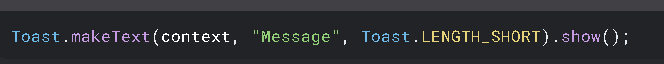
**UNIT-IV**

**1. Notifications**

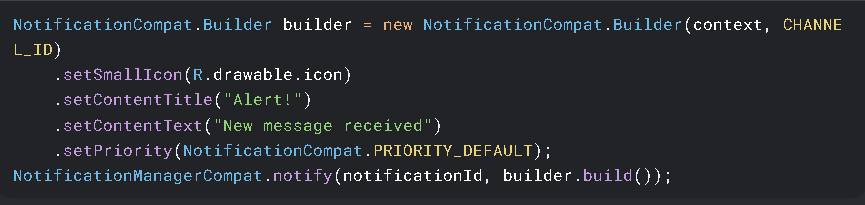
**Definition**:  
Notifications are messages displayed to the user outside the app's UI, typically in the status bar or as pop-ups (Toasts). They alert users about events (e.g., messages, alarms, or system updates) even when the app is not active.

**Key Components**:

1. **Toast Notifications**:
   * **What**: Temporary, non-interactive messages that appear briefly at the bottom of the screen.
   * **Use Case**: Feedback for actions (e.g., "Saved successfully").



1. **Status Bar Notifications**:
   * **What**: Persistent messages displayed in the notification drawer.
   * **Components**:
     + **Icon**: Small image representing the app.
     + **Title/Content**: Brief summary (e.g., "New email").
     + **Intent**: Action triggered when tapped (e.g., opens an activity).
   * **Code Example (Using**NotificationCompat.Builder**)**:



1. **Notification Channels (Android 8.0+)**:
   * **What**: Categories to manage notification behavior (sound, priority, etc.).
   * **Why**: Users can customize preferences per channel.

**Types of Notifications**:

* **Foreground Notifications**: For ongoing tasks (e.g., music playback).
* **Interactive Notifications**: Include buttons (e.g., "Reply" in messaging apps).
* **Heads-up Notifications**: Pop on screen for high-priority alerts.

**Best Practices**:

* Use clear, concise text.
* Set appropriate priority levels.
* Handle notification channels for Android 8.0+.

**Common Pitfalls**:

* Overloading users with unnecessary notifications.
* Forgetting to cancel notifications after their purpose is served.

**2. Broadcast Receivers**

**Definition**:  
A **Broadcast Receiver** is an Android component that allows apps to respond to system-wide or app-specific events (broadcasts) even when the app isn’t running. It acts as a "listener" for intents sent by the system or other apps (e.g., battery low, SMS received, airplane mode toggled).

**Types of Broadcasts**:

1. **System Broadcasts**:
   * Sent by the Android system (e.g., ACTION\_BOOT\_COMPLETED, ACTION\_BATTERY\_LOW).
   * **Example**: Listen for device boot to start a background service.
2. **Custom Broadcasts**:
   * Sent by apps (e.g., notifying other components of an event like "data downloaded").
3. **Explicit vs. Implicit Broadcasts**:
   * **Explicit**: Targeted at a specific app/component (e.g., Intent with the receiver’s class name).
   * **Implicit**: No specific target (e.g., Intent.ACTION\_SEND to share data).

**3. Services**

**Definition**:  
A Service is an Android component that performs long-running operations **in the background** without a user interface. Unlike Activities, Services continue running even if the user switches apps.

**Key Characteristics**

| **Feature** | **Description** |
| --- | --- |
| **No UI** | Runs invisibly (e.g., music playback, file downloads). |
| **Lifecycle** | Managed by onCreate(), onStartCommand(), onDestroy(). |
| **Threading** | Runs on the **main thread** by default (must spawn worker threads manually). |

**Types of Services**

1. **Started Service**
   * **Purpose**: Runs indefinitely until explicitly stopped (e.g., syncing data).
   * **Trigger**: Started via startService(intent).
   * **Stopping**: Call stopSelf() or stopService(intent).
2. **Bound Service**

* **Purpose**: Acts as a "server" for components (e.g., Activities) to interact with.
* **Trigger**: Bound via bindService(intent, connection, flags).
* **Communication**: Uses IBinder interface (exposes methods to clients).

