Practical Exercise – Fundamental Concepts of Android Development

Activity Lifecycle, Intents, Layouts, and Views

Course: Mobile Application Development (Chapter 5)
Level: Technical University Students
Duration: 6–8 hours self-study + 1–2 hours online discussion
Format: Individual Assignment with Code Examples (2–3 pages)

Exercise Overview

In this exercise, you will work with core Android components: **Activity lifecycle**, **Intents** (explicit and implicit), **Layout resources**, and **Views**. These fundamentals form the foundation of all Android applications.

Key Learning Outcomes:

- Understand the complete Activity lifecycle and when to implement each callback
- Distinguish between explicit and implicit Intents and their use cases
- Design responsive layouts using ConstraintLayout and other ViewGroups
- Handle user interactions with Views and event listeners
- Manage state and data persistence across Activity transitions

---

Part A: Activity Lifecycle Mastery

Task 1: Activity Lifecycle Mapping (Self-Study: 1.5–2 hours)

Objective: Master the Activity lifecycle states and implement appropriate logic in each callback.

Instructions:

1. **Draw the complete Activity lifecycle** showing all states and transitions:
    - onCreate() → onStart() → onResume() → onPause() → onStop() → onDestroy()
    - Include transitions for: user navigation away, system memory pressure, configuration changes
    - Mark which callbacks are guaranteed vs. optional
2. **Identify three critical moments** where you must add code:

Lifecycle Method	Why Important	Typical Operations
onCreate()	First initialization after process creation	Initialize UI, load data, set up listeners
onResume()	App returns to foreground	Refresh data, resume animations, re-enable sensors
onPause() / **onStop()**	App losing focus or becoming invisible	Save user data, pause animations, release sensors

3. **Write pseudo-code for each critical moment:**

```
// onCreate() - Initialize
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
setContentView(R.layout.activity_main)
loadUserPreferences()
initializeUI()
}

// onResume() - Prepare for interaction
override fun onResume() {
super.onResume()
refreshData()
startLocationUpdates()
}

// onPause() - Save state before leaving
override fun onPause() {
super.onPause()
saveUserInputToPreferences()
stopLocationUpdates()
}
```

4. **Answer in 3–4 sentences:**
    - When is savedInstanceState used and why is it important?
    - What's the difference between onStop() and onDestroy()?

Deliverable:

- Lifecycle diagram with all states and transitions

- Three pseudo-code implementations with brief explanations
- 3–4 sentence reflection on savedInstanceState and lifecycle guarantees

---

Part B: Understanding Intents

Task 2: Explicit vs. Implicit Intents (Self-Study: 1.5–2 hours)

Objective: Distinguish between Intents and write them for common scenarios.

Instructions:

1. **Define the two Intent types:**

Type	When Used	Example
Explicit Intent	Launching a specific Activity in your app	Navigate from LoginActivity → MainActivity
Implicit Intent	Request action without specifying component	Open camera, make phone call, share text

2. **Write code for five Intent scenarios** (provide Kotlin or Java code):

   Scenario A: Explicit Intent – Navigation with Data
   ```
   // From LoginActivity, navigate to MainActivity and pass userId
   val intent = Intent(this, MainActivity::class.java)
   intent.putExtra("userId", "12345")
   intent.putExtra("userName", "Alice")
   startActivity(intent)
   ```

   Scenario B: Explicit Intent – Start Activity for Result
   ```
   // Launch camera app and expect photo back
   val takePictureIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
   startActivityForResult(takePictureIntent, CAMERA_REQUEST_CODE)
   ```

   Scenario C: Implicit Intent – Open Camera
   ```
   val cameraIntent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
   startActivity(cameraIntent)
   ```

   Scenario D: Implicit Intent – Open Web Browser
   ```
   val webIntent = Intent(Intent.ACTION_VIEW,
   Uri.parse("https://www.android.com"))
   startActivity(webIntent)
   ```

   Scenario E: Implicit Intent – Make a Phone Call
   ```
   val callIntent = Intent(Intent.ACTION_CALL, Uri.parse("tel:+1234567890"))
   // Remember: Add android.permission.CALL_PHONE permission
   startActivity(callIntent)
   ```

3. **For each scenario, explain (1–2 sentences):**
    - What information is passed?
    - Why would you use this Intent in a real app?
4. **Data Passing and Retrieval:**
    - Show how to retrieve extras in the receiving Activity
      // In MainActivity - retrieve data
      val userId = intent.getStringExtra("userId")
      val userName = intent.getStringExtra("userName")

Deliverable:

- Five complete Intent code examples
- For each, a 1–2 sentence explanation of use case
- Code showing how to retrieve passed data in receiving Activity

---

Part C: Layouts and Views

Task 3: Responsive UI Design (Self-Study: 1.5–2 hours)

Objective: Design a practical mobile app screen using ConstraintLayout and handle user interactions.

