

# **Computer Science Large Practical: Android concepts and Kotlin programming**

---

Stephen Gilmore  
School of Informatics  
September 28, 2017

1. Android concepts
2. Android projects
3. Android Studio

# Android concepts

---

## Activities and contexts

- An Android app is split up into a number of different *activities*, which are subclasses of `android.app.Activity`, or subclasses of that class, such as `android.support.v7.app.AppCompatActivity`.

```
java.lang.Object
└─ android.content.Context (abstract class)
    └─ android.content.ContextWrapper
        └─ android.view.ContextThemeWrapper
            └─ android.app.Activity
```

- An activity represents a single screen with a user interface.
- One activity can invoke another. Every `Activity` is a `Context`.

# Android activities

- Activities differ in nature from the main class of a Kotlin application, in that it must be possible to **pause, suspend, and resume them** and have the app take action depending on which of these events happens.
- The allowable calls to methods such as
  - **onCreate()**,
  - **onStart()**,
  - **onResume()**,
  - **onPause()**,
  - **onStop()**,
  - **onRestart()**, and
  - **onDestroy()**.

make up the Android activity lifecycle.

## Sample onCreate method — create UI components

```
import kotlinx.android.synthetic.main.activity_main.*  
  
...  
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main) // load res/layout/activity_main.xml  
        setSupportActionBar(toolbar)  
  
        fab.setOnClickListener { view ->  
            Snackbar.make(view, "Replace with your own action",  
                Snackbar.LENGTH_LONG)  
                .setAction("Action", null).show()  
        }  
    }  
}
```

## Sample onCreate method — create UI components

```
import kotlinx.android.synthetic.main.activity_main.*  
  
...  
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main) // load res/layout/activity_main.xml  
        setSupportActionBar(toolbar)  
  
        fab.setOnClickListener { view ->  
            Snackbar.make(view, "Replace with your own action",  
                Snackbar.LENGTH_LONG)  
                .setAction("Action", null).show()  
        }  
    }  
}
```

`{ v -> exp }` is Kotlin syntax for a lambda (anonymous function).

# Application logic and user interface

- Android projects separate application logic (coded in Kotlin) from the user interface presentation layer (coded in XML).
- This **separation of concepts** means that the application logic does not get cluttered with presentation layer details about fonts, colours and positions of buttons in the user interface.
- Kotlin uses **data binding** to link XML variables to Kotlin values using the **Kotlin Android Extensions** framework.
- Data binding eliminates run-time lookup of XML variable via **findViewById()**, and thus a potential source of run-time errors.



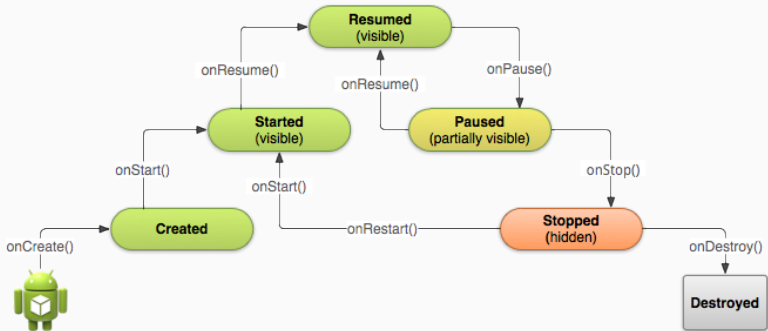
# Sample toolbar and button definition in XML

res/layout/activity\_main.xml

```
<android.support.v7.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    app:popupTheme="@style/AppTheme.PopupOverlay" />
```