1. **Foreground Service**

* **Purpose**: For user-visible tasks (e.g., music players). Requires a **notification**.
* **Android 9+**: Needs FOREGROUND\_SERVICE permission.

**Lifecycle in Depth**

1. **Started Service**:
   * onCreate() → onStartCommand() → (runs until) → onDestroy().
2. **Bound Service**:
   * onCreate() → onBind() → (clients interact) → onUnbind() → onDestroy().

**4. Notifications (Alarms) in Android**

Alarms allow apps to schedule tasks or notifications **at specific times**, even if the app isn’t running. Unlike timers (Handler.postDelayed), alarms persist after device reboots and can trigger actions while the app is closed.

**1. AlarmManager Basics**

**Key Features**:

* Schedule one-time or repeating tasks.
* Wake up the device (if necessary) to execute tasks.
* Supports **exact** (Android 4.4+) and **inexact** alarms.

**Types of Alarms**:

| **Type** | **Behavior** |
| --- | --- |
| RTC | Fires at a specified clock time (sleeps if device is off). |
| RTC\_WAKEUP | Wakes up the device to trigger the alarm. |
| ELAPSED\_REALTIME | Time relative to system boot (ignores timezone changes). |
| ELAPSED\_REALTIME\_WAKEUP | Wakes device based on time since boot. |

**5. Custom Components in Android**

Custom components allow developers to **extend or modify existing UI elements** (like buttons, tabs, dialogs) or create **entirely new ones** for unique user experiences. Below is a breakdown of key custom components with minimal code and more theory.

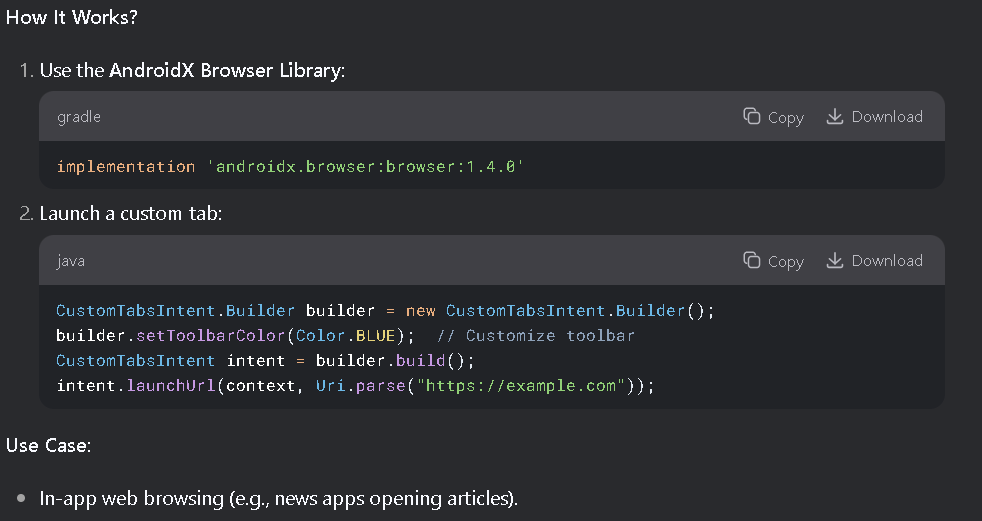
**1. Custom Tabs**

* Customized versions of Chrome tabs embedded in your app (for web content).
* Provides a **seamless transition** between app and web pages.

**Why Use?**  
✅ Maintains app branding (custom toolbar colors, icons).  
✅ Faster than opening a browser.  
✅ Supports gestures and custom actions.

**Key Features**:

* **Toolbar Customization**: Change color, add buttons.
* **Animation Control**: Smooth transitions.
* **Behavior Tweaks**: Handle navigation events.



