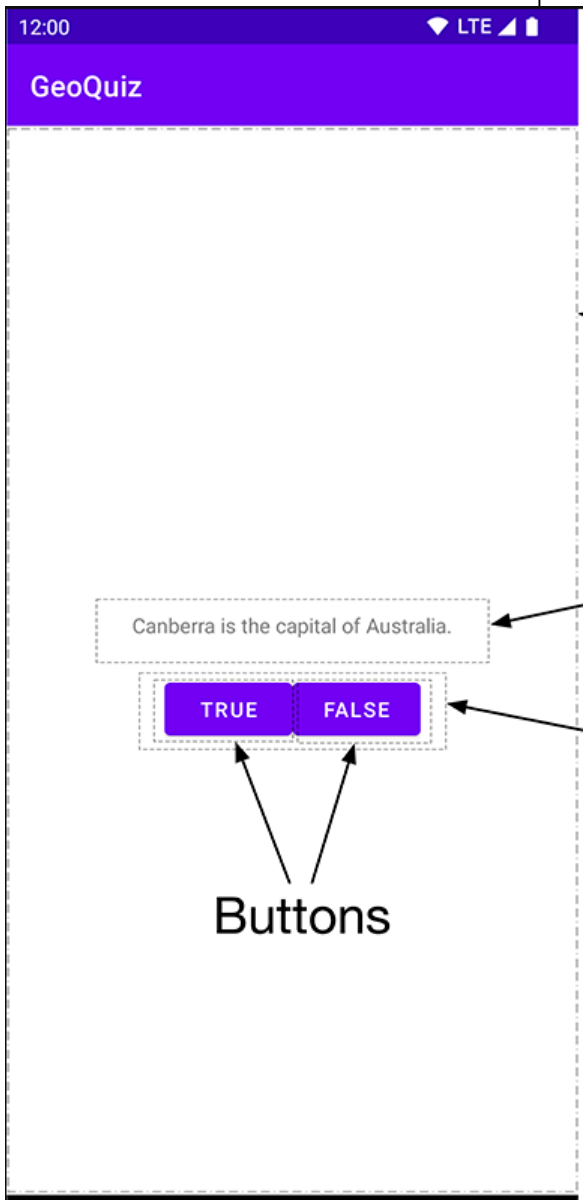


MainActivity.kt

Android Nerd Ranch (5th Edition), pages 57-63

- Insert line to print out debug message

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    Log.d(TAG, "onCreate(Bundle?) called")
    binding = ActivityMainBinding.inflate(layoutInflater)
    setContentView(binding.root)
    ...
}
```

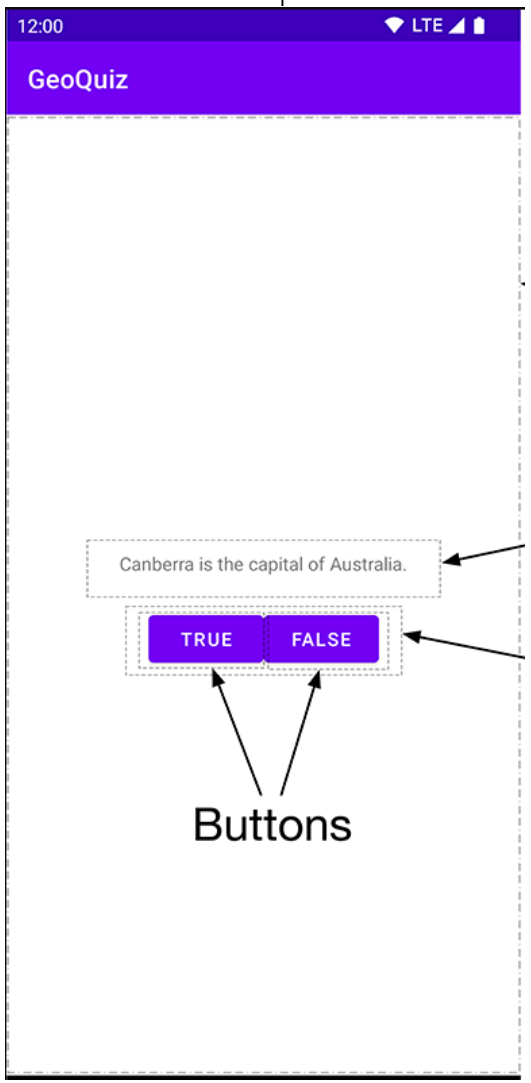


MainActivity.kt

Android Nerd Ranch (5th Edition), pages 57-63

- Can overide more lifecycle methods
- Print debug messages from other lifecycle methods

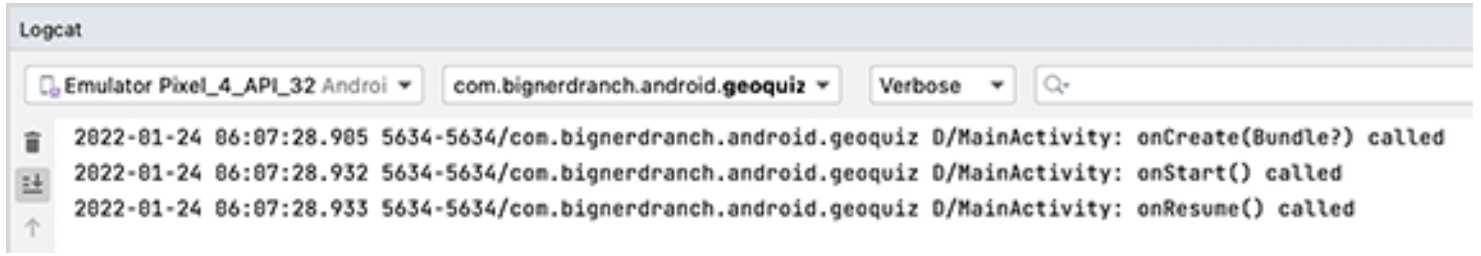
```
override fun onStart() {
    super.onStart()
    Log.d(TAG, "onStart() called")
}

override fun onResume() {
    super.onResume()
    Log.d(TAG, "onResume() called")
}

override fun onPause() {
    super.onPause()
    Log.d(TAG, "onPause() called")
}

