✅ Week 1 Summary – Introduction to Android & Android Studio

📱 What is Android?

- Android is an open-source mobile operating system developed by Google.
- It runs on smartphones, tablets, TVs, wearables, etc.

🛠️ What is Android Studio?

- **Android Studio** is the official IDE for Android app development.
- It provides tools to write code, design UI, test apps, and debug.

🔤 Programming Languages:

- Android apps are mainly built using **Java** or **Kotlin**.

🧩 Key Components of an Android App:

1. **Activities** – Screens the user interacts with.
2. **Fragments** – Reusable parts of UI within Activities.
3. **Services** – Background tasks with no UI.
4. **Broadcast Receivers** – Listen for system-wide events.
5. **Content Providers** – Share data between apps.

📁 Project Structure (Simplified):

- MainActivity.java or MainActivity.kt: Code for main screen.
- activity_main.xml: Layout of the screen (UI design).
- AndroidManifest.xml: App settings like permissions and registered components.

💡 Easy Analogy:

Think of an Android app like a house:

- **Activity** = Room
- **Fragment** = Furniture

- **Intent** = Door connecting rooms
- **Android Studio** = Your toolbox and blueprint workspace

✅ **Week 1 Sample Questions:**

1. **What is the role of Android Studio?**
   → It helps you build, test, and run Android apps.
2. **What is an Activity?**
   → A screen the user interacts with.
3. **What language can you use in Android development?**
   → Java or Kotlin.

Week 2 Summary – "Hello World for Android"

1. Project Setup

- Create a project using the **Empty Activity** template.
- Target **API 34: Android 14**.
- Use the default emulator configured for Android 14.
- The project includes:
  - MainActivity.java
  - activity_main.xml
  - Manifest file
  - Gradle files

2. MainActivity

- Entry point of the app is onCreate(Bundle savedInstanceState).
- Similar to JavaFX's start() method.
- Default code includes:

java

CopyEdit

EdgeToEdge.enable(this);

ViewCompat.setOnApplyWindowInsetsListener(...)

These lines are unnecessary for now and can be removed for simplicity.

3. Android vs. JavaFX

- Android apps are **event-driven** like JavaFX.
- No main() method.
- Apps are composed of **Activities** (screens).
- Use **setContentView()** to define UI layout.

4. Layout & XML

- UIs are defined using XML in res/layout/activity_main.xml.
- The default layout is **ConstraintLayout**.
    - Allows flexible positioning relative to parent or other widgets.
- Views (like TextView) can be edited in XML or visually in “Design” or “Split” view.

5. Device Rotation

- Android supports rotation.
- Rotating triggers onCreate() again, and UI gets redrawn.
- No need to manually reposition widgets due to flexible layout.

6. Manifest File

- AndroidManifest.xml contains metadata:
    - Package name
    - Declared activities (<activity android:name=".MainActivity" />)
    - Permissions (if any)
    - Theme and icons

7. Themes and Styles

- Defined in res/values/themes.xml and colors.xml.
- Themes apply styles globally (e.g., background color, status bar).

8. Gradle Files

- Used to manage builds.
- Define:
    - compileSdk, minSdk
    - Dependencies
- Normally handled automatically, but you may need to adjust versions.

9. Android Studio UI

- Many icons in the toolbar:
    - ▶️ Run app
    - 🐞 Debug
    - 🔨 Build only
    - 📱 Device manager
    - 🤖 Gemini (AI assistant)
- Hover for tooltip explanations.
- Use “hamburger” menu for more readable options.

10. Gemini Assistant

- Built-in AI assistant in Android Studio.
- Can generate Kotlin code by default.
- This course uses **Java**, not **Kotlin**.
- Gemini code might not always match course standards — use with caution.

❓ Q&A – Week 2

Q1: What is the role of onCreate() in Android apps?

A: It's called when the activity starts. Similar to JavaFX's start() method. It initializes the app UI using setContentView().

Q2: How is Android layout defined?

A: In XML under res/layout. The activity_main.xml defines the visual layout using widgets like TextView.

Q3: What is the purpose of setContentView()?

A: It sets the layout for the activity by referring to the XML layout file (e.g., R.layout.activity_main).

Q4: What happens when you rotate the device?

A: The onCreate() method is called again, and the layout is redrawn based on the XML file.

Q5: Why should we remove EdgeToEdge.enable(this) in early examples?

A: It adds complexity that's unnecessary for beginners. It deals with edge insets, which aren't important yet.

Q6: What's in the AndroidManifest.xml file?

A: Metadata about the app like package name, components (activities), permissions, theme, and icon.

Q7: What is a theme in Android?

A: A style applied globally to all views in the app. It defines colors, fonts, background, etc., and is stored in themes.xml.

Q8: What does Gradle do?

A: It manages the build process, SDK versions, and dependencies for the app.

Q9: Why shouldn't we rely too much on Gemini?

A: It generates Kotlin code by default and may introduce complexity or errors outside of course expectations.

Q10: What is an Activity?

A: A single screen in an Android app. MainActivity is usually the first one launched.

✅ **Week 3 Summary: Layout Editor – Welcome Name App**

- **Create a new Android project** using the **Empty Activity** template targeting **API 34 (Android 14)**.
- Open activity_main.xml in **Split View** to see both **Design** and **XML**.
- Use **Blueprint mode** and **"Show All Constraints"** to manage view alignment.
- Widgets used:
    - EditText – for user input (badly named "Plain Text")
    - Button – labeled "Say Hello"
    - TextView – displays "Welcome <name>"
- All widgets are added in the Layout Editor and constrained to **parent or other widgets**.
- Modify **widget attributes** like text, id, and onClick using the **Attributes panel**.
- Use **findViewById()** in Java to reference UI elements and update them.
- EditText.getText().toString() is used to retrieve input.
- Create a method like showWelcomeMsg(View v) and link it to the **onClick** attribute in XML.
- Update the **theme** to include the action bar by removing .NoActionBar in themes.xml.

- Export projects via **File → Export to Zip File...** and upload to **Canvas**.

❓ Q&A

Q1: What is the purpose of the Plain Text widget?
A: It's actually an EditText widget used to get user input.

Q2: What method gets input from an EditText?
A: getText().toString().

Q3: How do you update a TextView with code?
A:

java

CopyEdit

TextView tv = findViewById(R.id.textView);

tv.setText("Welcome John");

Q4: What's the correct format for a button click handler?
A:

java

CopyEdit

public void showWelcomeMsg(View view) { ... }

Q5: How do you link the button to the click handler?
A: Set the onClick attribute in XML:

xml

CopyEdit

android:onClick="showWelcomeMsg"

Q6: What is the base class for all UI components?
A: View.

Q7: What's the correct way to export your project for submission?
A: Use **Export to Zip File** in Android Studio under the File menu.

Activities in Android

- An **Activity** is a core component of an Android app, representing a single screen with a user interface.
- Most apps have **multiple activities** (screens), with **one main activity** that launches first.
- Only **one activity** is active (in focus) at a time.

