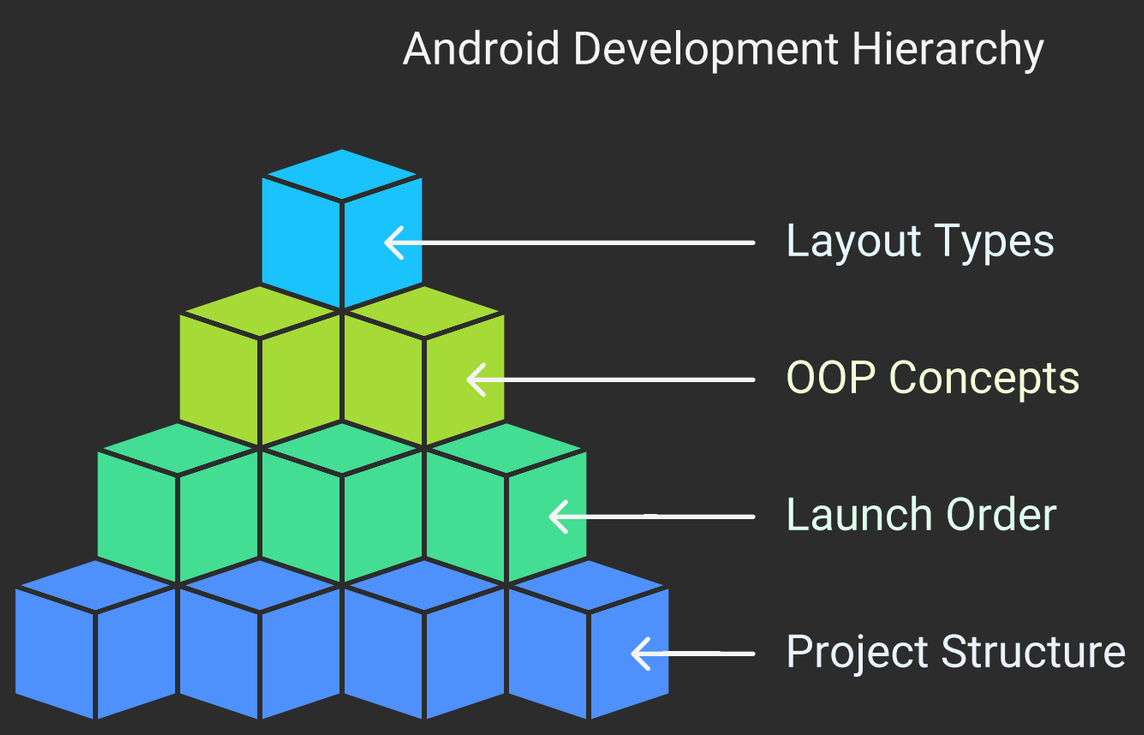
Lecture #2 – Android Project Structure, Layouts & Views

Abstract

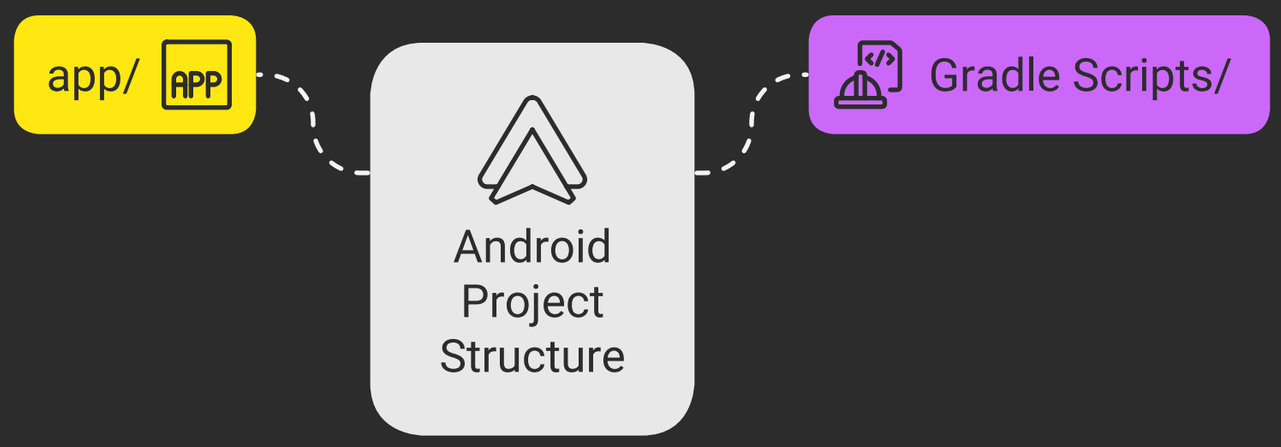
This document provides an overview of the Android project structure, focusing on the organization of files and directories, the launch order of components, and the fundamental concepts of Object-Oriented Programming (OOP) as applied in Android development. It delves into the various types of layouts available in Android, explaining their characteristics and use cases in detail. This information is essential for developers looking to create efficient and well-structured Android applications.