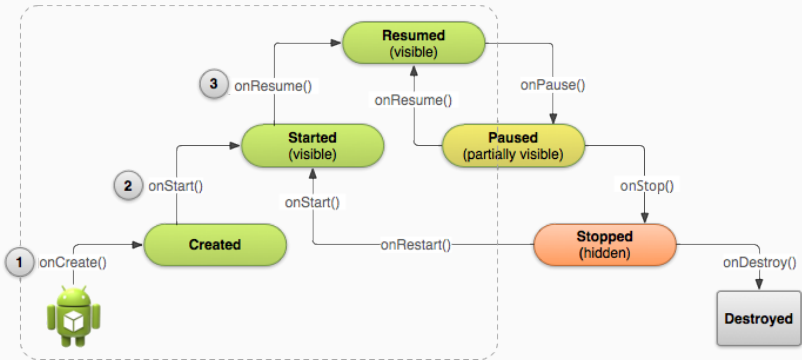
```
...
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="bottom|end"
    android:layout_margin="@dimen/fab_margin"
    android:tint="@android:color/white"
    app:srcCompat="@android:drawable/ic_input_add" />
```

# Android Activity lifecycle



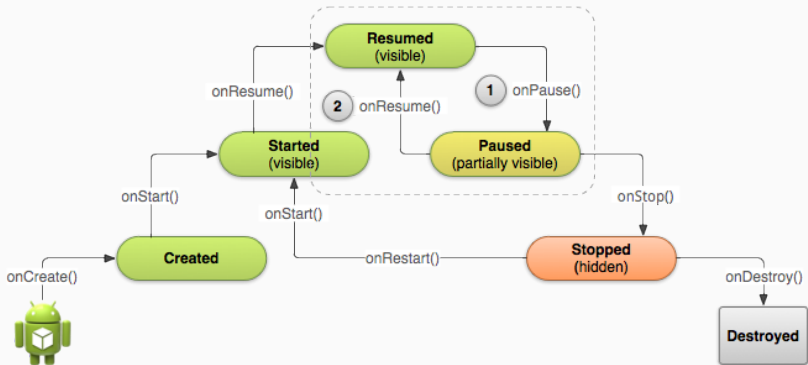
From <https://developer.android.com/training/basics/activity-lifecycle/starting.html>

# Android Activity lifecycle (create)



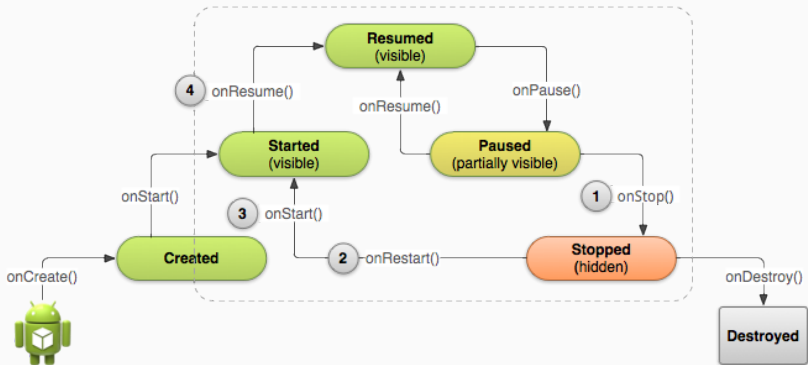
From <https://developer.android.com/training/basics/activity-lifecycle/starting.html>

# Android Activity lifecycle (paused)



From <https://developer.android.com/training/basics/activity-lifecycle/pausing.html>

# Android Activity lifecycle (stopping)



From <https://developer.android.com/training/basics/activity-lifecycle/stopping.html>

# Android Activity lifecycle (saving state)



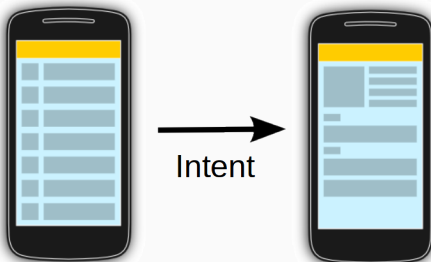
From <https://developer.android.com/training/basics/activity-lifecycle/recreating.html>

# Adding a new Activity

- Most apps have more than one **Activity**.
- Adding a new **Activity** (with File → New → Activity):
  - adds a new Kotlin class file,
  - adds a new XML layout file,
  - add the required `<activity>` element in **AndroidManifest.xml**,and may add other files as needed for specific types of activity.