🔹 **Activity Lifecycle**

- Activities go through different **states**: created, started, resumed (active), paused, stopped, destroyed.
- Key **lifecycle callback methods**:
    - onCreate() – initialize the activity.
    - onStart() – make the activity visible.
    - onResume() – activity is active and in focus.
    - onPause() – activity is partially visible (another one is in focus).
    - onStop() – activity is no longer visible.
    - onRestart() – preparing to start again.
    - onDestroy() – clean-up before the activity is destroyed.

🔹 **Logging the Lifecycle**

- Use Log.d("==LIFECYCLE==", "methodName()") to trace the lifecycle in **Logcat**.
- You can simulate lifecycle changes by rotating the phone or switching activities.

🔹 **Multiple Activities**

- Use Intent to switch between activities:

java

CopyEdit

Intent intent = new Intent(MainActivity.this, MainActivity2.class);

startActivity(intent);

- Add nextActivity(View view) method in Java and set the **Button** onClick attribute.

🔹 **Layouts & ConstraintLayout**

- Use **ConstraintLayout** for flexible UI.
- Add **EditText** with hint and **Button** with alignment constraints.
- Add a **TextView** in second activity's layout (activity_main2.xml).

🔹 **Activity Stack**

- Android uses a **LIFO (Last In First Out)** stack to manage activities.
- Pressing the **Back** button pops the current activity off the stack and resumes the previous one.
- Each activity should have its own lifecycle logging (==LIFECYCLE Activity2== for second activity).

✅ **Key Methods to Remember**

- onCreate() – when activity starts

- onPause() – when switching to another activity
- startActivity() – for moving between activities
- Log.d() – to monitor lifecycle transitions in Logcat

WEEK4

Short Summary

1. Information Hiding

- A principle of OOP to prevent direct access to an object's data.
- Helps prevent *privacy leaks* where one class can change another's internal data.

2. Static Getter Method for Data Sharing

- You can use a static variable and getter method to share data between activities.
- Not ideal—can cause memory leaks, tight coupling, and crashes on low-memory devices.

3. Recommended: Using Intent.putExtra()

- A safer, Android-recommended way to send data between activities.
- Add key-value pairs with putExtra("key", value) in Activity 1.
- Use getIntent().getStringExtra("key") to retrieve in Activity 2.

4. Layout Techniques

- Use a **Guideline** to anchor UI elements more flexibly.
- Create a **horizontal chain** of buttons (B1, B2, B3) for a responsive layout.

5. Reusing onClick(View)

- Instead of writing separate methods for each button, use View view to determine which was pressed.
- view.getId() gives the button ID, and ((Button)view).getText() gives its label.

6. Setting OnClick Programmatically

- Implement View.OnClickListener in your class.
- Register listeners in onCreate() using setOnClickListener(this).

❓ Q&A

Q1: Why is using a static getter to share data between activities risky?
A: It creates tight coupling and risks privacy leaks. Also, Android may destroy activities due to low memory, causing crashes if static references are accessed afterward.

Q2: What is the better alternative to static getters for sharing data?
A: Use Intent.putExtra() to pass data to the next activity and retrieve it using getIntent().getStringExtra().

Q3: What does this line do?

java

CopyEdit

switch2Activity2.putExtra("loginText", loginText);

A: It adds a string value (loginText) to the Intent object with the key "loginText" for the next activity to access.

Q4: How do you access the value sent via Intent in the second activity?
A:

java

CopyEdit

Intent intent = getIntent();

String incomingText = intent.getStringExtra("loginText");

Q5: What is a privacy leak in Android?
A: When a class exposes internal data (e.g., via a getter that returns a reference), allowing other classes to modify it directly.

Q6: How can you make a layout responsive to screen rotation using buttons?
A: Use a horizontal chain of buttons linked to a **Guideline**, so they adjust positions and spacing automatically.

Q7: What does implements View.OnClickListener do?
A: Allows your activity class to handle click events in a centralized onClick(View view) method for multiple buttons.

Q8: How can you determine which button triggered the shared onClick(View view)?
A: Use:

java

CopyEdit

String buttonName = ((Button)view).getText().toString();

int buttonId = view.getId();

SHORT SUMMARY

Objective: Learn how to start a second activity and get a result back in a modern, safe way using ActivityResultLauncher.

🔁 **Passing Data to an Activity**

- Use Intent with putExtra() to send data.
- Access it in the second activity using getIntent().getStringExtra().

🔄 **Getting Data Back from Second Activity**

- Use **ActivityResultLauncher<Intent>** and register it using registerForActivityResult(...).
- On the second activity, use setResult(RESULT_OK, intent) and call finish() to return data.
- Back button or RESULT_CANCELED can indicate cancellation.

📲 Modern API

- Replaces deprecated startActivityForResult() and onActivityResult().
- ActivityResultContracts.StartActivityForResult() is the contract used.
- The callback (e.g., handleResult(ActivityResult result)) is always registered in onCreate().

🛠️ MainActivity Workflow

1. User enters text → presses "Launch Second Activity"
2. Text is passed to MainActivity2
3. User presses “Close” in MainActivity2
4. Result (e.g., number 42) is passed back and logged in handleResult

---

❓ Q&A

Q1: How do you pass data to another activity?
A1: Use Intent intent = new Intent(this, SecondActivity.class);
intent.putExtra(Intent.EXTRA_TEXT, "yourData");

Q2: How do you return data from the second activity?
A2: Create an Intent, add data using putExtra(), call setResult(RESULT_OK, intent), then finish().

Q3: What is ActivityResultLauncher?
A3: It's a modern way to launch an activity and receive a result using a registered callback.

Q4: Why not use startActivityForResult()?
A4: It's deprecated and not recommended. Use ActivityResultLauncher for safety and compatibility.

Q5: What happens if the second activity is closed with the back button?
A5: The result is RESULT_CANCELED.

Q6: Where do you register ActivityResultLauncher?
A6: In the onCreate() method using registerForActivityResult(...).

Q7: How do you handle results in MainActivity?
A7: Use a method like handleResult(ActivityResult result) which is called when the second activity returns.

Q8: What is the use of Intent.EXTRA_TEXT?
A8: It's a predefined key used to pass simple text data through intents.

WEEK 5

Short Summary

This week focuses on:

🎯 **Using Strings and Spinners in Android**

- Strings are stored in res/values/strings.xml.
- Access via:
    - **Java:** R.string.name

    - **XML:** @string/name
- Spinners allow users to choose from a dropdown list (e.g., weekdays).
- Items can be added using <string-array> in strings.xml.

🛠️ GUI Layout

- Layout includes:
    - EditText (user input)
    - Spinner (dropdown)
    - Button (to save input)
- Layout is centered in the top half of the screen.

⚙️ Spinner Setup

- Use AdapterView.OnItemSelectedListener to listen for item changes:
    - onItemSelected() logs the selected item.
    - onNothingSelected() logs when no selection is made.
- Set default selection and listener in onCreate().

💾 Saving State with SharedPreferences

- Used to persist app state across launches.
- Example data saved:
    - Text from EditText
    - Selected index from Spinner
- Save on button click using an Editor object and apply().

📥 Loading State

- Load saved values on first app launch.
- Uses flags like firstRun to ensure preferences are only loaded once.