- Android Project Hierarchy (with Purpose)

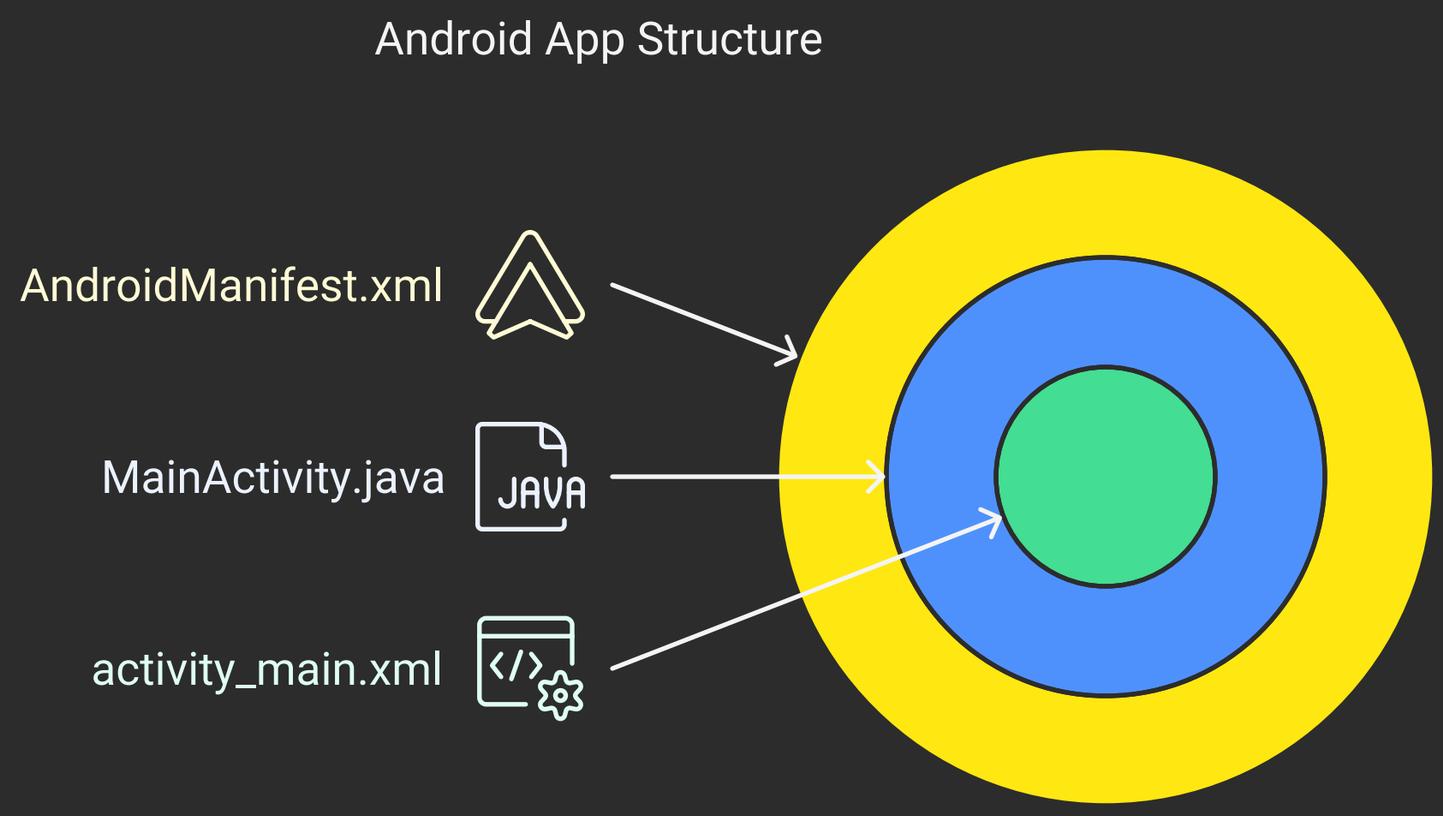
```
📅 Lec2/
 ├──📂 app/                          # Core Android app folder
 │  ├──📂 manifests/                # Contains manifest file for app configuration
 │  │  └──📄 AndroidManifest.xml     # Declares app components, permissions, and app-level metadata
 │  ├──📂 java/                     # Contains all Java source code
 │  │  ├── com.example.lecture2/     # Main package; contains MainActivity (app logic)
 │  │  ├── com.example.lecture2.test/    # Unit test source folder
 │  │  └── com.example.lecture2.androidtest/ # Instrumented (UI) test source folder
 │  └──📂 res/                      # All app resources (UI related)
 │     ├── drawable/         # Images and graphic assets
 │     ├── layout/           # XML layout files for activities and fragments
 │     ├── mipmap/           # App launcher icons in different sizes
 │     ├── values/           # Resource values (strings, colors, styles)
 │     └── xml/              # Other XML configurations
 └──📂 Gradle Scripts/              # Build system configurations
       ├── build.gradle (project-level)   # Config for all modules
```

