In Android development, design patterns are essential tools for writing clean, scalable, and maintainable code. The **Singleton** pattern is just one of many that are commonly used. Here's an overview of major design patterns (and related architecture patterns) that are frequently used in Android:

**🧩 Creational Design Patterns**

These deal with object creation.

1. **Singleton**
   * Ensures a class has only one instance and provides a global point of access to it.
   * Used for classes like Retrofit, Room DB instance, SharedPreferences.
2. **Factory Method**
   * Defines an interface for creating an object but lets subclasses alter the type of objects that will be created.
   * Common in UI libraries, e.g., LayoutInflater.from(context).
3. **Builder**
   * Allows step-by-step construction of complex objects.
   * Used with AlertDialog.Builder, NotificationCompat.Builder, etc.
4. **Prototype (Rare in Android)**
   * Create object clones without depending on their concrete classes.

**🔁 Structural Design Patterns**

These deal with object composition.

1. **Adapter**
   * Converts one interface to another.
   * Used extensively in RecyclerView.Adapter, or to connect old code with new.
2. **Decorator**
   * Adds responsibilities to objects dynamically.
   * Used in UI when wrapping views (e.g., using ContextWrapper).
3. **Facade**
   * Provides a unified interface to a set of interfaces.
   * Example: MediaPlayer class hides complex implementation details.
4. **Proxy**
   * A surrogate or placeholder for another object.
   * Example: Retrofit creates dynamic proxies for API interfaces.

**🔄 Behavioral Design Patterns**

These handle communication between objects.

1. **Observer**
   * One-to-many dependency. Used in LiveData, RxJava, Flow.
   * LiveData.observe() reacts to data changes.
2. **Strategy**
   * Enables selecting an algorithm at runtime.
   * Useful in implementing different image-loading strategies (e.g., Glide vs Picasso).
3. **Command**
   * Encapsulates a request as an object.
   * Example: View.OnClickListener acts like a command pattern.
4. **Mediator**
   * Central authority to communicate between objects.
   * ViewModel in MVVM acts like a mediator between UI and data layers.
5. **Memento**
   * Capture and restore object state.
   * Example: onSaveInstanceState(Bundle).
6. **State**
   * Allows an object to alter its behavior when its internal state changes.
   * Example: Media Player states (playing, paused, stopped).

**🏛️ Architectural Patterns**

These are higher-level patterns specific to organizing codebases.

1. **MVC (Model-View-Controller)**
   * Not commonly recommended in Android.
2. **MVP (Model-View-Presenter)**
   * UI logic in Presenter, used in legacy apps.
3. **MVVM (Model-View-ViewModel)**
   * Most common in modern Android (Jetpack components).
   * ViewModel + LiveData/StateFlow.
4. **MVI (Model-View-Intent)**
   * Unidirectional data flow. Popular with Kotlin and Coroutines.
5. **Clean Architecture**
   * Layered architecture (Presentation, Domain, Data).
   * Encouraged for testable, scalable apps.

**🧰 Other Patterns / Practices in Android**

* **Repository Pattern**: Abstracts access to multiple data sources.
* **Use Case / Interactor Pattern**: Represents a single business logic operation.
* **Service Locator** (anti-pattern if misused): Alternative to dependency injection.
* **Dependency Injection**: Via Hilt, Dagger, or Koin.

Would you like a diagram or categorized cheat sheet of these patterns for Android specifically?