override fun onStop() {
    super.onStop()
    Log.d(TAG, "onStop() called")
}

override fun onDestroy() {
    super.onDestroy()
    Log.d(TAG, "onDestroy() called")
}
```

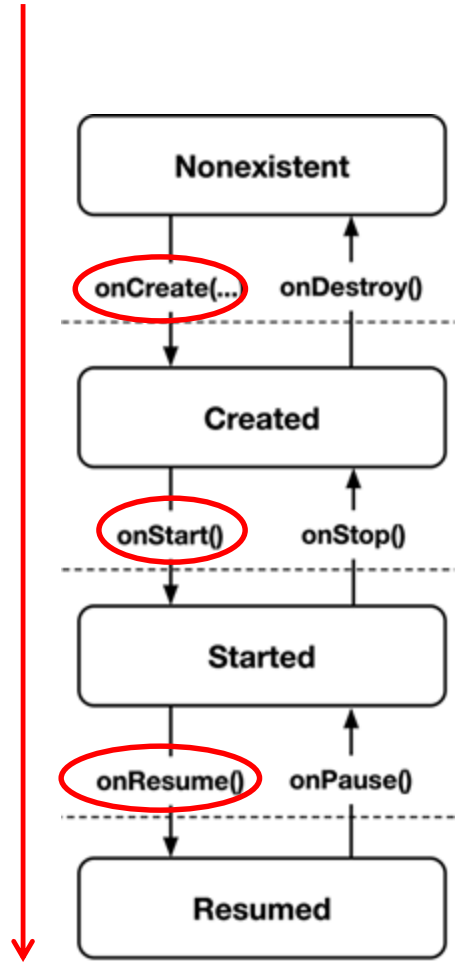


MainActivity.kt Debug Messages

Android Nerd Ranch (5th Edition), pages 57-63

- Launching GeoQuiz app activities **OnCreate, OnStart and onResume** methods

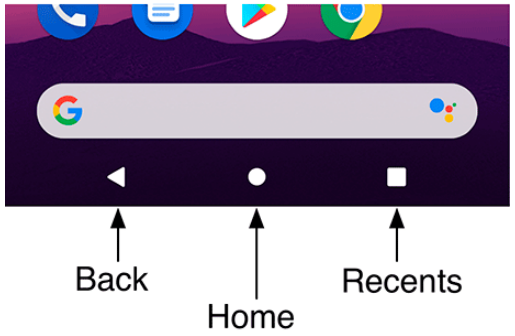


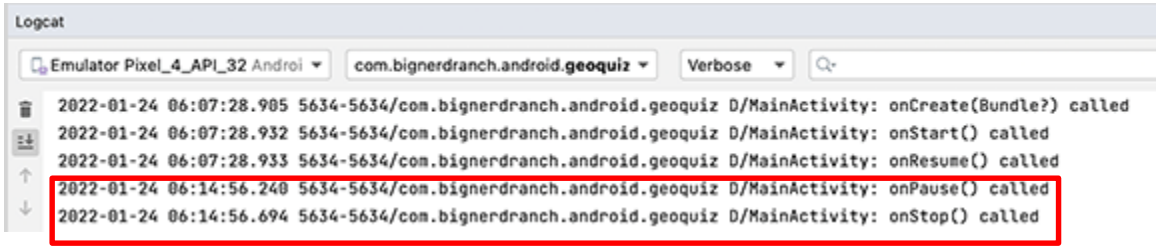


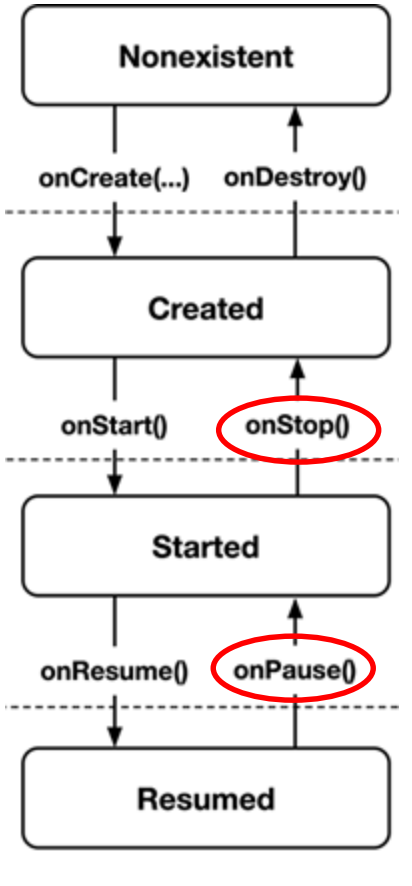
MainActivity.kt Debug Messages

Android Nerd Ranch (5th Edition), pages 57-63

- Pressing the Phone's Home button (leaving app temporarily, might come back), causes Android to pause the activity