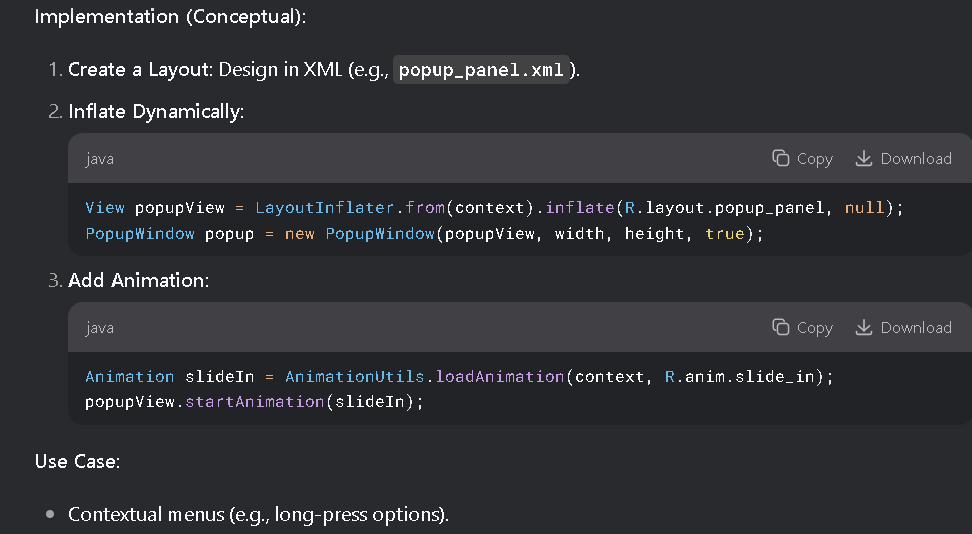
**2. Custom Animated Popup Panels**

* Floating UI elements that appear dynamically (e.g., tooltips, menus, alerts) with **animations**.

**Types**:

* **Dialogs**: Modal popups (user must interact).
* **Bottom Sheets**: Slide-up panels (e.g., Google Maps).
* **Toasts/Snackbars**: Non-blocking messages.

**Why Use?**  
✅ Better UX with smooth animations.  
✅ Focus user attention on critical actions.



**3. Other Custom Components**

**(A) Custom Views**

* **What?** Override View or ViewGroup to create **entirely new UI elements**.
* **Example**: A circular progress bar, signature pad.
* **Key Methods**:
  + onDraw(): Define rendering.
  + onMeasure(): Set dimensions.

**(B) Compound Components**

* **What?** Combine existing widgets (e.g., TextView + ImageView).
* **Example**: A "RatingBar" with stars.

**(C) Custom Drawables**

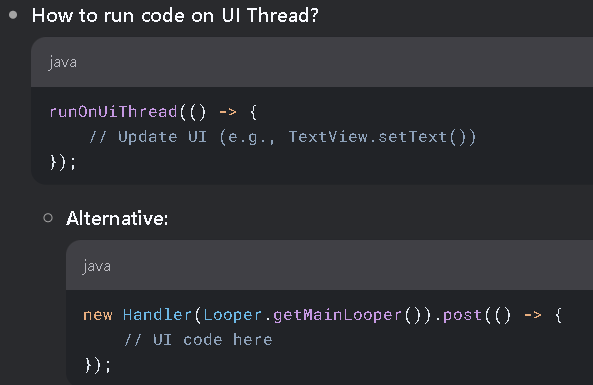
* **What?** Shape-shifting backgrounds/icons (e.g., gradient buttons).
* **Tools**:
  + **XML Shapes**: <shape>, <selector>.
  + **Vector Drawables**: Scalable graphics.

**6. Threads in Android: UI Thread, Worker Threads, Handlers & Runnable**

**1. Main Thread (UI Thread)**

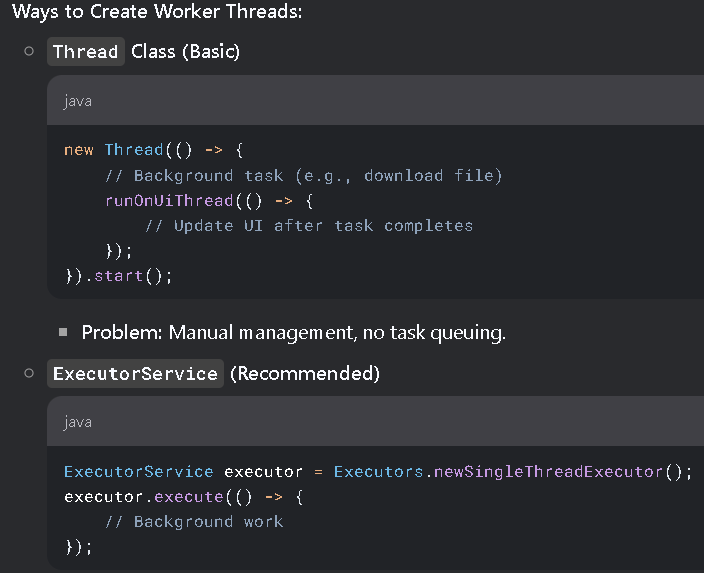
* **What?**
  + The primary thread where all UI operations (e.g., button clicks, screen updates) run.
  + Blocking it causes ANR (App Not Responding) errors.

