Android Applications

CS 346 Application Development

https://student.cs.uwaterloo.ca/~cs346

Contents

Introduction

Motivation

With the release of the iPhone in 2007, smartphones moved from niche to essential devices very quickly. They have become our essential, personal, computing devices.

A mashup of other devices.

- “An iPod, a phone, an internet communicator” [1]

What makes them unique?

- Touch-screens! Touch input, customizable output.
- Focus on simple, ad hoc interaction.
- A single device for everything.

Design concerns

- Processing efficiency, battery life.
- Security! Applications needed to be sandboxed.

Figure 1: The first iPhone, introduced in Jan 2007. Apple sold more than 6 million devices before replacing it with the iPhone 3G in 2008.

Background

Android was originally developed by Andy Rubin around 2003 as a camera operating system. He refocused on phone operating systems in 2004 and sold Android to Google in 2005.

By Dec 2006, Google was actively developing and testing Android. It was released on the HTC Dream phone in Oct 2008, as a competitor to the iPhone.

Android is extremely successful.

- Based on Linux kernel, with a layered/service architecture.
- Portions are open source, but Google services are closed.
- Broad hardware support e.g., phones, Smart-TVs, e-readers.
- “Billions of devices” shipped.

Figure 2: The first Android phone was the HTC Dream, which launched in October 2008 - approximately 18 months after the first iPhone.

Features

Graphical User Interface

- Installed applications are presented as pages of icons.
- Running applications are typically full-screen.
- Users can navigate through screens, move between activities.
- Tight system integraton with Google services e.g., maps.

Advanced UI Features

- Depend on hardware/vendor, Android can offer side-by-side applications, live regions, PiP and so on.

Characterized by a HUGE variety of devices.

- iOS is limited to Apple hardware, so very few devices need to be supported.
- Android is used by hundreds of vendors.
- Many have forked the OS for their own use e.g., Amazon for Kindle devices.

Architecture

Overview

Android is a layered architecture [2]:

- Applications interact at the Android Framework layer. User requests originate here.
- Requests flow downthrough well formed service APIs.
- The Android Runtime handles runtime behaviours, like memory allocation, garbage collection.

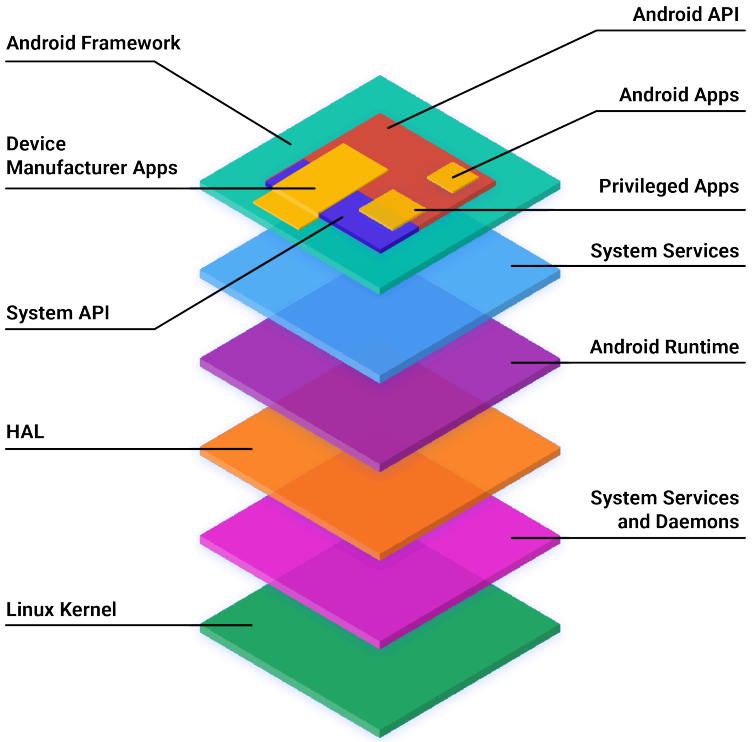


Figure 3: Android System Architecture. Applications usually interact with the Application Framework layer. Taken from the Android Architecture Overview.