# Using Intents

- An *intent* of `android.content.Intent` is a messaging object which can be used to communicate with another app component such as another `Activity`.



---

Image from

<http://www.vogella.com/tutorials/AndroidIntent/article.html>



# Using Intents

- You can start a new instance of an **Activity** by passing an **Intent** to **startActivity()**.
- The **Intent** describes the activity to start and carries any necessary data.
- If a result is expected then **startActivityForResult()** is called instead.
- An **Intent** can also be used to start a **Service** of class **android.app.Service**.

## Simple switch to another activity

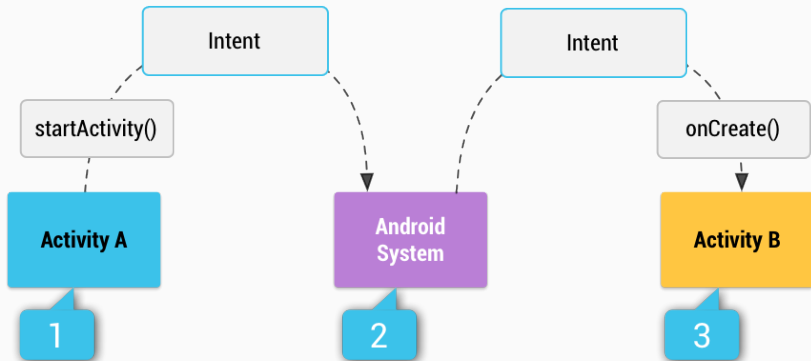
```
private fun switchToMap() {  
    val intent = Intent(this, MapsActivity::class.java)  
    startActivity(intent)  
}
```

## Simple switch to another activity

```
private fun switchToMap() {  
    val intent = Intent(this, MapsActivity::class.java)  
    startActivity(intent)  
}
```

- The *class literal* syntax `ClassName::class` returns a value of class `kotlin.reflect.KClass`.
- The projection `.java` returns a Java `java.lang.Class` instance corresponding to the given `KClass` instance.

# One mechanism of activity starting another



From [https:](https://developer.android.com/guide/components/intents-filters.html)

[//developer.android.com/guide/components/intents-filters.html](https://developer.android.com/guide/components/intents-filters.html)

## Passing information to another activity (sender)

```
import kotlinx.android.synthetic.main.content_main.*  
class MainActivity : AppCompatActivity() {  
    companion object {  
        const val EXTRA_MESSAGE = "com.example.myapp.MESSAGE"  
    }  
    fun sendMessage(view: View) {  
        val intent = Intent(this, DisplayMessageActivity::class.java)  
        val message = editText.text.toString() // editText defined in content_main.xml  
        intent.putExtra(EXTRA_MESSAGE, message)  
        startActivity(intent)  
    }  
}
```

## Passing information to another activity (sender)

```
import kotlinx.android.synthetic.main.content_main.*
class MainActivity : AppCompatActivity() {
    companion object {
        const val EXTRA_MESSAGE = "com.example.myapp.MESSAGE"
    }
    fun sendMessage(view: View) {
        val intent = Intent(this, DisplayMessageActivity::class.java)
        val message = editText.text.toString() // editText defined in content_main.xml
        intent.putExtra(EXTRA_MESSAGE, message)
        startActivity(intent)
    }
}
```

The companion object syntax gives us MainActivity.EXTRA\_MESSAGE

The const val syntax is for compile-time constants of simple type.

## Passing information to another activity (receiver)

```
class DisplayMessageActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_display_message)  
  
        val intent = getIntent() // Get the message from the intent  
        val message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE)  
  
        // Create the text view  
        textView.setTextSize(40F)  
        textView.setText(message)  
  
        if (textView.parent != null)  
            (textView.parent as ViewGroup).removeView(textView)  
        setContentView(textView)  
    }  
}
```

“obj as class” is Kotlin syntax for a cast.