**Key Rule:** Never perform long-running tasks (network calls, DB ops) on the UI thread**.**



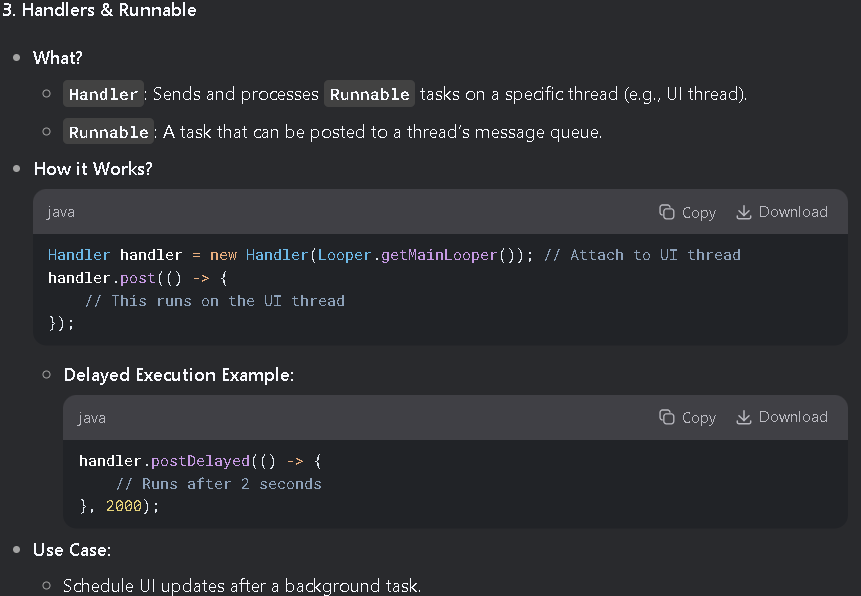
**2. Worker Threads (Background Threads)**:

* + Offload heavy tasks to avoid freezing the UI.



**3. Handlers & Runnable**

* + Handler: Sends and processes Runnable tasks on a specific thread (e.g., UI thread).
  + Runnable: A task that can be posted to a thread’s message queue.



**4. Threading Best Practices**

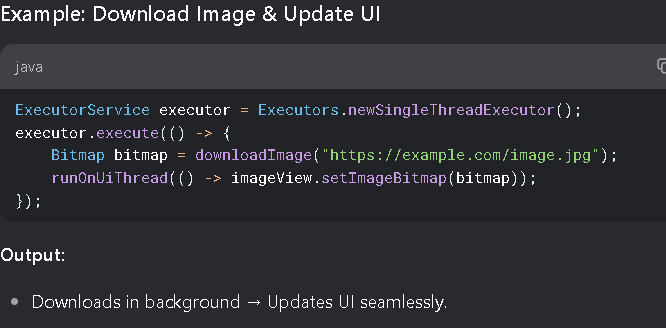
| **Technique** | **When to Use?** | **Pros & Cons** |
| --- | --- | --- |
| Thread | Simple one-off tasks. | ❌ Manual management, no pooling. |
| ExecutorService | Concurrent tasks (e.g., parallel downloads). | ✅ Thread pooling, efficient. |
| Handler | Communicating between threads. | ✅ Precise scheduling (UI updates). |
| AsyncTask | **Deprecated** (Use Coroutines/WorkManager). | ❌ Memory leaks, inflexible. |

**5. Modern Alternatives**



**Key Takeaways**

1. **UI Thread** = Only for fast operations.
2. **Worker Threads** = For heavy tasks (DB, network).
3. **Handlers** = Bridge between threads (e.g., post results to UI).
4. **Avoid**AsyncTask → Use **Coroutines** or ExecutorService.