Android Project Structure and Components



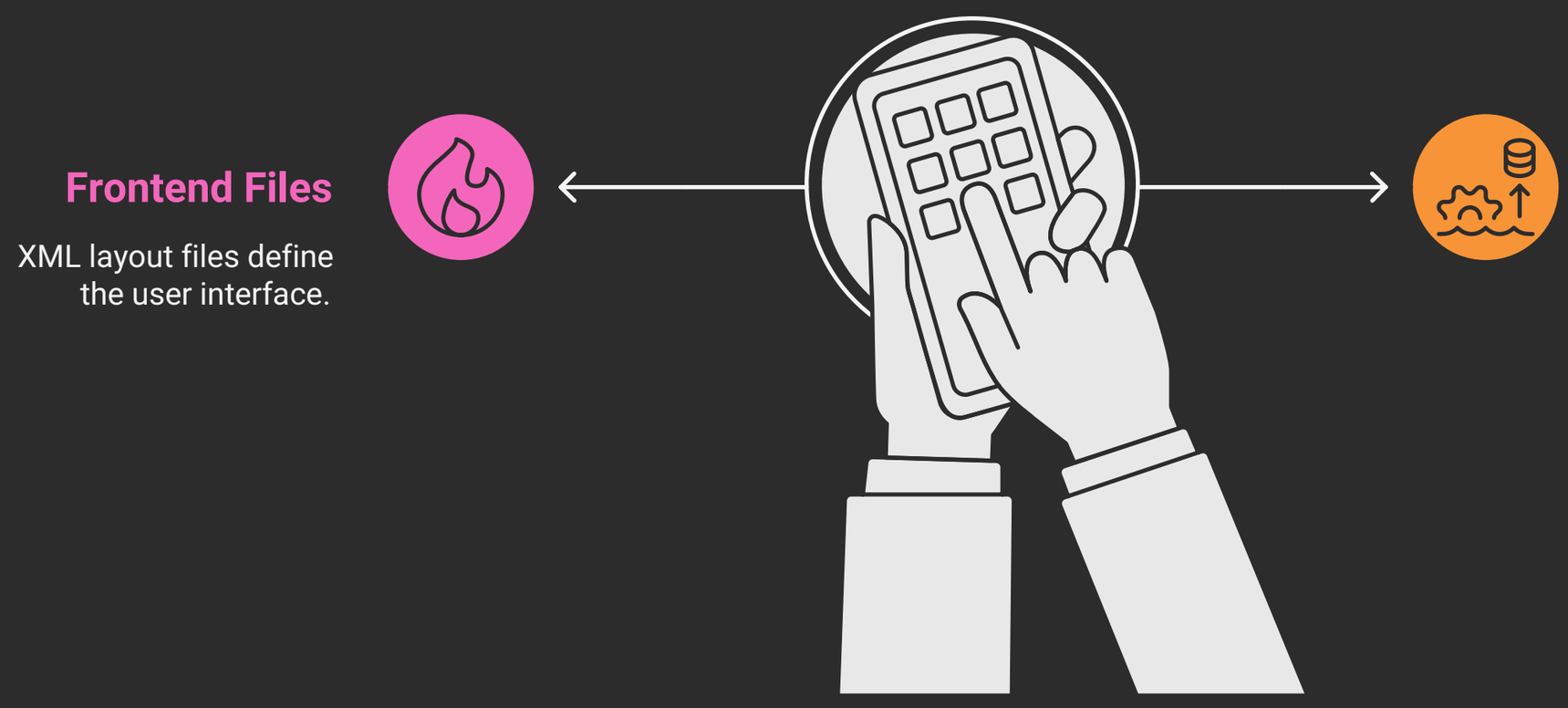
📅 App Launch Order (Frontend & Backend Roles)