Components

Types of Components

Android applications consists of some combination of components. There are base classes that integrate with the underlying system services; you can derive and customize any of these.

Component	Description
Activities	Screens, each with own state and lifecycle.
Fragments	Portions of a screen that can be managed separately.
Services	Provides long-running operations in the background.
Content Providers	Shares data with other applications.
Broadcast Receivers	Listens for system events e.g., phone calls

These components are registered with the OS, and applications can request to use each other's components e.g., you can use an existing `camera` component instead of creating one.

Components

Communication

Components communicate by sending requests to the system as intents.

- An intent is a message delivered to the OS, requesting a specific action or capability e.g., we might create an intent to “take a picture” and ask the OS to fulfil it.
- The OS delivers to a relevant component. This could be within your application, or it might be a component from a completely different application; the OS managed the handoff.
- This is a form of late binding that lets your application take advantage of other installed components, which you might not know about until your application is installed and launched.

Project Structure

The project structure is similar to desktop, with a few small differences:

- Your entry point is not `Main.kt` but instead the `MainActivity` class.
- Application `Manifest` file describes your project structure.
- Android stores resources in the `res` folder structure. There is an API to load them in code.

```
.
├── build
├── kotlin
├── kotlin.bat
├── module.yaml
└── src
    ├── AndroidManifest.xml
    └── MainActivity.kt
```

module.yaml

```
yaml
product: android/app

dependencies:
  - $compose.foundation
  - $compose.material3
  - androidx.activity:activity-
compose:1.7.2
  -
androidx.appcompat:appcompat:1.6.1

settings:
  android:
    namespace: "sample.app"
  compose: enabled
  junit: junit-4
```

Project Structure

MainActivity

Every application has a single main activity. This is the entry point for the application, and represents the main screen for your application.

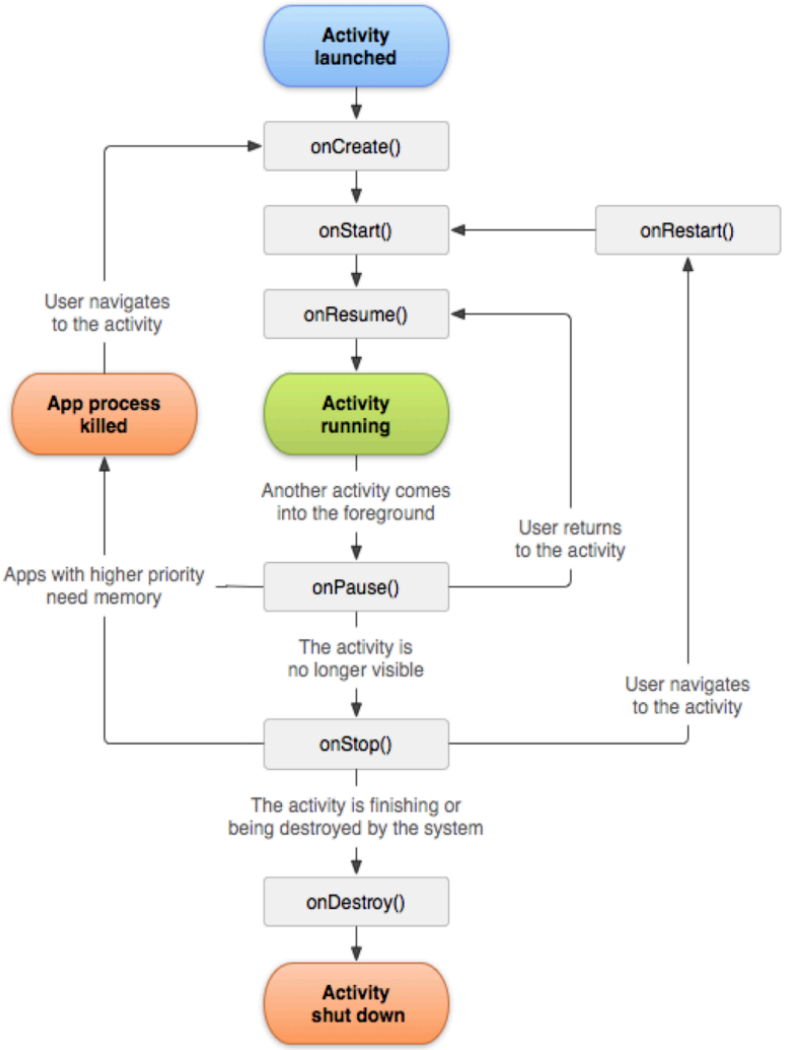
```
class MainActivity : ComponentActivity() {
  // entry point for any activity is the onCreate() method
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    // setup other components
    val database= getRoomDatabase(this)
    val taskModel = TaskModel(database.taskDao())
    val viewModel = TaskViewModel(taskModel)

    // specify a composable scope
    setContent {
      TaskView(viewModel) // top-level View/Composable
    }
  }
```