Instructions:

1. **Design a Login Screen Layout** in XML pseudo-code or sketch:

   Requirements:
    - **EditText** for email input (hint: "Enter email")
    - **EditText** for password input (inputType: password)
    - **Button** for login action (full width, prominent color)
    - **TextView** for "Forgot Password?" link (clickable)
    - **ProgressBar** (initially hidden, shown during login)
2. **XML Example Structure:**
   <androidx.constraintlayout.widget.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

```
<ImageView
    android:id="@+id/logo"
    android:layout_width="100dp"
    android:layout_height="100dp"
```

```
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="40dp" />

<EditText
    android:id="@+id/emailInput"
    android:layout_width="0dp"
    android:layout_height="48dp"
    android:inputType="textEmailAddress"
    android:hint="Email"
    app:layout_constraintTop_toBottomOf="@id/logo"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="30dp"
    android:layout_marginStart="16dp"
    android:layout_marginEnd="16dp" />

<EditText
    android:id="@+id/passwordInput"
    android:layout_width="0dp"
    android:layout_height="48dp"
    android:inputType="textPassword"
    android:hint="Password"
    app:layout_constraintTop_toBottomOf="@id/emailInput"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_marginTop="16dp"
    android:layout_marginStart="16dp"
    android:layout_marginEnd="16dp" />

<Button
    android:id="@+id/loginButton"
    android:layout_width="0dp"
    android:layout_height="48dp"
    android:text="Login"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    android:layout_margin="16dp" />

<TextView
    android:id="@+id/forgotPassword"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Forgot Password?"
    android:textColor="@color/blue"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/passwordInput"
    android:layout_marginStart="16dp"
```

```
    android:layout_marginTop="12dp" />

<ProgressBar
    android:id="@+id/loadingSpinner"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:visibility="gone"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintTop_toBottomOf="@id/loginButton" />
```

</androidx.constraintlayout.widget.ConstraintLayout>

3. **Implement Click Listeners and State Management:**

class LoginActivity : AppCompatActivity() {
private lateinit var emailInput: EditText
private lateinit var passwordInput: EditText
private lateinit var loginButton: Button
private lateinit var loadingSpinner: ProgressBar
private lateinit var forgotPassword: TextView

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

    // Initialize views
    emailInput = findViewById(R.id.emailInput)
    passwordInput = findViewById(R.id.passwordInput)
    loginButton = findViewById(R.id.loginButton)
    loadingSpinner = findViewById(R.id.loadingSpinner)
    forgotPassword = findViewById(R.id.forgotPassword)

    // Set click listeners
    loginButton.setOnClickListener {
        val email = emailInput.text.toString()
        val password = passwordInput.text.toString()

        if (validateInput(email, password)) {
            showLoading(true)
            performLogin(email, password)
        } else {
            showError("Please enter valid email and password")
        }
    }

    forgotPassword.setOnClickListener {
        startActivity(Intent(this,
ForgotPasswordActivity::class.java))
    }
}
```

```
private fun validateInput(email: String, password: String):
Boolean {
    return email.isNotEmpty() && password.isNotEmpty() &&
email.contains("@")
}

private fun performLogin(email: String, password: String) {
    // Simulate network call
    // In real app: call API, handle response
    Handler(Looper.getMainLooper()).postDelayed({
        showLoading(false)
        startActivity(Intent(this, MainActivity::class.java))
        finish()
    }, 2000)
}

private fun showLoading(isLoading: Boolean) {
    loadingSpinner.visibility = if (isLoading) View.VISIBLE else
View.GONE
    loginButton.isEnabled = !isLoading
}

private fun showError(message: String) {
    Toast.makeText(this, message, Toast.LENGTH_SHORT).show()
}
```