1. **AndroidManifest.xml** - Loads first; registers app components.
2. **MainActivity.java** - Acts as backend logic; executes onCreate()
3. **activity_main.xml** - UI loaded through **setContentView()**



Frontend Files: XML layout files [**activity_main.xml**, etc.]

Backend Files: Java files [**MainActivity.java**, etc.] , through Java code we make logic so that user interacts with the front end of the app.



 Java Packages in Android

- Used to organize classes and prevent naming conflicts.
- Main code logic goes into **com.example.lecture2**

 Inheritance in Java (Android)

- Used to **reuse and extend** functionality.
- **MainActivity extends AppCompatActivity** gives access to lifecycle and support features.

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

 Android Views and Layouts

View: A UI element (e.g., TextView, Button)

Layouts: Containers that organize Views.

1. LinearLayout

- Arranges children in a **single direction** (vertical or horizontal).
- Use the **orientation** attribute to specify the direction.
- **Use Case**: Ideal for simple layouts where elements need to be stacked either vertically or horizontally.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <TextView android:layout_width="wrap_content"
android:layout_height="wrap_content" android:text="Item 1"/>
    <TextView android:layout_width="wrap_content"
android:layout_height="wrap_content" android:text="Item 2"/>
</LinearLayout>
```

2. RelativeLayout

- Positions children **relative to each other** or the parent.
- More flexible than LinearLayout, allowing overlapping views and complex arrangements.
- **Use Case**: Suitable for layouts where elements need to be positioned in relation to one another.