📂 Android File System

- SharedPreferences saved in:

bash

CopyEdit

/data/data/<package_name>/shared_prefs/saved_configuration.xml

- Emulator can be wiped/reset via Device Manager if needed.

❓ Q&A

Q1: What is the purpose of strings.xml?

A: To manage all app strings in one place, allowing easier localization and organization.

Q2: How do you link a string array to a Spinner?

A: Define the array in strings.xml, then set it using android:entries="@array/your_array" in the Spinner XML.

Q3: How do you handle Spinner item selection?

A: Implement AdapterView.OnItemSelectedListener and override:

- onItemSelected()
- onNothingSelected()

Q4: What is SharedPreferences used for?

A: To save key-value pairs (like text input or selected spinner item) that persist between app launches.

Q5: What happens in the saveButton() method?

A: It:

- Validates input.
- Gets the SharedPreferences editor.

- Stores the input and spinner position.
- Applies the changes.

Q6: When should you load shared preferences?

A: During the first app launch (in onCreate()), using a boolean flag like firstRun.

Q7: How do you reset a broken Android emulator?

A: Power off the emulator, go to Device Manager > 3-dots menu > "Wipe Data", then "Cold Boot Now".

SHORT SUMMARY: Coding Conventions (COMP 10073)

Goal: Write clean, maintainable, and professional Android code.

📌 **Key Points:**

- **Readable Code:** Avoid overly complex or messy syntax. Structure code clearly.
- **Assignments vs Labs:** Labs focus on working code; assignments require **well-documented, readable** code.
- **Commenting:**
    - Avoid excessive line-by-line comments.
    - Prefer **JavaDoc** (/** ... */) with @param and @return tags.
- **Dead Code & TODOs:** Don't submit assignments with TODOs or commented-out unfinished code.
- **Exception Handling:** Always handle exceptions properly—never leave catch (Exception e) {} empty.
- **Logging:** Use Log.d() for debugging with proper tags and messages.
- **Avoid Hardcoded Strings:** Use final static String for keys (e.g., Intent extras) to prevent runtime errors.

- **Handlers:** Use setOnClickListener(this::methodName) for clarity, avoid deep nested anonymous classes.
- **Good Practices:**
    - Functions should be **short (25–50 lines)** and **do one thing well**.
    - Max 3 levels of indentation.
    - 5–10 local variables per method is ideal.

🔗 **Helpful Links:**

- Google Java Style Guide
- Oracle JavaDoc Tool
- Android Code Style

❓ **Q&A (Quick Review)**

Q1: Why is excessive commenting discouraged in production code?
A: It clutters the code, makes it longer, and requires extra effort to keep comments updated with changes.

Q2: What type of comment is preferred for professional Java code?
A: JavaDoc comments with @param, @return, etc.

Q3: What should you use instead of hardcoding strings like "loginText" in Intents?
A: Define a constant like:

java

CopyEdit

public final static String loginKey = "login";

Q4: What's wrong with this exception block?

java

CopyEdit

```
try {
   Float.valueOf(new_text);
} catch (Exception e) {}
```

A: The catch block is empty. It hides errors instead of handling them properly.

Q5: How should button handlers be written for readability?
A: Use named methods like this::logSomething instead of anonymous inner classes.

Q6: How many lines should a function ideally have?
A: 25–50 lines maximum; fewer if the logic is complex.

Q7: What's the problem with repeating code for each button handler?
A: Repetition makes code harder to maintain. Use a shared method for buttons with similar logic.

Q8: Why should logging be used during development?
A: To track and debug app behavior using Log.d(tag, message).

WEEK 6

Summary: Handling Rotation in Android App Development

Overview: In Android development, handling rotation is essential for ensuring that the app works smoothly across both portrait and landscape orientations. Key aspects include managing layout changes and detecting orientation using system tools like Display and WindowMetrics.

1. **Anchoring Widgets:**

- **Anchoring widgets to different parts of the layout (e.g., top or bottom) can affect how the layout changes when the device rotates.**
- **It is important to test layouts in both orientations to avoid overlap or hidden widgets.**

2. **Detecting Device Orientation:**
   - **Use the Display object to detect rotation and the WindowMetrics object to get screen dimensions. The rotation can be checked using getRotation() and dimensions using getBounds().**
3. **Handling Rotation:**
   - **Android's activity lifecycle triggers events like onCreate(), onResume(), and onPause() when the orientation changes.**
   - **You can update the UI (e.g., showing the orientation and screen size) by using these methods.**
4. **Layouts for Different Orientations:**
   - **Android allows creating separate XML layout files for portrait and landscape orientations using qualifiers like layout-land for landscape.**
   - **This approach lets you rearrange widgets to better fit the screen in each orientation.**
5. **Recognizing Rotation:**
   - **Detect if the device was just rotated by comparing the current orientation with the last known orientation.**
   - **You can use a static variable (lastRotation) to track changes and update the UI accordingly.**

Q&A

1. **How does anchoring widgets affect rotation handling?**
   - **Anchoring widgets at different positions in the layout impacts how they move when the device rotates. For example, anchoring a TextView to the top and another to the bottom causes their positions to change upon rotation, which can lead to layout issues if not properly tested.**

2. **What tool is used to detect device rotation in Android?**
    - **The Display object is used to detect rotation, with the getRotation() method returning the rotation state (e.g., ROTATION_0, ROTATION_90, etc.).**
3. **How can you manage different layouts for portrait and landscape modes?**
    - **By creating separate XML files for each orientation. For instance, a activity_main.xml layout file is used for portrait mode, and a activity_main.xml (land) file is created for landscape mode. This allows customizing the layout for each orientation.**
4. **What lifecycle methods are important when handling rotation?**
    - **The key lifecycle methods are onCreate(), onResume(), and onPause(). onCreate() is triggered when the layout needs to be redrawn, onResume() happens just before an activity becomes visible, and onPause() occurs when the activity goes into the background.**
5. **How can you recognize a rotation event and update the UI accordingly?**
    - **You can track the rotation state using a static variable (lastRotation) and compare it with the current rotation. If the rotation has changed, you can update the UI to reflect the new orientation.**

Summary of Toasts, Snackbars, and Dialogs in Android

1. **Toasts:**
    - **A Toast is a brief, non-intrusive popup message that shows over the app content.**
    - **They are used for showing transient messages that do not need user interaction.**
    - **You can use Toast.makeText(context, message, duration).show() to display a Toast.**
    - **There are two types of durations: Toast.LENGTH_SHORT and Toast.LENGTH_LONG.**

- **Toasts are basic and don’t support actions or interactions directly.**

2. **Snackbars:**
    - **Snackbars are similar to Toasts but offer more features like action buttons.**
    - **They provide lightweight feedback and can show a message at the bottom of the screen.**
    - **Snackbars can be dismissed either automatically or through user interaction.**
    - **A Snackbar can be customized by adding buttons, changing colors, and even using custom layouts.**
    - **Snackbars support the duration Snackbar.LENGTH_INDEFINITE to keep the message visible until the user interacts with it.**
3. **Dialogs:**
    - **Dialogs are full-screen popups that require user interaction to proceed.**
    - **You can use AlertDialog.Builder to create simple dialogs with options like Yes/No or custom lists.**
    - **Dialogs block user interaction with the app until an action is taken.**
    - **They are useful for confirming actions, displaying messages, or showing complex forms.**