- Calls **onPause** and **onStop**



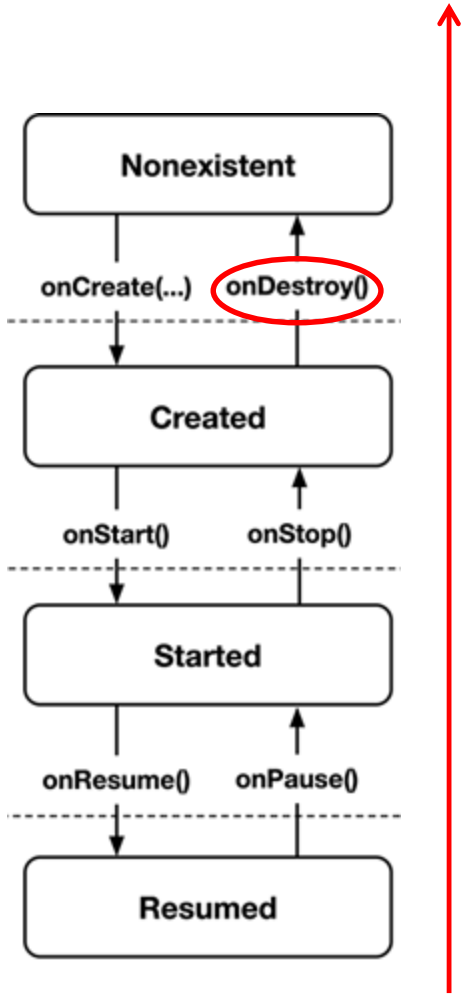


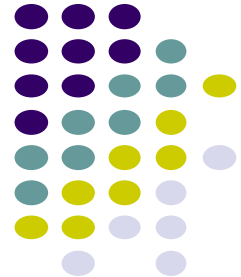


MainActivity.kt Debug Messages

Android Nerd Ranch (5th Edition), pages 57-63

- Closing app destroys the activity (calls **onDestroy**)



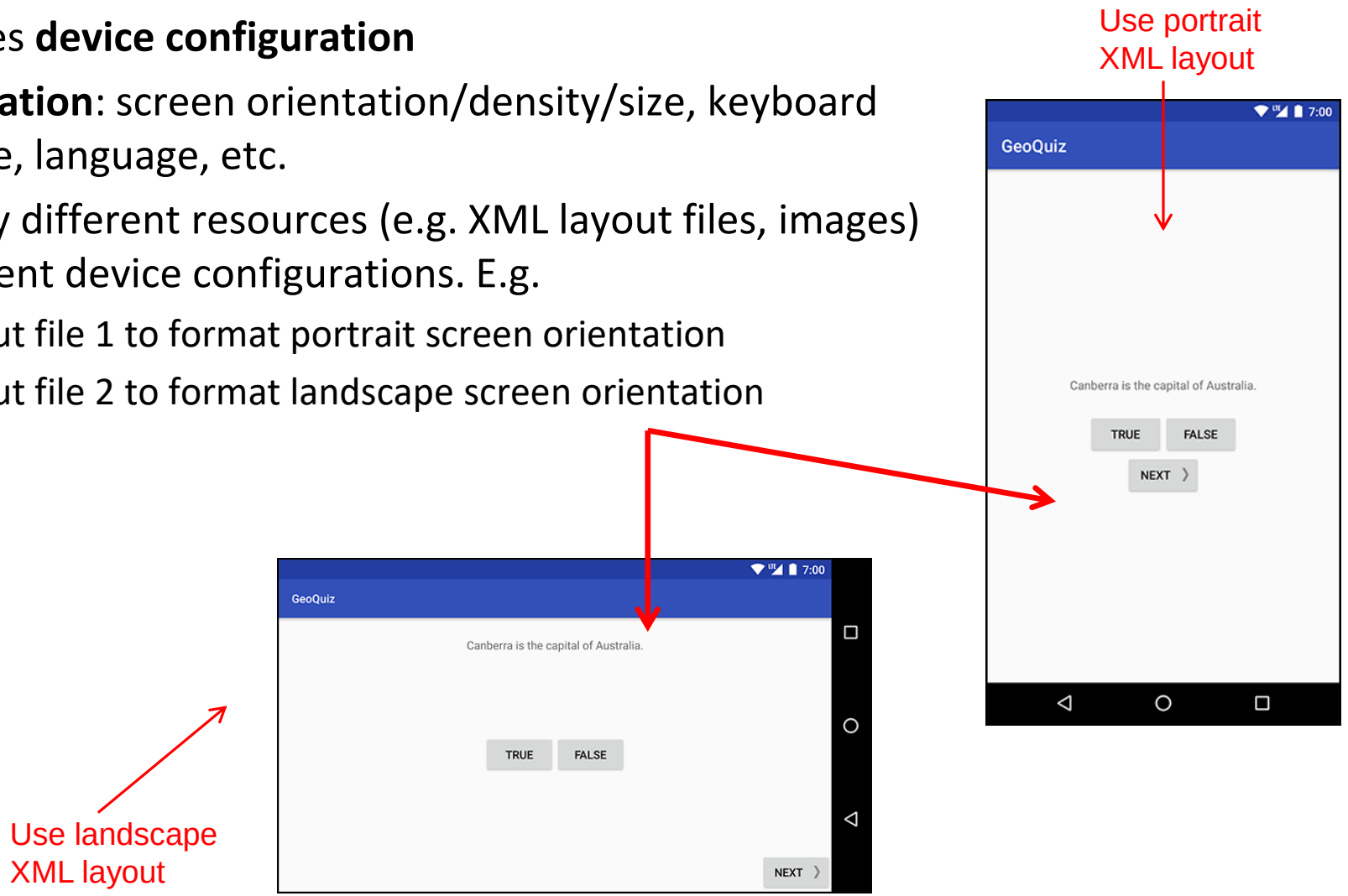


Rotating Device

Rotating Device: Using Different Layouts

Android Nerd Ranch (5th Edition), pages 64-63

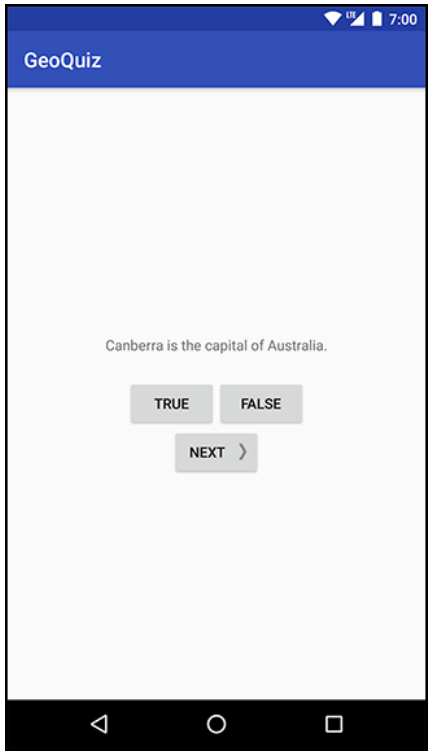
- Rotating device (e.g. portrait to landscape) kills current activity and creates new activity in landscape mode
- Rotation changes **device configuration**