Project Structure

Activity Lifecycle

Activities have a lifecycle, with these loops:

1. The entire lifetime of an activity happens between the first call to `onCreate()` through to a single final call to `onDestroy()`.
2. The visible lifetime happens between a call to `onStart()` until a corresponding call to `onStop()`. The activity is visible, but may not be in the foreground.
3. The foreground lifetime of an activity happens between a call to `onResume()` until a corresponding call to `onPause()`. During this time the activity is in visible, active and interacting with the user.



Project Structure

Be aware of this lifecycle!

What are the implications?

- Activities can be restarted when the OS decides that it needs to reclaim resources (uncommon), or when you rotate the device (common!)
- Restarting activities means relaunching and losing data.

How do you avoid this?
- Save and restore data manually
- Override the `onPause()` and `onResume()` methods and save data in a `Bundle`.
- Use a `ViewModel` to store your state. Android will automatically save and restore `ViewModel` data!

https://developer.android.com/topic/libraries/architecture/viewmodel

Application Manifest

Every Android project has a single Application manifest file named `AndroidManifest.xml`. This is an XML file contains information required by the Android build tools and operating environment to manage your application.

These include:

- Identifying the `MainActivity` which launches on startup i.e., `main` method.
- Identifying the name and icon to use for your application.
- Determining the location of resources in your project.
- Defining permissions that the application requires to execute.

> ⚠ **Warning**
>
> For simple projects, you can probably just use the defaults. The main reasons to change this file would be to change your starting activity, add intents, or to request network or file permissions for your application.

Application Manifest

```
<application
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:theme="@style/Theme.Mmandroid"
    tools:targetApi="31">
  <activity
    android:name="ca.uwaterloo.mm.MainActivity"
    android:exported="true"
    android:label="@string/app_name"
    android:theme="@style/Theme.Mmandroid">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
  </activity>
</application>
```

Views & Activities

How do we build a UI using activities? Google's sugggestions have changed over time.

1. Each View is an Activity (early-Android)
   - Every screen is represented by a corresponding Activity.
   - Use an Intent (message to the OS) to swap between them.
   - Not recommended. It's very slow.

2. Activities + Fragments (old Android)
   - Activities are each built from components called Fragments.
   - Load the Activity, then use fragments that together build a screen.
   - Not recommended! An improvement, but still slow.

3. One Activity, which you populate (new Android)
   - Use your MainActivity as a container for Compose views.
   - Navigation code/libraries just chooses which View to launch.
   - Recommended.

Jetpack Compose

Views

A `view` is a custom user interface class for a single screen. In the example below, we pass in the viewModel that contains the data we wish to display.

```
@Composable
fun TaskView(viewModel: TaskViewModel) {
    Scaffold(
        topBar = {
            Toolbar(
                addHandler = { viewModel.showAddDialog = true },
                editHandler = { viewModel.showEditDialog = true },
                deleteHandler = {
                    viewModel.delete(viewModel.selectedTask)
                    viewModel.selectedTask = null
                }
            )
        },
        bottomBar = {  },
    ) { /* add screen contents here */
    }
}
```

Composables

Finding Composables

Your UI is basically a tree of composables. Everything that you wish to reflect in the UI needs to be a composable.

- Containers
- Elements

Where do they all come from?

- Jetpack Compose includes all of the Android composables, many of which also work on desktop.
- Compose Multiplatform contains the desktop versions.
- There are online collections where you can find more composables that developers have made available online.

Composables

Scaffold

A scaffold [scaffold] divides the screen into top, bottom and content regions.

```
Scaffold(
  topBar = {
    Row(
      modifier = Modifier.height(40.dp)
                 .fillMaxWidth()
                 .background(Color.Green),
    ) {
      Text(fontSize = 21.sp,
           text = "Scaffold topBar")
    }
  },
  ) {
    Column(
      modifier = Modifier.fillMaxSize(),
    ) { Text("Hello, World!") }
  }
}
```

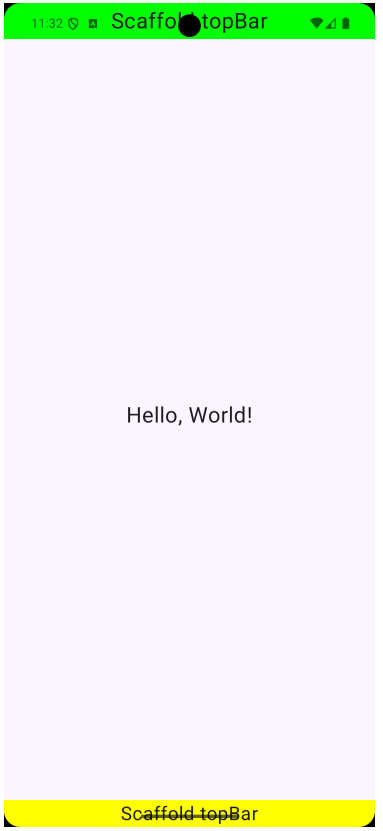
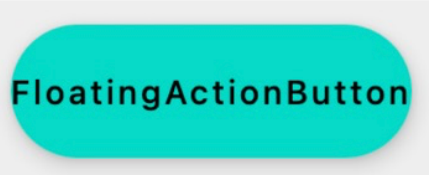


Figure 4: Scaffold with top, bottom and content regions.

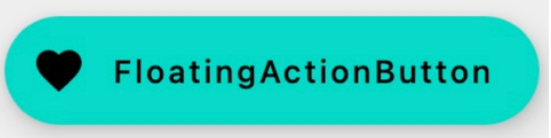
Floating Action Buttons

A floating action button is meant to trigger the main action on a screen, and is often positioned in the bottom-right corner. It is customizable and can contain images.

```
@Composable
fun FABDemo() {
    FloatingActionButton(onClick = { }) {
      Text("FloatingActionButton")
  }
}
```




```
@Composable
fun FABImageDemo() {
  ExtendedFloatingActionButton(
    icon =
{ Icon(Icons.Filled.Favorite,"") },
    text = { Text("FloatingActionButton") },
    onClick = { /*do something*/ }
  )
}
```



Card

A card is meant to be a container for a “single coherent piece of content”. [3]

```
@Composable
fun CardDemo() {
  Card(
    modifier = Modifier.fillMaxWidth()
                       .padding(15.dp)
                       .clickable{ },
        elevation = 10.dp
  ) {
      Column(modifier = Modifier.padding(15.dp)) {
        Text("Jetpack Compose Playground")
        Text("Now you are in the Card section")
      }
  }
}
```

Bibliography

[1] Steve Jobs, “Steve Jobs Introducing The iPhone At MacWorld 2007.” [Online]. Available: https://www.youtube.com/watch?v=x7qPAY9JqE4

[2] Google Inc., “Android Developer Documentation.” [Online]. Available: https://source.android.com/docs

[3] Google Inc., “Jetpack Compose Documentation.” [Online]. Available: https://developer.android.com/develop/ui/compose/documentation