**7. Networking in Android**

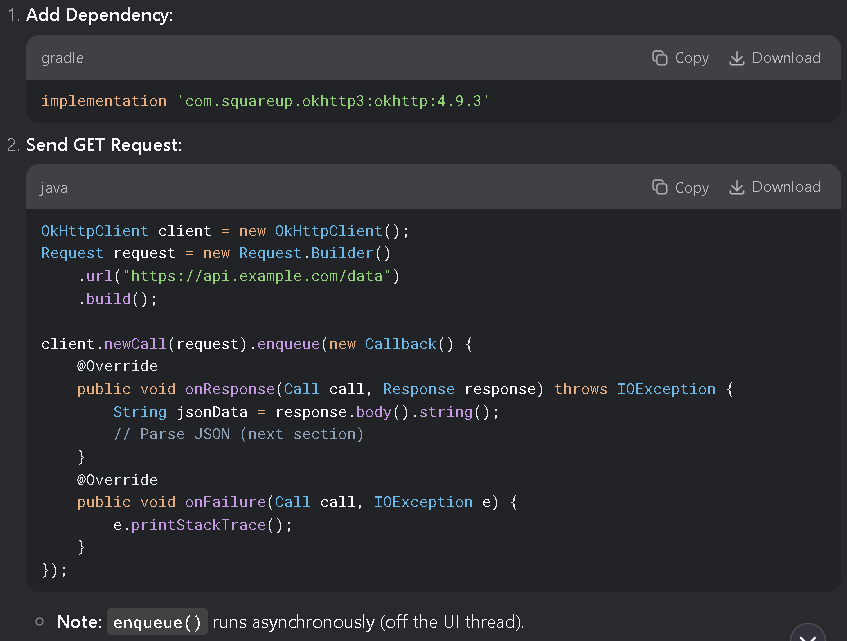
Networking allows apps to communicate with servers, fetch data (APIs), and enable real-time features. Below is a structured breakdown of **HTTP, JSON, and Sockets** in Android.

**1. Consuming Web Services Using HTTP**

**Key Libraries:**

* HttpURLConnection (Built-in, low-level)
* OkHttp (Recommended: Efficient, supports HTTP/2)
* Retrofit (Best for REST APIs: Converts HTTP calls to Java/Kotlin interfaces)

**Steps to Make an HTTP Request (OkHttp Example)**

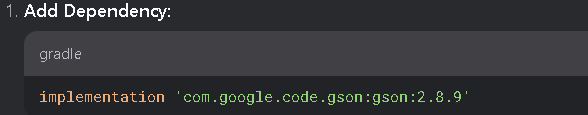


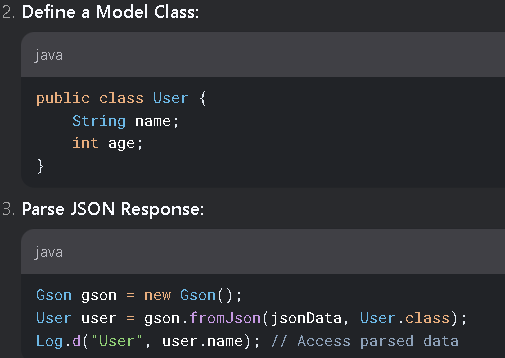
**2. Consuming JSON Services**

**JSON Parsing Libraries:**

* org.json (Built-in, verbose)
* Gson (Google’s library: Converts JSON ↔ Objects)
* Moshi (Modern, Kotlin-friendly)

**Example: Gson for JSON Parsing**





**3. Sockets Programming (Real-Time Communication)**

* **Use Cases:** Chat apps, multiplayer games, live tracking.
* **Protocols:** TCP (reliable), UDP (fast, unordered).



**Key Notes:**

* Sockets **must run on a background thread** (blocking operations).
* For production, use **WebSockets** (OkHttp supports them) or **Firebase Realtime Database**.

**Best Practices**

1. **Always Use HTTPS** (Not HTTP) for security.
2. **Handle Offline Scenarios:** Cache responses with Room or Retrofit + Cache.
3. **Threading:** Never block the UI thread (use OkHttp’s enqueue() or Coroutines).
4. **Timeouts:** Set to avoid hanging requests: