Embedded AOSP

Lecture 2. Android App Development

09-13 July

Polytechnic University of Bucharest

Partnered with Google

Android Application Development

- Basic API components: activities, services, intents
- Times are changing
  - Old way: Java + XMLs
  - New way: Kotlin + Jetpack Compose
- Android Studio
- Build a “virtual” Led control app (part 1)

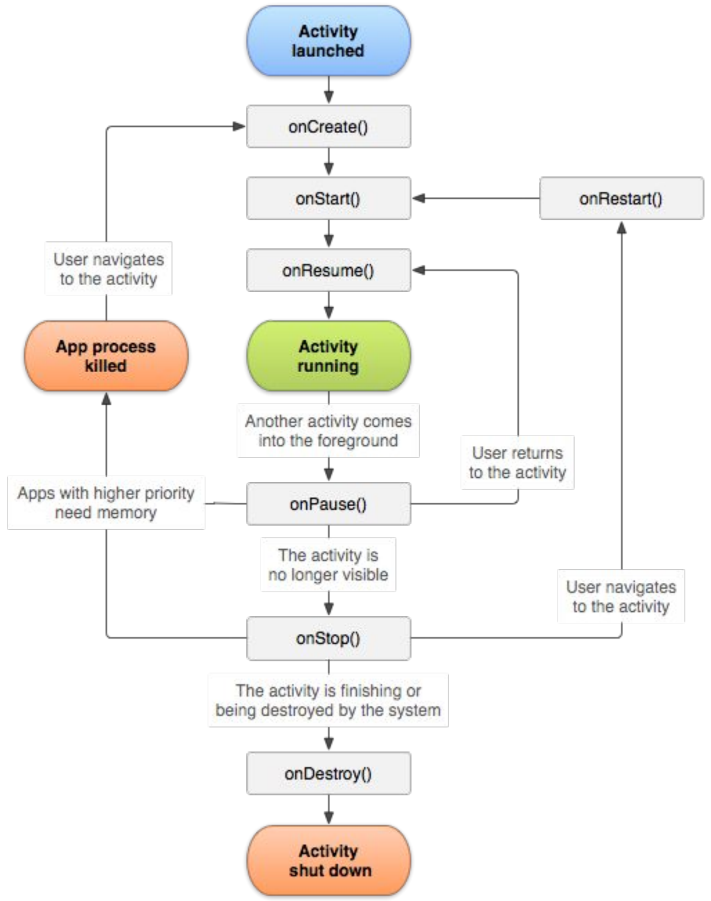
Activity

- An activity controls a screen within an app
- Receives events, renders the UI
- Started using “intents” by other applications (e.g., the Launcher)

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Activity Lifecycle

- When an activity is requested, it is either created, re-started or resumed
- Android may stop/suspend/kill an activity whenever it wants
- An activity must save its state to persistent storage when the appropriate method is called!


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Intents

- Messages for communication between components
- Used to launch activities or services
- May include arguments, e.g.:

```kotlin
val intent=Intent(Intent.ACTION_SEND).apply {
    type="text/plain"
    putExtra(Intent.EXTRA_EMAIL, arrayListOf("youremailid@gmail.com"))
    putExtra(Intent.EXTRA_SUBJECT, "This is the subject of the mail")
    putExtra(Intent.EXTRA_TEXT, "This is the text part of the mail")
}
if (intent.resolveActivity(packageManager)!=null){
    startActivity(intent)
}
```

Services, broadcast receivers, content providers

- Service: background activity without UI
    - *e.g., music, downloads*
    - Started Service: run until stopped
    - Bound Service: allows components to bind to it
        - keeps running until last app finishes using it
- Broadcast Receivers:
    - Respond to broadcast messages (e.g., battery low, airplane mode)
    - Lightweight, shall not run long operations!
- Content Providers:
    - Manage shared app data (e.g., contacts, images)

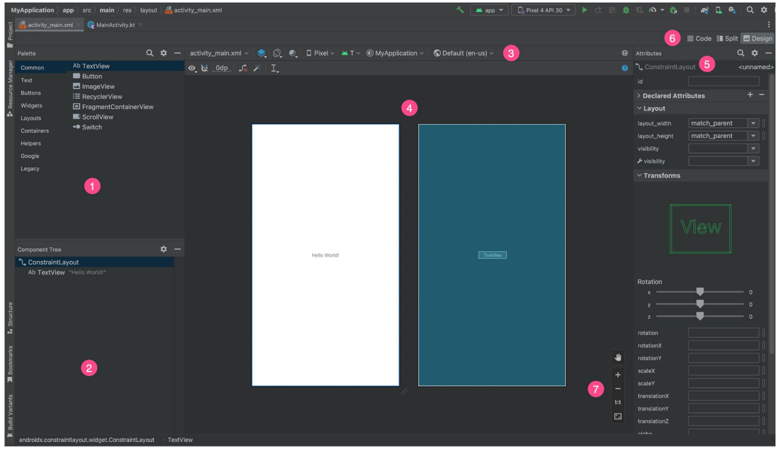
Android Manifest XML

- Main descriptor of an app
- Icon & label
- Components (activities, services, content providers)
- Required API versions
- Permissions, intent filters etc.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:icon="@drawable/app_icon.png">
        <activity
android:name="com.example.project.ExampleActivity"
                  android:label="@string/example_label">
        </activity>
        ...
    </application>
    <uses-feature
android:name="android.hardware.sensor.compass"
                  android:required="true" />
</manifest>
```

UI: The Traditional Way (XML)

- XML Layouts: Your app's UI is defined in XML files located in app/src/main/res/layout/
- Layout Editor: Drag and drop UI elements (Widgets) onto the screen
  - TextView, EditText, Button etc.
- Give names to components and reference them in code using `findViewById()`



UI: The Modern Way (Jetpack Compose)

- **Jetpack Compose** is a declarative UI toolkit for building native Android apps.
  - Describe your UI in Kotlin lang
  - The framework handles the rendering and property updates
- Advantages:
  - **Declarative**: describe what the UI should show, not how to create and update it
  - **Less Code**: no need for XML FTW!
  - **Kotlin-based**: UI and logic in the same language, allowing for seamless integration

```
@Composable
fun GreetingScreen() {
  var name by remember { mutableStateOf("") }
  Column(modifier = Modifier.padding(16.dp),
    horizontalAlignment =
Alignment.CenterHorizontally)
  {
    Text(text = "Hello, ${if (name.isNotBlank())
name else "World"}!")
    Spacer(modifier = Modifier.height(8.dp))
    OutlinedTextField(
      value = name,
      onValueChange = { name = it },
      label = { Text("Name") }
    )
  }
}
```