```
}
```

4. **Layout Design Questions** (answer in 2–3 sentences each):
    - **Why use ConstraintLayout over LinearLayout here?**
      ConstraintLayout is more flexible and requires fewer nested layouts, improving performance and allowing complex responsive designs.
    - **How would this login screen adapt on a tablet vs. phone?**
      You could use resource qualifiers (layout-large, layout-xlarge) to create wider layouts with centered content for tablets.
    - **What's one accessibility improvement you'd make?**
      Add content descriptions to ImageView, ensure button text is descriptive, and use proper contrast ratios for text colors.

Deliverable:

- Login screen layout (XML sketch or pseudocode)
- Click listener code implementation
- 2–3 sentence answers to layout design questions

Part D: Reflection & Best Practices

Task 4: Android Development Principles (Online: 0.5–1 hour)

Objective: Develop strategic thinking about Android component design.

Instructions:

Answer **all three questions** (200–250 words total):

1. **Lifecycle Importance:**
    - Why does Android separate Activities into distinct lifecycle states rather than having a simple "on/off" model?
    - What problems would arise if Activities didn't have onPause() and onStop()?
2. **Intent Design Philosophy:**
    - Why does Android encourage implicit Intents for system actions (camera, phone, share) instead of just providing built-in APIs?
    - What's one advantage and one disadvantage of this approach?
3. **Layout Performance:**
    - You have two options: a LinearLayout with 10 nested views, or a ConstraintLayout with the same structure. Which is better and why?
    - What metric would you use to measure layout performance?

Deliverable: Answers to all 3 questions, 200–250 words total.

Evaluation Criteria

Criteria	Excellent (9–10)	Good (7–8)	Acceptable (5–6)	Needs Improvement (<5)
Lifecycle Understanding	Complete lifecycle mapped with all transitions; clear callbacks	Lifecycle mostly correct; minor omissions	Basic lifecycle shown; some confusion	Incomplete or incorrect lifecycle
Intent Implementation	All 5 scenarios correct; good data passing explanations	4–5 scenarios correct; mostly accurate	3–4 scenarios; some errors	Fewer than 3 scenarios or major errors

Layout Design	Responsive, accessible, well-structured XML; good constraints	Good layout design; mostly responsive	Functional layout; accessibility gaps	Poor layout design or non-responsive
Code Quality	Clean Kotlin/Java; follows Android best practices	Generally clean; minor style issues	Readable but inconsistent	Difficult to read or unclear
Strategic Thinking	Thoughtful reflection on Android design philosophy	Good reflection; understands rationale	Basic reflection; limited depth	Minimal or missing reflection
Communication	Clear explanations; well-organized code examples	Generally clear; good code comments	Readable but lacks detail	Disorganized or unclear

---

Deliverable Checklist

Submit as **PDF or DOCX (2–3 pages)** containing:

- [ ] **Task 1:** Activity lifecycle diagram + three pseudo-code implementations + reflection
- [ ] **Task 2:** Five Intent code examples + explanations + data retrieval code
- [ ] **Task 3:** Login screen layout (XML or sketch) + click listener implementation + design questions
- [ ] **Task 4:** Strategic reflection (3 questions answered, 200–250 words total)

Resources & References

Official Android Documentation:

- Android Activity Lifecycle
- Intents and Intent Filters
- Layouts
- ConstraintLayout

Best Practices:

- Google Android Architecture Components: https://developer.android.com/jetpack/guide

- Android Developer Blog: https://android-developers.googleblog.com/

Design Tools:

- Android Studio Layout Editor (drag-and-drop UI design)
- Figma (prototyping layouts)

Timeline Guidance

Self-Study (6–8 hours):

- Hours 1–2: Task 1 (Activity lifecycle mastery)
- Hours 2–4: Task 2 (Intent scenarios and data passing)
- Hours 4–7: Task 3 (Layout design and click listeners)
- Hours 7–8: Task 4 (Reflection and best practices)

Online Sessions (1–2 hours):

- 0.5 hours: Task 1 discussion (lifecycle callbacks, savedInstanceState)
- 0.5 hours: Task 2 demo (Intents in practice, common pitfalls)
- 0.5 hours: Task 3 layout review (responsive design, accessibility)
- 0.5 hour: Task 4 reflection & Q&A (Android philosophy, design patterns)

Extension Tasks (Optional, Advanced)

1. **Process Death Simulation:**
    - Implement logic to handle app process termination and restoration
    - Use onSaveInstanceState() and bundle persistence
2. **Advanced Intent Filtering:**
    - Create custom Intent actions and implement implicit Intent receivers
    - Write intent filters in AndroidManifest.xml
3. **Custom Views:**
    - Design a custom View subclass (e.g., circular progress indicator)
    - Implement onDraw() and handle touch events
4. **Responsive Layout Challenge:**
    - Design a layout that works on phones (small), tablets (medium), and large displays
    - Use configuration qualifiers and ConstraintLayout
5. **Lifecycle Testing:**

    - Write unit tests for Activity lifecycle callbacks using AndroidX testing libraries
    - Verify savedInstanceState logic with Espresso tests

Notes for Instructors

This exercise covers **foundational Android concepts** that appear in every mobile app. Students develop:

- Understanding of Activity lifecycle and state management
- Practical Intent usage for navigation and system integration
- Responsive UI design skills using modern layout frameworks
- Debugging and testing strategies for Android components

Discussion prompts for online session:

- "Why does Android destroy and recreate Activities on configuration changes?"
- "When would you use Explicit Intents vs. Implicit Intents?"
- "How do you optimize layout performance for large view hierarchies?"
- "What's the best way to handle long-running operations in Activities?"
- "How does the Activity lifecycle relate to memory management?"

Common student mistakes:

- Assuming Activities persist across app restarts without proper serialization
- Forgetting to add permissions for implicit Intents (camera, call, location)
- Creating deeply nested layouts that impact performance
- Not handling lifecycle callbacks properly (e.g., leaking resources in onResume())
- Ignoring accessibility in UI design (missing content descriptions, poor contrast)

Assessment Notes

Grading rubric integration:

- Emphasizes *understanding* over perfect syntax
- Values correct lifecycle management and state handling
- Recognizes proper Intent usage patterns
- Encourages responsive, accessible design

Partial credit guidance:

- Lifecycle: 25% (correct callbacks and transitions)
- Intents: 30% (appropriate usage, data passing)

- Layouts: 30% (responsive design, accessibility)
- Strategic thinking: 15% (reflection on Android philosophy)