Context & Kotlin Scope Functions

By Muhammad Haris Bin Abid

Context

What is Context in Android?

- **Definition**: An interface to global information about the application environment
- **Purpose**: Provides access to application-specific resources and classes
- **Importance**: Almost all Android components require Context to function

Context: The Bridge to Android System

Gives access to:

- Resources (strings, drawables, layouts)
- System services (WindowManager, LayoutInflater)
- File operations (opening/creating files)
- Database operations (SQLite)
- Application preferences
- Starting activities and services

Types of Context

Application Context

- Tied to the application lifecycle
- Remains available as long as the application is running
- Obtained via getApplicationContext()
- **Best for**: Long-lived operations, singletons, background tasks

Activity Context

- Tied to an Activity's lifecycle
- Only valid while the Activity is active
- Obtained via this in Activity or context in many callbacks
- **Best for**: UI operations, dialogs, layouts, UI-related permissions

Common Context Uses

- // Access resources
- val appName = context.getString(R.string.app_name)
- val drawable = ContextCompat.getDrawable(context, R.drawable.icon)
- val color = ContextCompat.getColor(context, R.color.colorPrimary)

- // Access system services
- val layoutInflater = context.getSystemService(Context.LAYOUT_INFLATER_SERVICE) as LayoutInflater
- val connectivityManager = context.getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager

- // File operations
- val file = context.getFileStreamPath("myfile.txt")
- val cacheFile = File(context.cacheDir, "cached_data.tmp")

Common Context Mistakes

Memory Leaks

```
// ❌ BAD: Storing Activity context in a singleton
object MySingleton {
    private lateinit var context: Context  // Can cause memory leak!

    fun initialize(context: Context) {
        this.context = context  // If this is an Activity context, it can't be garbage collected
    }
}

// ✅ GOOD: Using Application context instead
object MySingleton {
    private lateinit var context: Context

    fun initialize(context: Context) {
        this.context = context.applicationContext  // Safe to store
    }
}
```

Context Safety Tips

- Use **applicationContext** for singletons and long-running operations
- Never store Activity contexts in static variables
- Consider the lifecycle of your context
- Pass the right type of context based on your needs
- Be careful with inner classes that may hold implicit references to Activity contexts

Introduction to Kotlin Scope Functions

Introduction to Kotlin Scope Functions

- Special functions that create a temporary scope for an object
- Makes code more concise and readable
- Each has a different purpose and use case
- Main functions: let, run, with, apply, and also

The **let** Function

Purpose: Execute code if object is not null; scope transformation

Context object available as: it (can be renamed)

Return value: Lambda result

Use when: Working with nullable objects, transforming objects

Let - Basic Syntax

```
nullable?.let {
    // 'it' refers to the non-null value of nullable
    // Only executes if nullable is not null
}

// Practical example
findViewById<TextView>(R.id.textView)?.let {
    it.text = "Hello World"
    it.visibility = View.VISIBLE
}
```

More let Examples

```
// Using a named parameter instead of 'it'
user?.let { user ->
    nameTextView.text = user.name
    emailTextView.text = user.email
}
// Transforming values
val length = nullableString?.let { it.length } ?: 0

// Chaining let calls
getUser()?.let { user ->
    getAddress(user)?.let { address ->
        updateUserAddress(user, address)
    }
}
```

The run Function

Purpose: Execute block of code on an object; compute a result

Context object available as: this (implicit)

Return value: Lambda result

Use when: You want to compute a result based on the objec

Basic syntax

```
val result = someObject.run {
    // 'this' refers to someObject (implicit)
    // Do something with the object
    // Return a result
}

// Practical example with Context
val displayMetrics = context.run {
    val metrics = resources.displayMetrics
    "${metrics.widthPixels} x ${metrics.heightPixels}"
}
```

More **run** Examples

```
// Computing a value from an object
val userSummary = user.run {
    "Name: $name, Age: $age, Email: $email"
}
// Configuring an object and returning a different result
val isAdult = person.run {
    println("Checking if $name is an adult")
    age >= 18
}
// Using run with Context
val density = context.run {
    resources.displayMetrics.density
}
```

The **apply** Function

Purpose: Configure an object

Context object available as: this (implicit)

Return value: The object itself

Use when: Configuring objects, especially during initialization

Basic syntax

```
val configuredObject = someObject.apply {
    // 'this' refers to someObject (implicit)
    // Configure the object
}  // Returns the modified someObject


// Practical example
val textView = TextView(context).apply {
    text = "Hello World"
    textSize = 16f
    setTextColor(Color.BLACK)
    setPadding(16, 16, 16, 16)
}
```

More **apply** Examples

```
// Creating and configuring a dialog
AlertDialog.Builder(context).apply {
    setTitle("Warning")
    setMessage("Are you sure?")
    setPositiveButton("Yes") { dialog, _ -> dialog.dismiss() }
    setNegativeButton("No") { dialog, _ -> dialog.dismiss() }
}.create().show()


// Configuring Intent with extras
Intent(context, DetailActivity::class.java).apply {
    putExtra("USER_ID", userId)
    putExtra("SHOW_DETAILS", true)
    flags = Intent.FLAG_ACTIVITY_NEW_TASK
}.also { context.startActivity(it) }
```

The **also** Function

- Similar to ***apply*** but uses *it* instead of this
- Returns the object itself
- Good for side effects or logging

```
val user = User("John").also {
    Log.d("User", "Created user: ${it.name}”)
}
```

The **with** Function

- Similar to ***run*** but called differently
- Not an extension function
- Good for grouping operations on an object
- Prefer the implicit **this** reference

```
val result = with(user) {
    println("User: $name")
    age * 2
}
```

Comparison table

Function	Object Reference	Return Value	Use Case
let	`it` (explicit)	Lambda result	Executing code on non-null objects; transforming objects
run	`this` (implicit)	Lambda result	Computing a result from an object; executing multiple operations
with	`this` (implicit)	Lambda result	Grouping operations on an object (not an extension function)
apply	`this` (implicit)	Object itself	Configuring an object, especially during initialization
also	`it` (explicit)	Object itself	Additional operations or side effects (logging, validation)