```
<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TextView
        android:id="@+id/text1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Text 1"/>
    <TextView
        android:id="@+id/text2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Text 2"
        android:layout_below="@id/text1"/>
</RelativeLayout>
```

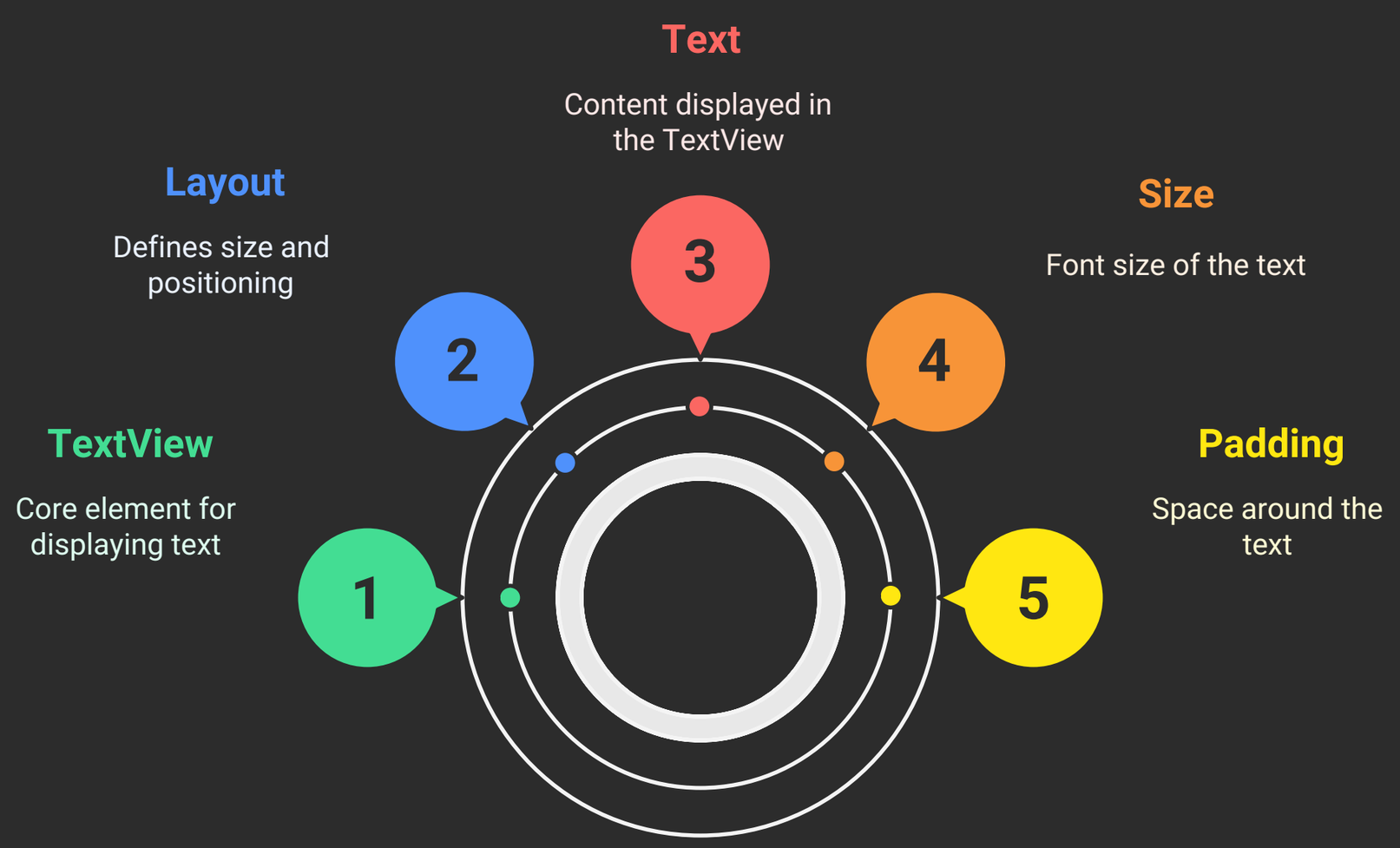
3. ConstraintLayout

- Allows **responsive design** using constraints, enabling complex layouts without nesting.
- Most efficient and widely used in modern Android apps.
- **Use Case**: Best for complex layouts that require a responsive design across different screen sizes.

```
<androidx.constraintlayout.widget.ConstraintLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">
    <TextView
        android:id="@+id/text1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Text 1"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>
    <TextView
        android:id="@+id/text2"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:text="Text 2"
        app:layout_constraintTop_toBottomOf="@id/text1"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

TextView Element

```
<TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    android:textSize="16sp"
    android:padding="8dp" />
```



Attributes:

- **wrap_content** / **match_parent**: Defines view size.
- **sp**: Used for text (scales with user settings).
- **dp**: Used for padding/margins (device-independent).

This structure prepares your GitHub repo to be readable, educational, and useful for Napkin AI or documentation later!