# Android projects

---



# Android projects

- Android projects contain a mix of Kotlin and XML code in a structured project which contains
  - manifests** Contains the *AndroidManifest.xml*, file which provides essential information about your app to the Android system, to allow it to run your code.
  - java** Contains the *Kotlin source code files*, separated by package names, including *JUnit* test code.
  - res** Contains *all non-code resources*, such as XML layouts, UI strings, and bitmap images.
- Java code describing resources is automatically generated from XML source code by Android Studio.

# Android build files

- Android Studio uses the **Gradle** build system which specifies Android version requirements and app dependencies.

...


```
dependencies {  
    implementation fileTree(dir: 'libs', include: ['*.jar'])  
    implementation 'com.android.support:appcompat-v7:26.1.0'  
    implementation 'com.android.support.constraint:constraint-layout:1.0.2'  
    implementation 'com.android.support:design:26.1.0'  
    implementation 'com.google.android.gms:play-services-maps:11.4.0'  
    implementation 'com.android.support:support-v4:26.1.0'  
    testImplementation 'junit:junit:4.12'  
    androidTestImplementation('com.android.support.test.espresso:espresso-core:3.0.1', {  
        exclude group: 'com.android.support', module: 'support-annotations'  
    })  
    implementation "org.jetbrains.kotlin:kotlin-stdlib-jre7:$kotlin_version"  
}
```

# Android Studio

---

# Android Studio

- Android Studio is the official Integrated Development Environment (IDE) for Android app development. It is based on **JetBrain's IntelliJ IDEA**.
- Because it is an Android-specific development environment, Android Studio can make suggestions regarding issues such as missing import statements.



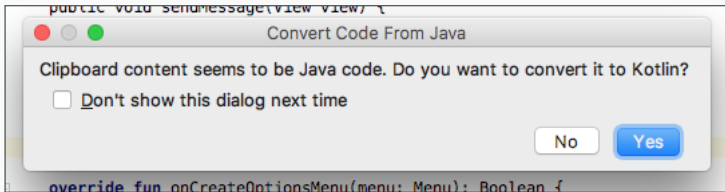
```
val EXTRA ? android.view.View? \u2197 app.MESSAGE"
fun sendMessage(view: View) {
    val intent = Intent(this, DisplayMessageActivity::class.java)
```

The screenshot shows a code editor with a Kotlin snippet. A blue tooltip is visible over the text `? android.view.View?`, suggesting the type `android.view.View?`. The code includes a function `sendMessage` that takes a `View` parameter and creates an `Intent` to start `DisplayMessageActivity`.

- A helpful introduction to Android Studio is available at <https://developer.android.com/studio/intro>

# Android Studio

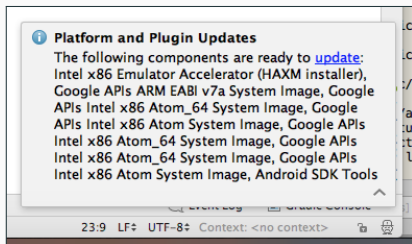
- Android Studio will also offer to convert Java code fragments into Kotlin syntax.



- This can be helpful, but it is only a syntax-driven translation of a fragment of Java code. There might be a better way to achieve the same effect in Kotlin using other language features. For example, the converter will translate a Java call to `findViewById()` to a Kotlin call to `findViewById()` and not suggest that using `data binding` can eliminate this call.

# Platform updates

- Android Studio and the Android APIs and device emulators are *active, current software projects*. It is quite usual when starting up Android Studio to see that updates are available for some of the components that you use.



- We recommend applying these as they become available.

# Links

- <https://developer.android.com/> — Android information
- <https://developer.android.com/studio/> — to download Android Studio
- <https://developer.android.com/develop/> — Android developer documentation
- <https://kotlinlang.org/docs/reference/> — Kotlin reference
- Android studio 3 - Create hello world App in Kotlin, Hitesh Choudhary, <https://youtu.be/-nz-zwfhrLg>
- Kotlin Tutorial, Derek Banas, [https://youtu.be/H\\_oGi8uuDpA](https://youtu.be/H_oGi8uuDpA)