**Section 43 — Android Developer Roadmap (Detailed Notes)**

**1. Programming Languages**

* **Java**
  + Class-based, OOP, mature, widely used in Android SDK.
  + Verbose, but excellent for learning fundamentals.
* **Kotlin** *(Google’s preferred language since 2019)*
  + Concise → fewer lines, less boilerplate.
  + Interoperable with Java.
* **Recommendation:** Start with Java → transition to Kotlin for modern features.
* **Industry trend:** Kotlin-first development in new projects.

**2. Android Studio Setup**

* Download from developer.android.com/studio.
* Includes IDE, UI Designer, Gradle build tools, Emulator, Debugger.
* **Project Structure:**
  1. **Java/Kotlin classes** → Business logic.
  2. **XML Layouts** → UI design.
  3. **Gradle files** → Dependencies & build configurations.
* **Tip:** Use Gradle for versioned dependency management.

**3. Android Components**

1. **Activities** → UI screen controllers.
   * Lifecycle: onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy().

java

CopyEdit

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main); // Load UI

}

}

1. **Services** → Background tasks (music playback, data sync).
2. **Broadcast Receivers** → Listen for system-wide events (e.g., connectivity changes).
3. **Content Providers** → Data sharing between apps (contacts, media).

**4. Intents**

* **Explicit** → Launch specific component in your app.

java

CopyEdit

Intent i = new Intent(this, SecondActivity.class);

startActivity(i);

* **Implicit** → Request action from another app.

java

CopyEdit

Intent i = new Intent(Intent.ACTION\_VIEW, Uri.parse("https://google.com"));

startActivity(i);

* **Intent Filters** in AndroidManifest.xml define which actions your app can handle.

**5. Static User Interface**

* **Views**: Button, TextView, ImageView, EditText.
* **ViewGroups**: Containers (LinearLayout, RelativeLayout, ConstraintLayout).
* **Best practice:** Use **ConstraintLayout** for performance and responsive design.

**6. Dynamic User Interface**

* **RecyclerView** → Efficient lists/grids, recycles views for performance.
* **ViewPager** → Swipe between fragments/pages.
* **Spinner** → Dropdown selection list.

**7. Custom Views (Canvas API)**

* Core objects:
  1. **Bitmap** → Holds pixels.
  2. **Canvas** → Hosts draw calls.
  3. **Paint** → Colors/styles.
  4. **Primitives** → Shapes, text, paths.

**8. UI Resources**

* Stored in /res:
  + drawable/ → Images, shapes.
  + values/strings.xml → Localizable text.
  + values/styles.xml → Themes, UI styles.
* **Best practice:** Keep all text in strings.xml for translation support.

**9. Fragments**

* Reusable UI modules inside activities.
* Own lifecycle (onAttach, onCreateView, onDestroyView).
* Good for tablet layouts or multi-pane UIs.

**10. Support UI Components**

* **ProgressBar** → Show background task progress.
* **Dialogs** → Confirmation prompts.
* **Toast** → Short messages.
* **Snackbar** → Actionable bottom messages.

**11. Storage Options**

1. **SharedPreferences** → Small key-value data (e.g., settings).
2. **File System** → Store files in app’s internal/external storage.
3. **SQLite** → Local DB.
4. **Room DB** → ORM over SQLite for easier, type-safe access.

**12. Build & Gradle**

* **Debug build** → For testing.
* **Release build** → Optimized, signed for Play Store.
* Gradle manages dependencies, build tasks.

**13. Threading**

* **Main/UI Thread** → Handles UI updates.
* Heavy tasks → Run in **background threads** (Handler, Coroutines, RxJava).
* **Rule:** Only main thread can update UI.

**14. Debugging**

* **Logcat** → View logs.
* **Memory Profiler** → Detect high memory usage.
* **Exception handling** → Use try-catch to avoid crashes.

**15. Memory Leaks**

* Common cause: holding Context too long.
* Use WeakReference if needed.
* Always release resources in onDestroy().

**16. Third-Party Libraries**

* **Networking**: Retrofit, Volley, Fast Android Networking.
* **Image Loading**: Glide (recommended), Picasso.
* **Dependency Injection**: Dagger (modern: Hilt).

**17. Data Formats**

* **JSON** → Common API format.
* **Gson** → Convert JSON ↔ Java objects.

java

CopyEdit

Gson gson = new Gson();

MyClass obj = gson.fromJson(jsonString, MyClass.class);

**18. Android Jetpack**

* **Foundation**: AppCompat, MultiDex.
* **Architecture**: LiveData, ViewModel, Navigation, WorkManager, Paging.
* **UI**: Animations, transitions.
* **Behavior**: Notifications, Permissions.

**19. Architecture Patterns**

* **MVVM (Recommended)** → Model-View-ViewModel separates UI from logic.
* ViewModel survives configuration changes (e.g., screen rotation).

**20. Unit Testing**

* **Local** → Runs without Android dependencies.
* **Instrumentation** → Runs on device/emulator with Android framework.

**21. Firebase**

* **Auth**, **Firestore**, **Cloud Messaging**, **Crashlytics**, **Remote Config**, **Analytics**.
* Cloud-hosted, scalable backend.

**22. Security**

* **ProGuard/R8** → Obfuscate & shrink code.
* Encrypt sensitive data.
* Keep **Keystore file** safe.

**23. App Release**

* Build **AAB** (Android App Bundle).
* Sign with Keystore.
* Upload via Google Play Console.

**24. Keep Learning**

* Stay updated with Android docs, Jetpack releases, new libraries.

**Part B — Missing but Important for Future Projects**

1. **Jetpack Compose** — Modern declarative UI, replacing XML layouts.
2. **Kotlin Coroutines Flow** — Advanced async/reactive programming.
3. **Hilt DI** — Simpler DI framework than Dagger.
4. **DataStore** — Modern replacement for SharedPreferences.
5. **Navigation Component** — Safer navigation with argument passing.
6. **Paging 3** — Efficiently load paginated API data.
7. **WorkManager** — Background jobs with guaranteed execution.
8. **Modularization** — Break large apps into modules for scalability.
9. **CI/CD** — Automate builds/tests/deployments.
10. **Offline-first** — Cache & sync strategies for low-network areas.