- **Device configuration**: screen orientation/density/size, keyboard type, dock mode, language, etc.
- Apps can specify different resources (e.g. XML layout files, images) to use for different device configurations. E.g.
  - Use XML layout file 1 to format portrait screen orientation
  - Use XML layout file 2 to format landscape screen orientation

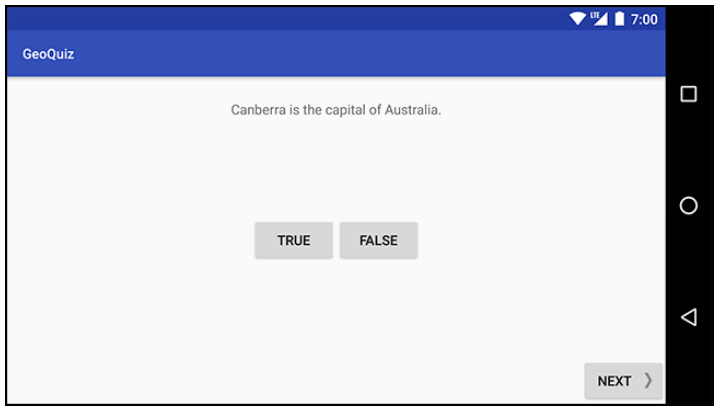


Rotating Device: Using Different Layouts

- **Portrait:** use an XML layout file in **res/layout**
- **Landscape:** use an XML layout file in **res/layout-land/**
- Copy XML layout file (activity_quiz.xml) from **res/layout** to **res/layout-land/** and customize it
- If configuration changes, current activity destroyed, **setContentView (R.layout.activity_quiz)** in **OnCreate( )** called again

onCreate called whenever user switches between portrait and landscape





Persisting App State using ViewModel

Android Nerd Ranch (5th Edition), Chapter 4

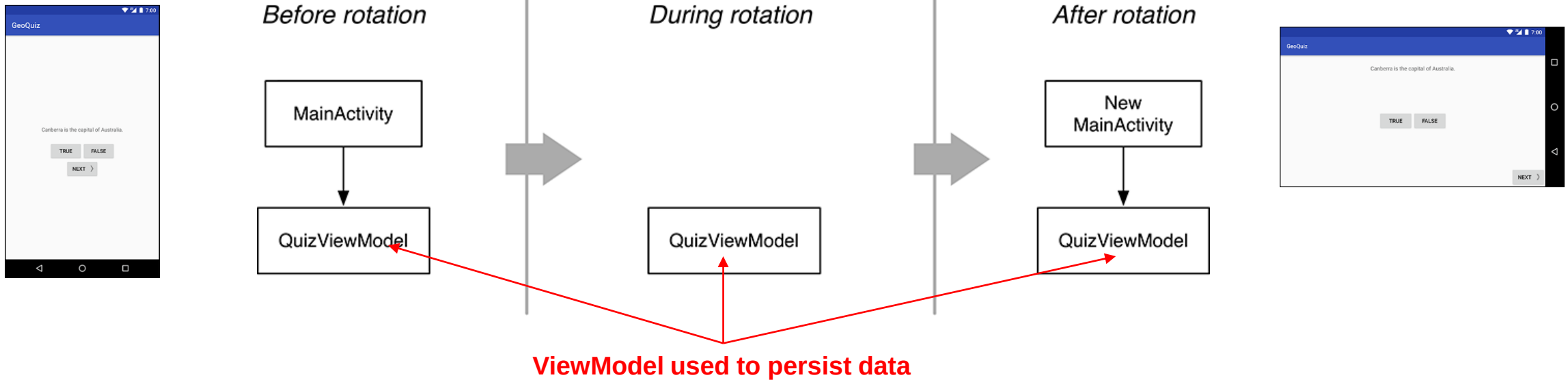
- Rotation changes device configuration, forgets values of app variables (e.g., counters get reset to 0)
  - Unless code is written to persist variables
- **ViewModel:** can persist state, has simple lifecycle
- General idea: Activity can store variables to persist in ViewModel
- First, need to include 2 libraries in dependencies in app/build.gradle

```
plugins {
  id 'com.android.application'
  id 'kotlin-android'
}

android {
    ...
}

dependencies {
    implementation 'androidx.core:core-ktx:1.7.0'
    implementation 'androidx.appcompat:appcompat:1.4.1'
    ...
}
```

Persisting App State using ViewModel

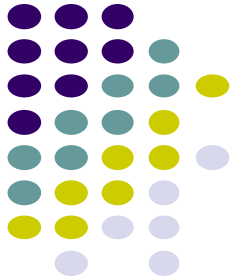
Android Nerd Ranch (5th Edition), Chapter 4

- General idea: Activity can store variables to persist in ViewModel
- ViewModel stays in memory during configuration change (e.g., device rotation)
- During configuration change:
  - Activity instance is destroyed (e.g., in portrait) and re-created (e.g., in landscape)
  - Any associated (or scoped) ViewModels stay in memory
  - When re-created activity (e.g., in landscape) queries associated ViewModel, the initially created instance (e.g., in portrait) is returned



Persisting App State using ViewModel

Android Nerd Ranch (5th Edition), Chapter 4

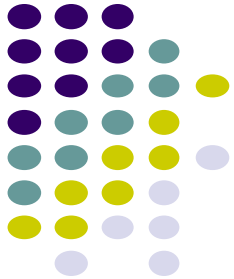


- Create **QuizViewModel.kt** file that contains ViewModel code
- onCleared( ) method is called just before ViewModel is destroyed
  - Can insert cleanup code in onCleared( )