Q&A:

1. **What is a Toast in Android?**
    - **A Toast is a small, non-interactive message that appears briefly on the screen to provide feedback to the user.**
2. **How do you create a Toast in Android?**
    - **Use the method Toast.makeText(context, message, duration).show() to display a Toast message.**
    - **Example: Toast.makeText(this, "Hello, World!", Toast.LENGTH_SHORT).show();**

3. **What is the difference between a Toast and a Snackbar?**
    - **Toasts are simple messages that appear briefly and disappear without user interaction.**
    - **Snackbars are similar but allow for user interaction through actions like buttons and can stay visible for a longer period.**
4. **How do you create a Snackbar in Android?**
    - **Use Snackbar.make(view, message, duration) to create a Snackbar and show it with snackBar.show().**
    - **Example:**

java

CopyEdit

```
Snackbar.make(view, "This is a Snackbar", Snackbar.LENGTH_LONG).show();
```

5. **What is the purpose of LENGTH_INDEFINITE in a Snackbar?**
    - **LENGTH_INDEFINITE makes the Snackbar stay visible until the user interacts with it, like clicking a button or dismissing it manually.**
6. **How do you add a custom layout to a Snackbar?**
    - **You can inflate a custom layout using LayoutInflater and add it to the Snackbar's view.**
    - **Example:**

java

CopyEdit

```
View customSnackView = inflater.inflate(R.layout.custom_snackbar, null);
snackbarLayout.addView(customSnackView, 0);
```

7. **How do you create a Dialog in Android?**
    - **Use AlertDialog.Builder to create a simple dialog with buttons or messages.**
    - **Example:**

java

CopyEdit

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);

builder.setMessage("Do you prefer Dialogs?");

builder.setPositiveButton("I like Dialogs", (dialog, id) -> { /* handle click */ });

builder.setNegativeButton("I like SnackBars", (dialog, id) -> { /* handle click */ });

builder.setCancelable(false);

builder.create().show();
```

WEEK 7

Summary: Broadcast Senders and Receivers in Android

In Android, Broadcast Senders and Receivers allow apps to send and receive broadcast messages from the Android system or other apps. This is similar to the publish-subscribe design pattern, where the sender publishes a message without knowing who will receive it. Examples include notifications when the device starts charging or when an SMS is received.

Broadcast Receiver Setup:

- **Manifest Declaration: A Broadcast Receiver needs to be declared in the AndroidManifest.xml file.**
- **Permissions: For receiving SMS, you must declare the android.permission.RECEIVE_SMS permission.**
- **Receiver Class: A BroadcastReceiver class, such as MyReceiver, is needed to handle received broadcasts.**

Key Concepts:

- **Broadcast Receiver: A class that listens for specific broadcasts and triggers actions when an event occurs (e.g., new SMS received).**

- **Permissions: Apps must request runtime permissions for receiving SMS in Android 6.0 (SDK 23) and higher.**
- **Intent Flags: Flags like FLAG_ACTIVITY_NEW_TASK and FLAG_ACTIVITY_CLEAR_TOP help control the app's behavior when it's restarted by a broadcast.**

Q&A:

1. **What is a Broadcast Receiver in Android? A Broadcast Receiver listens for specific broadcast messages sent by the system or other apps and executes a defined action upon receiving the message.**
2. **How do you declare a Broadcast Receiver in Android? You declare it in the AndroidManifest.xml file, using a <receiver> element with an <intent-filter> for the specific event you want to listen to.**
3. **What permissions are needed for receiving SMS in Android? You need to declare the android.permission.RECEIVE_SMS permission in the manifest. For Android 6.0 and higher, you must also request permission at runtime.**
4. **What is the purpose of FLAG_ACTIVITY_NEW_TASK? This flag ensures that a new task is created when launching an activity, meaning it will start as the topmost activity in the stack.**
5. **What happens if you omit FLAG_ACTIVITY_CLEAR_TOP? If omitted, launching the activity might result in multiple instances being created on top of each other, which can lead to unexpected behavior when navigating back.**
6. **How do you handle permissions for SMS in the app? You use ActivityCompat.requestPermissions() to request runtime permissions, and handle user responses to whether they allow or deny the permissions.**

Summary: Issues with Broadcasts and Permissions in Android

1. **Broadcast System Issues:**
    - **Older versions of Android (up to 5.0) allowed apps to respond to broadcasts (like SMS) even when not running, leading to high memory usage, background processes, and battery drain.**

- **Android 8.0 and beyond have restricted this behavior to prevent performance issues.**
- **Implicit broadcasts (those not targeting a specific app) can trigger many apps, consuming CPU cycles and causing lag. Examples include frequent broadcasts for activity changes, where multiple apps wake up at once, resulting in sluggish performance.**

2. **Types of Permissions:**
   - **Normal Permissions: Allow access to non-sensitive data (e.g., accessing the internet, battery level).**
   - **Runtime Permissions (Dangerous Permissions): Allow access to sensitive data (e.g., contacts, location, SMS) and require user consent before access.**
3. **Location Manager:**
   - **The LocationManager class provides access to system location services, allowing apps to get periodic updates on the device's location.**
   - **To use location services, you need to request permissions (ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION) in the app's manifest and at runtime.**
4. **Requesting Permissions:**
   - **Before accessing location services, check if the app already has the necessary permissions using ContextCompat.checkSelfPermission().**
   - **If not, request permissions at runtime using ActivityCompat.requestPermissions().**
   - **Permissions are granted or denied asynchronously, so use a callback (onRequestPermissionsResult()) to handle the result.**

Q&A:

1. **Why did Android restrict broadcasts starting in version 8.0?**
   - **Android restricted broadcasts to improve performance and prevent excessive battery usage, memory thrashing, and CPU consumption caused by multiple apps waking up in response to frequent implicit broadcasts.**

2. **What types of permissions are there in Android?**
    - **Normal permissions: These permissions pose minimal risk to privacy or security (e.g., accessing the internet).**
    - **Runtime permissions: These permissions grant access to sensitive data or actions that could impact user privacy or system performance (e.g., accessing the camera, location).**
3. **What is the LocationManager used for in Android?**
    - **The LocationManager provides access to the system's location services, enabling apps to obtain location updates or detect proximity to specific geographic areas.**
4. **How do you request runtime permissions in Android?**
    - **Use ActivityCompat.requestPermissions() to request runtime permissions. Before doing this, check if the app already has the required permissions using ContextCompat.checkSelfPermission().**
5. **How does Android handle permission requests asynchronously?**
    - **Permission requests are handled asynchronously, meaning that the app will continue executing while the permission dialog is displayed. Use the onRequestPermissionsResult() callback to handle the user's decision.**

WEEK8

Summary:

In this week's Android App Development lesson (Week 8), the focus is on building a simple SMS messaging application that allows users to send and receive text messages. Key concepts include handling permissions for sending and receiving SMS, managing activity state, and preventing data loss during background activity transitions.

- **Main Components:**

- **Activity Layout (activity_main.xml): Includes an EditText for typing the message, a Button for sending the message, and a TextView for displaying received messages.**
- **Permissions: The app requests runtime permissions for sending and receiving SMS messages.**
- **SMS Sending and Receiving:**
  - **MainActivity.java handles the UI and sending messages.**
  - **MyReceiver.java listens for incoming SMS and sends the message to MainActivity for display.**
- **Activity State: Ensures the message being typed is not lost when the activity is paused or resumed, using onPause() and TextWatcher.**
- **Message Reception: Instead of creating a new activity every time an SMS is received, MainActivity can directly update the TextView with the received message.**

- **Solutions:**
  - **Use onPause() to store the message in editMsg before the activity is paused.**
  - **Use TextWatcher to track changes in the EditText.**
  - **Implement a custom method to update the TextView without starting a new activity when a message is received.**

Q&A:

Q1: What permissions are needed for this app to send and receive SMS messages?

- **The app needs the SEND_SMS and RECEIVE_SMS permissions, which should be declared in the AndroidManifest.xml file.**

Q2: How do we manage the data loss when an activity is paused or resumed?

- **By saving the message in a static variable (editMsg) in onPause() and restoring it in onCreate(), you can prevent the message from being cleared when the app is paused.**

Q3: Why should we not start a new activity every time an SMS is received?

- **Starting a new activity for every SMS message would consume more memory and resources. Instead, it's better to update the existing activity's TextView with the new message.**

Q4: How can we track changes in the EditText without losing the input?

- **You can use a TextWatcher to monitor changes in the EditText. The afterTextChanged() method captures the updated text and saves it in editMsg.**

Q5: How can you restrict certain keys in the EditText (e.g., prevent vowels)?

- **You can use an OnKeyListener to monitor key presses and prevent specific keys from being entered by returning true for keys you want to block.**

Summary: Running Long Jobs in Android

In this section, we discussed handling long-running tasks in Android applications without blocking the user interface (UI). These tasks, such as downloading media, can have unpredictable completion times and may disrupt the UI if run on the main thread.

Key Concepts:

1. **Blocking the UI: Running tasks on the main thread (UI thread) can cause the app to freeze, as it blocks UI updates.**
2. **Simulating Slow Connections: You can simulate slower connections in the Android emulator to test how your app behaves under different network conditions (e.g., slower cellular networks).**
3. **AsyncTask: An easier alternative to threads, though deprecated in API 30, AsyncTask allows for background work without blocking the UI. It supports background execution with methods like doInBackground() and UI updates via onProgressUpdate() and onPostExecute().**
4. **Thread Management: Threads provide more flexibility than AsyncTask but are harder to manage. They can run concurrently, and you need to carefully handle synchronization to avoid errors.**
5. **Synchronization with ReentrantLock: Using locks like ReentrantLock helps ensure that only one task runs at a time, preventing conflicts in concurrent threads.**

Q&A

Q1: What is the main issue with running long tasks on the main thread in Android?

- **A1: Running long tasks on the main thread blocks the UI, causing the app to freeze or become unresponsive. This leads to poor user experience.**

Q2: How can you simulate different network conditions in the Android emulator?

- **A2: You can use the extended controls menu in the Android emulator to simulate different network conditions by adjusting signal quality and network type (e.g., 2G, 3G, LTE).**

Q3: What is AsyncTask, and why is it used?

- **A3: AsyncTask is a class in Android that helps run tasks in the background without blocking the UI. It simplifies threading by serializing tasks and allows progress updates on the UI.**

Q4: Why is AsyncTask deprecated in API 30?

- **A4: AsyncTask is deprecated due to issues like context leaks, missed callbacks, inconsistent behavior on different Android versions, and exception handling problems.**

Q5: How do threads differ from AsyncTask in Android?

- **A5: Threads offer more flexibility and can run multiple tasks concurrently, but they are harder to manage and require careful synchronization. AsyncTask simplifies task management by serializing tasks but is limited to running only one task at a time.**

Q6: How can synchronization be handled in threads?

- **A6: Synchronization in threads can be handled using constructs like ReentrantLock, which ensures that only one thread executes a task at a time, preventing conflicts or errors when accessing shared resources.**

Q7: Is it always necessary to use threads or AsyncTask for background tasks?

- **A7: Not always. If tasks are short and fast, or if you have a reliable network connection, you might get away with running them on the main thread. However, for longer tasks, it's better to use AsyncTask or threads to avoid blocking the UI.**

WEEK 9

Summary: Fetching Images in Android App Development (COMP 10073)

This tutorial demonstrates how to create an Android application that can download and display images from the web. The application provides a simple interface with an EditText field for entering a URL and a Button to trigger the image download. The application displays a default cat image unless a user specifies a valid image URL. The download process is handled asynchronously to avoid blocking the main UI thread.

Key steps:

1. **activity_main.xml defines the UI with an EditText for the URL, a Button to trigger the image fetch, and an ImageView to display the fetched image.**
2. **MainActivity.java contains the logic to manage the UI and handle the click event for fetching the image. It uses an AsyncTask (DownloadImageTask) to download images from the internet in the background.**
3. **Internet Permission is required in the AndroidManifest.xml to allow the app to fetch images over the web.**
4. **AsyncTask is used to perform the background download and update the ImageView on completion.**

Questions and Answers

Q1: What does the getImage(View view) function do?

- **A1: It fetches the image from the provided URL or defaults to a cat image if the URL is not valid. It then creates and executes the DownloadImageTask to download the image in the background.**

Q2: Why do we use AsyncTask in this application?

- **A2: AsyncTask is used to download the image in the background without blocking the main UI thread, ensuring the app remains responsive while fetching the image.**

Q3: What is the role of the DownloadImageTask class?

- **A3: The DownloadImageTask class extends AsyncTask and handles the background task of downloading an image. It fetches the image from the URL and updates the ImageView once the download is complete.**

Q4: How does the app handle different image URLs?

- **A4: If the user types a URL containing "dog," the app will fetch a dog image. If the URL is invalid or doesn't contain "http," the app defaults to a cat image. Otherwise, it tries to fetch the image from the provided URL.**

Q5: What permissions are needed for the app to fetch images from the internet?

- **A5: The app requires the INTERNET permission to download images from the web. This is added in the AndroidManifest.xml file.**

Week 10

Short Summary:

This module introduces the ListView widget in Android, which is used to display a list of items. A ListView requires an adapter (such as ArrayAdapter) to connect the data with the views. The data can be an array or list of items, which are displayed in the ListView. The course demonstrates creating a simple app that generates a list of items and handles interactions like item clicks. Additionally, custom layouts for list items and adapters are discussed to display complex views with both text and images.

Key Topics Covered:

1. **ListView: Used to display scrollable lists of items.**
2. **ArrayAdapter: Binds data to a ListView and provides a layout resource for displaying each item.**
3. **Event Handling: List item click handling.**
4. **Custom Layouts: How to create custom list items using layouts.**
5. **Custom Adapters: Building custom adapters to work with complex data.**

Q&A:

1. **Q: What is a ListView in Android?**
    - **A: A ListView is a widget that displays a scrollable list of items. It uses an adapter to fetch the data and create views as needed.**
2. **Q: What is an ArrayAdapter in Android?**
    - **A: An ArrayAdapter is a bridge between a data source (like an array or list) and a ListView. It provides a layout to display each item in the list.**
3. **Q: How do you handle item clicks in a ListView?**
    - **A: You can handle item clicks by setting an OnItemClickListener on the ListView and defining the behavior in the onItemClick method.**
4. **Q: How can you customize the items displayed in a ListView?**
    - **A: You can customize ListView items by creating a custom layout for each item and using a custom adapter (e.g., extending ArrayAdapter) to bind the data.**
5. **Q: How do you create a custom adapter for a ListView?**
    - **A: To create a custom adapter, extend the ArrayAdapter class and override the getView method. This allows you to define how each item should be displayed in the ListView.**
6. **Q: What is the purpose of the getView method in a custom adapter?**
    - **A: The getView method is used to create and return the view for each item in the ListView. This method is where you bind the data (e.g., setting text or images) to the view components.**
7. **Q: How do you handle different types of data in a ListView?**
    - **A: You can define a custom class (e.g., ListItem) to store different types of data and use a custom adapter to display that data.**
8. **Q: What is the role of LayoutInflater in custom adapters?**
    - **A: LayoutInflater is used to inflate the custom layout for each item in the ListView. This is necessary for creating the view hierarchy from XML.**
9. **Q: How do you modify the layout of a ListView item?**
    - **A: You modify the layout by creating a custom XML layout for the item and using a custom adapter to bind data to the new layout.**

10. **Q: What is the ListItem class used for in the demo?**
    - **A: The ListItem class is used to store the data for each list item, which includes an image resource ID and a string for the text. This allows the adapter to display both text and images in the ListView.**

Summary:

In this week's Android Application Development class, you learned how to interact with a Web API to retrieve data and display it in an Android application. The main concepts covered include permissions, downloading data using AsyncTask, and displaying that data in a TextView in the UI. The Web API in this example is used to fetch science fair project data from a given URL, with a focus on filtering based on a year entered by the user.

You also learned to handle network requests with HttpURLConnection, read responses with BufferedReader, and handle large data using a StringBuilder. In the upcoming lesson, you'll use the GSON library to parse the JSON response into a more structured format.

Key Concepts:

- **Permissions: INTERNET and ACCESS_NETWORK_STATE permissions are required for network operations.**
- **AsyncTask: Used for background operations like downloading data from the web.**
- **HttpURLConnection: Manages the connection to the web API and retrieves the response.**
- **TextView: Displays the downloaded data in the UI.**

Q&A:

1. **What permissions do you need to access a Web API in Android?**
    - **You need the INTERNET and ACCESS_NETWORK_STATE permissions.**

2. **What class is used to download data from a URL in the background?**
    - **The AsyncTask class is used to download data in the background.**
3. **How do you ensure data is displayed in a scrollable manner?**
    - **You set setMovementMethod(new ScrollingMovementMethod()) on the TextView to make it scrollable.**
4. **How do you handle the download of large data in Android?**
    - **You use a BufferedReader and a StringBuilder to efficiently handle and append the data.**
5. **What is the purpose of onPostExecute() in an AsyncTask?**
    - **onPostExecute() is used to process and display the result once the background task (data download) is completed.**
6. **How does the HttpURLConnection work?**
    - **It opens a connection to the URL and retrieves the response. The response body is read from an input stream, and headers provide metadata about the response.**
7. **What does startDownload() do in this example?**
    - **It builds the API request URL based on the year entered by the user and triggers the DownloadAsyncTask to download the data.**
8. **How is the JSON data formatted for readability?**
    - **Carriage returns are inserted in the downloaded JSON data to make it more readable before displaying it in the TextView.**

week 11

Summary: Android Application Development - Web API Demonstration (Part II)

In this lecture, you continue working with Web APIs, focusing on downloading and displaying data from an external API in an Android app. The app interacts with the Bay Area Science and Engineering Fair (BASEF) API, which provides data about science fair

