

UNIT 3

Designing the User Interface

(5 Hours) — Mobile Programming (CAC351)

3.1 Android Layout Types

A **layout** is a ViewGroup that defines how child views are arranged on screen. Layouts are declared in XML (res/layout) or created in code. The choice of layout affects performance: deeply nested layouts are slow, so flatter hierarchies are preferred.

Layout	How it arranges children	Typical use
LinearLayout	In a single row or column (android:orientation); supports layout_weight	Forms, simple lists of widgets
RelativeLayout	Each child positioned relative to siblings or parent (below, toRightOf, alignParentBottom)	Overlapping or aligned items
TableLayout	Rows (TableRow) and columns like an HTML table	Calculator, data entry grids
AbsoluteLayout	Exact x, y coordinates. Deprecated – does not adapt to screen sizes	Avoid in real apps
ConstraintLayout	Views connected by constraints to parent/siblings/guidelines; flat hierarchy	Default in modern Android Studio; complex responsive UIs
FrameLayout	Stacks children on top of each other	Fragment container, overlays

LinearLayout

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal">
    <Button android:layout_width="0dp" android:layout_height="wrap_content"
        android:layout_weight="1" android:text="OK" />
    <Button android:layout_width="0dp" android:layout_height="wrap_content"
        android:layout_weight="1" android:text="Cancel" />
</LinearLayout>
```

Listing 3.1: Two buttons sharing the width equally using layout_weight

RelativeLayout

```

<RelativeLayout ... >
    <TextView android:id="@+id/title" android:text="Title"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_centerHorizontal="true" />
    <Button android:id="@+id/btn" android:text="Submit"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        android:layout_below="@id/title"
        android:layout_alignParentEnd="true" />
</RelativeLayout>

```

Listing 3.2: RelativeLayout positions

TableLayout

```

<TableLayout android:layout_width="match_parent" android:layout_height="wrap_content"
    android:stretchColumns="1">
    <TableRow>
        <TextView android:text="Name:" />
        <EditText android:hint="Enter name" />
    </TableRow>
    <TableRow>
        <TextView android:text="Email:" />
        <EditText android:hint="Enter email" />
    </TableRow>
</TableLayout>

```

Listing 3.3: TableLayout with two rows

ConstraintLayout

Every view needs at least one **horizontal** and one **vertical** constraint. Constraints connect a side of a view to another view's side or to the parent, using attributes named *layout_constraint[MySide]_to[TargetSide]Of*.

```

<androidx.constraintlayout.widget.ConstraintLayout ... >
    <TextView android:id="@+id/label" android:text="Hello"
        android:layout_width="wrap_content" android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />
    <Button android:id="@+id/go" android:text="Go"
        android:layout_width="0dp" android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@id/label"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

Listing 3.4: ConstraintLayout (0dp = match constraints)

Note: AbsoluteLayout was deprecated in API level 3; do not use it in new projects.

3.2 Layout Attributes

Attribute	Values / Meaning
layout_width, layout_height	match_parent (fill parent), wrap_content (fit content), or fixed dp

Attribute	Values / Meaning
layout_margin	Space outside the view; also marginTop/Start/End/Bottom
padding	Space between view's border and its content
gravity	Alignment of content <i>inside</i> the view (center, start, end, top, bottom)
layout_gravity	Alignment of the view <i>inside its parent</i>
layout_weight	Share of extra space in LinearLayout
visibility	visible, invisible (keeps space), gone (removes space)
background	Colour (@color/...) or drawable

Note: Do not confuse gravity (content of the view) with layout_gravity (position of the view in its parent).

3.3 Android Widgets and Attributes

TextView and EditText

TextView displays read-only text. **EditText** is an editable text field. Useful attributes: *text*, *textSize*, *textColor*, *textStyle*, *hint*, *inputType* (text, number, textPassword, textEmailAddress, phone), *maxLength*, *lines*, *ems*.

```
<EditText android:id="@+id/etEmail"
    android:layout_width="match_parent" android:layout_height="wrap_content"
    android:hint="Email address" android:inputType="textEmailAddress" />
```

Listing 3.5: EditText for email

Button, CheckBox, RadioButton, Spinner

Widget	Purpose	Key methods
Button / ImageButton	Triggers an action on click	setOnClickListener()
CheckBox	Independent on/off option; many can be selected	isChecked(), setOnCheckedChangeListener()
RadioButton + RadioGroup	One choice from a group	getCheckedRadioButtonId()
Spinner	Drop-down list; one item selected	setAdapter(), getItem(), setOnItemSelectedListener()
ToggleButton / Switch	Two-state on/off control	isChecked()
ImageView	Displays an image (src, scaleType)	setImageResource()
ProgressBar / SeekBar / RatingBar	Progress, slider, star rating	setProgress(), getRating()

Widget	Purpose	Key methods
DatePicker / TimePicker	Choose date or time	getYear(), getHour()

```

<CheckBox android:id="@+id/cbJava" android:text="Java"
    android:layout_width="wrap_content" android:layout_height="wrap_content" />

<RadioGroup android:id="@+id/rgGender" android:orientation="horizontal"
    android:layout_width="wrap_content" android:layout_height="wrap_content">
    <RadioButton android:id="@+id/rbMale" android:text="Male" />
    <RadioButton android:id="@+id/rbFemale" android:text="Female" />
</RadioGroup>

<Spinner android:id="@+id/spCity"
    android:layout_width="match_parent" android:layout_height="wrap_content"
    android:entries="@array/city_array" />

```

Listing 3.6: CheckBox, RadioGroup and Spinner in XML

3.4 Event Handling

An **event** is a user action such as a click, long-click, touch or text change. Android uses the **event listener** model: a listener interface (e.g., *View.OnClickListener*) is registered on a view, and its callback runs when the event occurs.