```
private const val TAG = "QuizViewModel"

class QuizViewModel : ViewModel() {

    init {
        Log.d(TAG, "ViewModel instance created")
    }

    override fun onCleared() {
        super.onCleared()
        Log.d(TAG, "ViewModel instance about to be destroyed")
    }
}
```

Persisting App State using ViewModel

Android Nerd Ranch (5th Edition), Chapter 4



- In **MainActivity.kt**, associate activity with instance of **QuizViewModel**

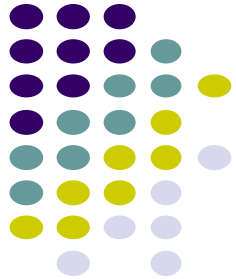
```
class MainActivity : AppCompatActivity() {

    private lateinit var binding: ActivityMainBinding

    private val quizViewModel: QuizViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        setContentView(binding.root)

        Log.d(TAG, "Got a QuizViewModel: $quizViewModel")

        binding.trueButton.setOnClickListener { view: View ->
            checkAnswer(true)
        }
        ...
    }
    ...
```

Declare viewModel

Print debug message “Got a …”

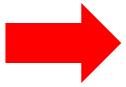
Persisting App State using ViewModel

Android Nerd Ranch (5th Edition), Chapter 4



- Move any variables to persist state from Activity to the ViewModel

```
class MainActivity : AppCompatActivity() {
    ...
    private val questionBank = listOf(
        Question(R.string.question_australia, true),
        Question(R.string.question_oceans, true),
        Question(R.string.question_mideast, false),
        Question(R.string.question_africa, false),
        Question(R.string.question_americas, true),
        Question(R.string.question_asia, true)
    )

    private var currentIndex = 0
    ...
}
```



```
class QuizViewModel : ViewModel() {

    init {
        Log.d(TAG, "ViewModel instance created")
    }

    override fun onCleared() {
        super.onCleared()
        Log.d(TAG, "ViewModel instance about to be destroyed")
    }

    private val questionBank = listOf(
        Question(R.string.question_australia, true),
        Question(R.string.question_oceans, true),
        Question(R.string.question_mideast, false),
        Question(R.string.question_africa, false),
        Question(R.string.question_americas, true),
        Question(R.string.question_asia, true)
    )

    private var currentIndex = 0
}
```

Persisting App State using ViewModel

Android Nerd Ranch (5th Edition), Chapter 4

- Remember to also insert logic to update indices, counters, associated code in ViewModel

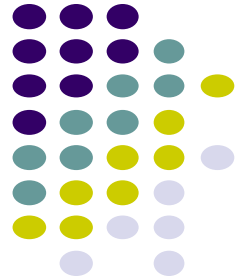
```
class QuizViewModel : ViewModel() {

    private val questionBank = listOf(
        ...
    )

    private var currentIndex: Int = 0

    val currentQuestionAnswer: Boolean
        get() = questionBank[currentIndex].answer

    val currentQuestionText: Int
        get() = questionBank[currentIndex].textResId

    fun moveToNext() {
        currentIndex = (currentIndex + 1) % questionBank.size
    }
}
```

Get current Answer (True or False)

Get current Questions

Persisting App State using ViewModel

Android Nerd Ranch (5th Edition), Chapter 4

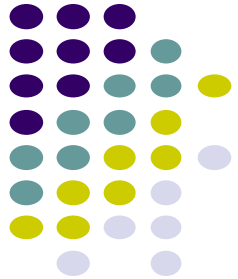


- Update Activity so that updates to counter are now done in viewModel

```
class MainActivity : AppCompatActivity() {
    ...
    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        binding.nextButton.setOnClickListener {
            currentIndex = (currentIndex + 1) % questionBank.size
            quizViewModel.moveToNext()
            updateQuestion()
        }
        ...
    }
    ...
    private fun updateQuestion() {
        val questionTextResId = questionBank[currentIndex].textResId
        val questionTextResId = quizViewModel.currentQuestionText
        binding.questionTextView.setText(questionTextResId)
    }

    private fun checkAnswer(userAnswer: Boolean) {
        val correctAnswer = questionBank[currentIndex].answer
        val correctAnswer = quizViewModel.currentQuestionAnswer
        ...
    }
```

Saving Data Across Process Death

Android Nerd Ranch (5th Edition), Chapter 4

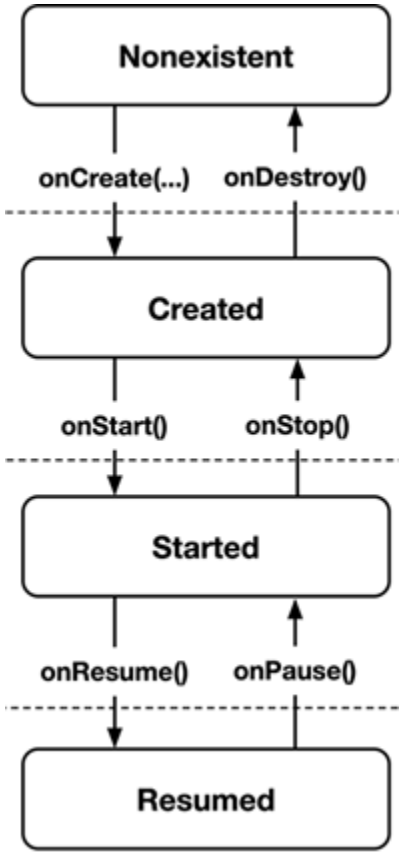


- Android app only guarantees saving state of apps in resumed or started states
- Android OS MAY reclaim/delete data associated with other app states (e.g. stopped)
  - E.g. if user hits home button and goes to watch TV
- To save UI state data, store data in **saved instance data** using **SavedStateHandle**.
- Can
  - Use **SavedStateHandle** like key-value map (store data like integers, strings)
  - Pass a **SavedStateHandle** into a **ViewModel**

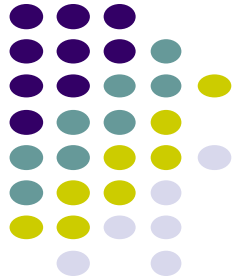
```
private const val TAG = "QuizViewModel"
const val CURRENT_INDEX_KEY = "CURRENT_INDEX_KEY"

class QuizViewModel(private val savedStateHandle: SavedStateHandle) : ViewModel() {
    ...
    private var currentIndex: Int =0
        get() = savedStateHandle.get(CURRENT_INDEX_KEY) ?: 0
        set(value) = savedStateHandle.set(CURRENT_INDEX_KEY, value)
    ...
}
```

Question

- Whenever I watch YouTube video on my phone, if I receive a phone call and video stops at 2:31, after call, when app resumes, it should restart at 2:31.
- How do you think this is implemented?
  - In which Android methods should code be put into?
  - What other Android constructs do I need?
  - How?



Reading Assignment & Skipped Chapters



- **Important:** Read Android Nerd Ranch (ANR), 5th edition, Ch. 2, 3 and 4

- Ch. 5: Debugging Android Apps
  - How to use Android Studio Debugging tools:
    - Android Lint
    - Android Studio Debugger

- Ch 6: Testing
  - Unit testing: Writing and testing small programs (units) within app
  - Ensures new working pieces (units) did not destroy previously working functionality

References

- Android Nerd Ranch, 5th edition