projects. The core concepts involve making HTTP requests, parsing JSON responses, and displaying the data dynamically in the UI.

Key elements:

- **Permissions: Make sure to include ACCESS_NETWORK_STATE and INTERNET in the Android manifest.**
- **BASEF API: Data contains project details such as ProjectNum, NameFull, SchoolName, Description, and picURI.**
- **UI Components: EditText for keyword input, TextView for displaying data, and a Button to trigger the data download.**
- **AsyncTask: Used for downloading the data in the background, with results displayed after parsing the JSON response.**
- **GSON: A Java library for converting Java objects to JSON and vice versa. Used to parse the JSON response into an ArrayList of Project objects.**

Key Steps:

1. **Manifest File: Include the necessary permissions and allow clear-text traffic for HTTP.**
2. **UI Layout: Use LinearLayout, EditText, Button, and TextView components to create the app's interface.**
3. **MainActivity: Handles the setup, downloading, and displaying of the data. It also manages text updates in TextView components.**
4. **AsyncTask: Fetches data from the BASEF API and handles errors (e.g., malformed URLs or IO exceptions).**
5. **JSON Parsing with GSON: Parse the JSON response into Java objects, then display key project details in the UI.**

Q&A:

1. **Q: What permissions need to be included in the Android manifest for this application?**
    - **A: The app needs ACCESS_NETWORK_STATE and INTERNET permissions to access the network.**
2. **Q: What does the onPostExecute method do in the DownloadAsyncTask class?**

- **A: It is called after the background task finishes downloading data. It parses the JSON response and updates the UI with the relevant project details.**

3. **Q: How do we parse JSON data in Android?**
    - **A: We use the GSON library to convert the JSON data into Java objects. The JSON response is parsed into an ArrayList of Project objects.**
4. **Q: What is the role of the Gson object in this example?**
    - **A: The Gson object is used to parse the JSON data into an ArrayList of Project objects and to convert between JSON and Java objects.**
5. **Q: What is the purpose of the EditText widget in the UI?**
    - **A: The EditText is used for user input, allowing the user to enter a keyword (e.g., project description) to filter the data.**
6. **Q: How is the app structured to handle background tasks like data downloading?**
    - **A: The DownloadAsyncTask class performs the downloading of data in the background to avoid blocking the main UI thread, and the result is processed in onPostExecute.**
7. **Q: What is the issue with the app when the phone is rotated?**
    - **A: The layout is destroyed during rotation, which can lead to the app re-downloading the data. This is inefficient, and it's suggested to avoid expensive operations during rotation.**

Android SQLite Overview (COMP 10073)

SQLite in Android is a lightweight relational database system that comes integrated with Android OS. It allows apps to create their own local databases with a simple SQL interface and minimal memory usage. The database is stored on the device and can be accessed via /data/data/<app_name>/databases/.

Key Concepts:

1. **SQLiteOpenHelper:**

- **A helper class for managing database creation, version management, and schema upgrades.**
- **Methods:**
  - **onCreate(SQLiteDatabase): Creates tables and initializes data when the database is first created.**
  - **onUpgrade(SQLiteDatabase, int, int): Handles database schema upgrades when the version number changes.**
  - **onOpen(SQLiteDatabase): Optional method for custom behavior when the database is opened.**

2. **Database Operations:**
   - **getWritableDatabase(): Opens the database for reading and writing. If it doesn't exist, it is created.**
   - **ContentValues: Used to insert data into the database by specifying column names and values.**
   - **Cursor: Used to query the database and iterate over the result set.**
3. **Example Workflow:**
   - **MyDbHelper: A subclass of SQLiteOpenHelper to manage database creation and versioning.**
   - **MainActivity: Handles user interaction, inserts data into the database, and displays database contents in a TextView.**

Q&A:

1. What is SQLite used for in Android apps?
SQLite is used for storing structured data locally on Android devices. It is especially useful for apps that need to store large amounts of data without relying on a network connection.

2. What is the role of SQLiteOpenHelper?
SQLiteOpenHelper simplifies database management by handling tasks such as database creation, schema updates, and database opening.

3. How do you add data to the database?
You create a ContentValues object, put the data into it, and then use the insert() method of SQLiteDatabase to insert the data into a specific table.

4. What does a Cursor do?
A Cursor is used to read the result of a database query. It provides access to the rows returned by a query, allowing iteration through the results.

5. How do you display database results in an Android app?
You can use a Cursor to retrieve all database entries and then append them into a string that is displayed in a TextView or other UI components.

week 12

Summary:

This lecture focuses on using SQLite in Android development to manage databases. SQLite is an embedded relational database engine that is self-contained and serverless, making it ideal for Android apps. The course introduces creating, updating, and deleting records, and queries in a local database, using SQLiteOpenHelper and SQLiteDatabase classes. The database schema is specified, and data manipulation is done through SQL commands such as INSERT, UPDATE, and DELETE. The course also covers the use of ContentValues for adding data, querying the database using Cursor, and implementing actions in Android activities like inserting, updating, deleting, and querying records.

Q&A:

1. **What is SQLite and how is it used in Android?**
    - **SQLite is a lightweight, embedded SQL database engine used in Android applications for local data storage. It allows you to create, read, update, and delete records using SQL commands within an app, without the need for a separate database server.**
2. **What is the role of SQLiteOpenHelper?**

- **SQLiteOpenHelper is a helper class that simplifies database creation and version management. It handles database setup and upgrades. You extend this class to define your schema and manage database creation and version changes.**

3. **What is ContentValues used for in SQLite?**
    - **ContentValues is used to store key-value pairs, where the keys are column names and the values are the data to be inserted or updated in the database.**
4. **How do you insert data into a SQLite database in Android?**
    - **Data is inserted using the insert() method on an SQLiteDatabase object, passing a ContentValues object containing the data to be inserted.**
5. **What is a Cursor in SQLite and how is it used?**
    - **A Cursor is an interface for accessing data returned by database queries. It is used to iterate through the rows of a query result.**
6. **What is a common bug in SQL queries within Android's SQLite?**
    - **A common issue is SQL injection vulnerabilities and improperly sanitized input, which can crash the app or cause unintended behavior.**
7. **How do you update a record in SQLite?**
    - **Use the update() method of SQLiteDatabase, providing a ContentValues object with the new values and specifying the record to update via a WHERE clause.**
8. **How do you delete a record in SQLite?**
    - **Use the delete() method of SQLiteDatabase, specifying the record to delete using a WHERE clause.**
9. **How do you query data in SQLite?**
    - **Use the query() method of SQLiteDatabase, providing column names, selection conditions, and sorting parameters.**
10. **How do you avoid SQL injection in database operations?**

- **Avoid direct concatenation of user input into SQL queries. Instead, use parameterized queries or methods like SQLiteDatabase.query() with placeholders for parameters.**

Android App Development Final Exam Review

Week-by-Week Summary and Key Concepts

Week 1: Introduction to Android & Android Studio

Key Concepts:

- Android is an open-source mobile OS developed by Google
- Android Studio is the official IDE for Android development
- Main programming languages: Java and Kotlin
- Core app components:
    - Activities (screens)
    - Fragments (reusable UI parts)
    - Services (background tasks)
    - Broadcast Receivers (system events)
    - Content Providers (data sharing)

Q&A:

1. What is an Activity in Android?
    - A: A single screen that users interact with (like a room in a house).
2. What file defines the app's permissions and components?
    - A: AndroidManifest.xml

Week 2: Hello World App Basics

Key Concepts:

- Project structure: MainActivity, activity_main.xml, manifest
- onCreate() is the entry point (similar to JavaFX's start())
- UI defined in XML layouts (ConstraintLayout by default)
- Device rotation triggers onCreate() again
- Gradle manages dependencies and builds

Q&A:

1. What does setContentView() do?
    - A: Sets the UI layout for an Activity by referencing an XML file.
2. Why remove EdgeToEdge.enable() in early examples?
    - A: It adds unnecessary complexity for beginners.

Week 3: Activities and Intents

Key Concepts:

- Activity lifecycle: onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy()
- Use Intents to switch between activities
- putExtra()/getExtra() to pass data between activities
- Activity stack is LIFO (Back button pops current activity)

Q&A:

1. What's the difference between onPause() and onStop()?
    - A: onPause() when activity is partially visible, onStop() when completely hidden.