Listener	Callback	Triggered by
View.OnClickListener	onClick(View v)	Tap on view
View.OnLongClickListener	onLongClick(View v)	Press and hold
View.OnTouchListener	onTouch(View v, MotionEvent e)	Touch/move/release
CompoundButton.OnCheckedChangeListener	onCheckedChanged(button, boolean)	CheckBox/Switch toggled
AdapterView.OnItemSelectedListener	onItemSelected(...)	Spinner item chosen
TextWatcher	onTextChanged(...)	EditText text edited

Three ways to handle a click

```

// 1. Anonymous inner class
btn.setOnClickListener(new View.OnClickListener() {
    @Override public void onClick(View v) { show("Clicked"); }
});

// 2. Lambda (Java 8+)
btn.setOnClickListener(v -> show("Clicked"));

// 3. XML attribute: android:onClick="handleClick"
public void handleClick(View view) { show("Clicked"); }

```

Listing 3.7: Event handling styles

Example 1 — Simple registration form

Create a form with EditText (name), RadioGroup (gender), CheckBox (hobbies), Spinner (city) and a Submit button that shows a Toast with the entered values.

Solution (Java)

Get references with *findViewById()*, register a click listener on the button, read the values, and show a Toast.

```
submit.setOnClickListener(v -> {
    String name = etName.getText().toString().trim();
    if (name.isEmpty()) { etName.setError("Name required"); return; }

    int id = rgGender.getCheckedRadioButtonId();
    String gender = (id == R.id.rbMale) ? "Male" : "Female";
    String hobby = cbMusic.isChecked() ? "Music" : "None";
    String city = spCity.getSelectedItem().toString();

    Toast.makeText(this, name + ", " + gender + ", " + hobby + ", " + city,
        Toast.LENGTH_LONG).show();
});
```

Listing 3.8: Reading widget values

3.5 Working with Strings, String Arrays and Colors

```
<!-- res/values/strings.xml -->
<string name="welcome">Welcome to MyApp</string>
<string-array name="city_array">
    <item>Kathmandu</item>
    <item>Pokhara</item>
    <item>Biratnagar</item>
</string-array>

<!-- res/values/colors.xml -->
<color name="primary">#1F618D</color>
<color name="accent">#E67E22</color>
```

Listing 3.9: strings.xml and colors.xml

- In XML: `@string/welcome`, `@array/city_array`, `@color/primary`.
- In Java: `getString(R.string.welcome)`, `getStringArray(R.array.city_array)`, `ContextCompat.getColor(this, R.color.primary)`.
- Colours use hex format `#RRGGBB` or `#AARRGGBB` (with alpha/transparency).
- Localisation: create `values-ne/strings.xml` for Nepali translations; Android picks the file matching the device language.

3.6 Working with Resources and Drawables

Resources are non-code assets stored under *res/*. Android chooses the best match automatically using **qualifiers** (e.g., *drawable-hdpi*, *layout-land*, *values-night*).

Folder	Content
layout/	UI XML files (layout-land/ for landscape)
values/	strings, colors, dimens, styles/themes, arrays
drawable/	Images (PNG, JPG, WebP), vector drawables, shape and selector XML
mipmap/	Launcher icons at multiple densities
menu/, anim/, raw/, xml/	Menus, animations, raw files, misc XML config

Drawables can be bitmap images or XML: *shape* (rectangles with corners and gradients), *selector* (different look per state such as pressed) and *vector* (scalable icons).

```
<!-- res/drawable/round_button.xml -->
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle">
    <solid android:color="@color/primary" />
    <corners android:radius="12dp" />
</shape>

<!-- use it -->
<Button android:background="@drawable/round_button" ... />
```

Listing 3.10: A shape drawable

Note: Resource file names must be lowercase letters, digits and underscores only (e.g., ic_home.png), and must not start with a digit.

3.7 Adding an Icon to the Project

- Right-click **res** → **New** → **Image Asset** (Image Asset Studio).
- Choose *Launcher Icons (Adaptive and Legacy)*; select your image, clipart or text as foreground and set the background colour.
- Android Studio generates icons for all densities (mdpi to xxxhdpi) in the *mipmap* folders.
- Confirm the icon in AndroidManifest.xml: *android:icon="@mipmap/ic_launcher"* and *android:roundIcon="@mipmap/ic_launcher_round"*.
- For in-app icons, use **New** → **Vector Asset** and choose a Material icon.

Key Takeaways

- ✓ Layouts: Linear (row/column), Relative, Table, Absolute (deprecated), Constraint (recommended), Frame.
- ✓ Know the difference: gravity vs layout_gravity, match_parent vs wrap_content, dp vs sp.
- ✓ Widgets: TextView, EditText, Button, CheckBox, RadioButton, Spinner, ImageView etc., each configured by XML attributes.
- ✓ Event handling uses listeners such as setOnClickListener; lambdas keep code short.
- ✓ Keep text, colours and arrays in resource files; use drawables and Image Asset Studio for images and icons.

— *End of Unit 3* —

bcalink.com