2. How do you start a new activity?
    - A: startActivity(new Intent(CurrentActivity.this, NextActivity.class))

Week 4: Data Sharing Between Activities

Key Concepts:

- Avoid static variables for data sharing (can cause memory leaks)
- Preferred method: Intent.putExtra()
- ActivityResultLauncher replaces deprecated startActivityForResult()
- Guidelines and chains for responsive layouts

Q&A:

1. Why is using static getters risky for activity data sharing?
    - A: Can cause memory leaks and crashes if activities are destroyed.

2. What's the modern way to get results from an activity?
    - A: ActivityResultLauncher with registerForActivityResult()

Week 5: UI Components & Persistence

Key Concepts:

- Strings stored in res/values/strings.xml
- Spinners for dropdown selection
- SharedPreferences for simple persistent data
- Coding conventions: proper commenting, exception handling, method length

Q&A:

1. Where should app strings be stored?
    - A: res/values/strings.xml, accessed via @string/name (XML) or R.string.name (Java).
2. What's wrong with empty catch blocks?
    - A: They silently hide errors instead of handling them properly.

Week 6: Rotation & User Feedback

Key Concepts:

- Handle rotation with separate layout files (layout-land/ for landscape)
- Display class detects orientation changes
- Toasts: brief non-interactive messages
- Snackbars: messages with optional actions
- Dialogs: require user interaction

Q&A:

1. How to create landscape-specific layouts?
    - A: Create layout-land/ folder with alternative XML files.
2. Difference between Toast and Snackbar?
    - A: Toasts are simple messages; Snackbars can include actions and stay longer.

Week 7: Broadcast Receivers

Key Concepts:

- Broadcasts are system-wide notifications
- Receivers listen for specific broadcasts (declared in manifest)
- Need permissions (like RECEIVE_SMS)
- Runtime permissions required for dangerous permissions (API 23+)

Q&A:

1. What changed with broadcasts in Android 8.0?
    - A: Restricted implicit broadcasts to improve performance/battery life.
2. What are the two permission types in Android?
    - A: Normal (automatic) and Dangerous (require runtime approval).

Week 8: Background Tasks

Key Concepts:

- Never block UI thread with long operations
- AsyncTask (deprecated but useful for learning) has:
    - doInBackground() - runs off UI thread
    - onPostExecute() - runs on UI thread after
- Threads offer more control but need manual management

Q&A:

1. Why was AsyncTask deprecated?
    - A: Issues with memory leaks, inconsistent behavior across versions.
2. What happens if you block the UI thread?
    - A: App becomes unresponsive (ANR - Application Not Responding).

Week 9: Networking & Images

Key Concepts:

- Need INTERNET permission for network access

- Use AsyncTask or threads for network operations
- HttpURLConnection for web requests
- ImageView displays images (local or remote)

Q&A:

1. What permission is needed for internet access?
    - A: INTERNET permission in manifest.
2. Why do network operations need AsyncTask?
    - A: To avoid blocking the UI thread during slow network operations.

Week 10: Lists & Web APIs

Key Concepts:

- ListView displays scrollable lists
- ArrayAdapter connects data to ListView
- Custom adapters for complex list items
- Web APIs provide remote data (JSON/XML)

Q&A:

1. What's the role of an Adapter?
    - A: Bridges data source and ListView, creating views for each item.
2. How to handle item clicks in a ListView?
    - A: setOnItemClickListener() with onItemClick() handler.

Week 11-12: SQLite Database

Key Concepts:

- SQLite is lightweight embedded database
- SQLiteOpenHelper manages database creation/versioning
- ContentValues for inserting data
- Cursor for query results
- CRUD operations: insert(), update(), delete(), query()

Q&A:

1. What are the key SQLiteOpenHelper methods?
    - A: onCreate() (initial setup), onUpgrade() (schema changes).
2. How to prevent SQL injection?
    - A: Use parameterized queries, not string concatenation.

Final Exam Tips

1. **Focus Areas:**
    - Activity lifecycle and Intents
    - Data sharing between components
    - Background tasks (AsyncTask/threads)
    - Basic UI components and layouts
    - SQLite database operations
    - Permissions and broadcasts
2. **Exam Strategy:**
    - Read questions carefully
    - Manage time (about 1 minute per point)
    - For code questions:
        - Identify lifecycle methods
        - Check for proper threading
        - Verify data sharing approaches
        - Look for proper error handling
3. **Common Pitfalls:**
    - Blocking UI thread
    - Improper activity/data lifecycle handling
    - Insecure data sharing methods
    - Missing permissions

- Poor exception handling

General Exam Questions

Q1: What is the purpose of a final exam?
A: To assess how much a student knows in a short period of time, similar to a job interview where candidates must demonstrate their skills under pressure.

Q2: What is the format of the final exam?
A:

- Closed book
- Covers most of the course
- ~6 written questions
- Some involve Java code
- No multiple choice or math
- Worth 120 points (~1 point per minute)
- Worth 25% of the final grade

🗂️ **Week-by-Week Questions & Answers**

Week 1: Introduction, Tools, Hierarchy

Q3: Briefly describe differences between Android development tools like Kotlin, React Native, and Flutter.
A:

- **Kotlin**: Official Android language, interoperable with Java.
- **React Native**: JavaScript-based, cross-platform.
- **Flutter**: Dart-based, fast UI rendering, also cross-platform.
  Each has unique syntax, ecosystems, and performance trade-offs.

Week 2: First Project, Layout, onClick()

Q4: What default template code is typically removed and why?
A: Code like EdgeToEdge.enable(this); and complex insets handlers are often removed for simplicity when first learning.

Q5: How does onClick() work in Android?
A: It links a button in XML to a method in Java using android:onClick="methodName", allowing button taps to trigger Java code.

Week 3: Activity Lifecycle, Intents

Q6: What is the purpose of onCreate(), onStart(), and onResume()?
A:

- onCreate(): Initializes the activity.
- onStart(): Called just before UI becomes visible.
- onResume(): Called when the activity becomes interactive.

Q7: Name 3 other important lifecycle methods.
A: onPause(), onStop(), onDestroy()

Q8: How is putExtra() used in Intents?
A: Adds extra data to an Intent before launching a new Activity:

java

CopyEdit

intent.putExtra("key", "value");

Q9: Why can this be used with setOnClickListener()?
A: Because the current class implements View.OnClickListener, and this refers to that instance.

Week 5: Shared Preferences, Coding Conventions

Q10: Where is SharedPreferences data stored and how long does it last?
A: Stored in internal storage in a private XML file; persists across app restarts until explicitly removed.

Q11: Why avoid anonymous classes and lambdas in Android development (as per course)?
A: They can reduce readability and debuggability for beginners. Prefer named inner classes or implementing interfaces for clarity.

Q12: Name 2 additional coding conventions from Week 5.
A:

1. Consistent naming (camelCase for variables)
2. Logging with consistent tags (e.g., ==TAG==) for easier debugging

---

Week 6: Rotation, Toast vs Snackbar

Q13: Aside from layout, what are 2 issues developers face with rotation?
A:

1. Preserving app state
2. Re-initialization of resources or UI elements

Q14: Identify bugs in this line: TextView layout = findViewById("layout");
A: findViewById() expects an ID (e.g., R.id.layout), not a String literal.

Q15: What's the difference between Toast and Snackbar?
A:

- **Toast**: Simple message, doesn't require user interaction.
- **Snackbar**: Appears at bottom, can include action buttons.
  Use Snackbar when you need user action or feedback.

Week 7: Permissions, Broadcast Receivers

Q16: Why is .requestPermissions() required for SMS access?
A: Android (API 23+) requires runtime permission requests for sensitive features like SMS. Without it, app crashes or silently fails.

Q17: How can you tell if a permission was granted?
A: Override onRequestPermissionsResult() to check permission results.

Q18: Why is the onReceive() method shown in Week 7 considered heavy-handed?
A: Launching UI directly from a broadcast can be disruptive. Better approach: show notification or defer to user action.

Week 8: AsyncTask, Threads, Interfaces

Q19: Why can't Java use multiple inheritance?
A: To avoid ambiguity from the "diamond problem." Interfaces offer a safer way to inherit behavior.

Q20: What's the difference between implements and extends?
A:

- implements: Adds interface behavior.
- extends: Inherits functionality from a class.

Q21: Compare doAsync() (AsyncTask) vs doThreads() (Thread):
A:

- **AsyncTask**: Simplifies threading + UI updates.
- **Thread**: Manual management; more control, less convenience.
  Use AsyncTask for quick tasks, Threads for more control.

Week 9: PicViewer

Q22: Will you be tested on PicViewer in the exam?
A: No, it won't be on the exam.

Week 10: Lists and Adapters

Q23: What is a ListAdapter and why is it used?
A: Binds data to a ListView. Converts data (like Strings) into viewable list items using layout resources.

Q24: What are standard Android list item layouts?
A:

- android.R.layout.simple_list_item_1

- simple_list_item_2
- two_line_list_item

Week 11–12: SQLite and Web APIs

Q25: Will there be a question about SQLite?
A: Yes. Expect a question based on SQLite implementation.

Q26: What topics related to Web APIs should be understood?
A:

- HTTP requests/responses
- Permissions for Internet access
- JSON handling with libraries like GSON