Mobile Application Development

Using Android

Module 2: User Interface Design & Layout Management

Comprehensive Study Material

1. UI Components Overview

1.1 Android UI Hierarchy

Android's UI is built on a hierarchy of View and ViewGroup objects. Every UI component is either a View (leaf node) or a ViewGroup (container for other views).

1.2 View vs ViewGroup

Aspect	View	ViewGroup
Definition	Individual UI component	Container for other Views/ViewGroups
Rendering	Renders itself on screen	Manages layout of child views
Events	Handles user interaction	Passes events to children
Examples	TextView, Button, ImageView	LinearLayout, RelativeLayout, ConstraintLayout
Base class	Base class for all widgets	Base class for all layouts

1.3 Common UI Attributes

```
android:layout_width="match_parent"    // Fill parent
android:layout_width="wrap_content"    // Size to content
android:layout_margin="16dp"           // Space outside view
android:padding="16dp"                 // Space inside view
android:visibility="visible"           // Visible
android:visibility="invisible"         // Hidden but occupies space
android:visibility="gone"              // Hidden and no space
android:gravity="center"               // Content alignment
android:layout_gravity="center"        // View alignment within parent
```

2. TextView

2.1 Overview

TextView displays text to the user. It is the most fundamental UI component in Android development.

2.2 Common XML Attributes

Attribute	Description	Example
android:text	Text to display	"Hello"
android:textSize	Font size (use sp)	"24sp"
android:textColor	Text color	#FF0000
android:textStyle	bold, italic, normal	"bold|italic"

android:fontFamily	Font typeface	"sans-serif"
android:gravity	Text alignment	"center"
android:maxLines	Maximum lines shown	"2"
android:ellipsize	Truncation style	"end"
android:shadowColor	Shadow color	#000000
android:autoLink	Auto-detect links	"web|email"

2.3 Programmatic Usage

```
TextView textView = findViewById(R.id.myTextView);
textView.setText("New Text");
textView.setTextColor(Color.RED);
textView.setTextSize(TypedValue.COMPLEX_UNIT_SP, 24);
textView.setTypeface(Typeface.DEFAULT_BOLD);
textView.setVisibility(View.VISIBLE);   // VISIBLE / INVISIBLE / GONE

// Click listener
textView.setOnClickListener(v -> {
    Toast.makeText(this, "Clicked!", Toast.LENGTH_SHORT).show();
});
```

3. EditText

3.1 Overview

EditText is a user input field that allows text entry and editing. It extends TextView with additional text-editing functionality.

3.2 Input Types

Input Type	Description
text	Plain text
textPersonName	Person name
textEmailAddress	Email address
textPassword	Password (hidden)
textMultiLine	Multi-line text
number	Numeric only
numberDecimal	Decimal numbers
phone	Phone number
date	Date
time	Time

3.3 IME Options (Keyboard Actions)

Option	Description
actionGo	"Go" button
actionSearch	"Search" button
actionSend	"Send" button
actionNext	"Next" button — moves focus to next field
actionDone	"Done" button — closes keyboard
actionPrevious	"Previous" button

3.4 Key Programmatic Methods

```
String text = etUsername.getText().toString();  // Read text
etUsername.setText("New value");                // Set text
etUsername.setError("Required");                // Show error
etUsername.setError(null);                       // Clear error
etUsername.requestFocus();                       // Gain focus
etUsername.clearFocus();                         // Lose focus

// TextWatcher for real-time changes
etUsername.addTextChangedListener(new TextWatcher() {
    @Override public void afterTextChanged(Editable s) {
        tvResult.setText("Length: " + s.length());
    }
    // ... beforeTextChanged, onTextChanged
});
```

4. Button

4.1 Button Types

Type	Description
Button	Standard button
AppCompatButton	Material Design compatible button
MaterialButton	Material Components button (recommended)
ImageButton	Button with image instead of text
ToggleButton	On/off toggle button
CompoundButton	Checkable button (CheckBox, RadioButton)

4.2 Click Listener Methods

```
// Method 1: Anonymous class
btnSubmit.setOnClickListener(new View.OnClickListener() {
    @Override public void onClick(View v) { handleSubmit(); }
```

```
});

// Method 2: Lambda (Java 8+)
btnCancel.setOnClickListener(v -> handleCancel());

// Method 3: Activity implements View.OnClickListener
public class MainActivity extends AppCompatActivity
    implements View.OnClickListener {
    @Override public void onClick(View v) {
        if (v.getId() == R.id.btnSubmit) { /* handle */ }
    }
}
```

5. ImageView

5.1 Scale Types

Scale Type	Description
center	Center image without scaling
centerCrop	Scale to fill, crop excess (most common for profile pics)
centerInside	Scale to fit inside, maintain aspect ratio
fitCenter	Scale to fit center, maintain aspect ratio
fitXY	Scale to fill, stretch (distorts aspect)
matrix	Scale using matrix transformation

5.2 Setting Images Programmatically

```
ivProfile.setImageResource(R.drawable.profile_image);
ivProfile.setImageBitmap(bitmap);
ivProfile.setImageDrawable(drawable);
ivProfile.setScaleType(ImageView.ScaleType.CENTER_CROP);
ivProfile.setColorFilter(Color.BLUE, PorterDuff.Mode.SRC_IN);
ivProfile.setAlpha(0.5f);

// Glide (external library — recommended for URLs)
Glide.with(this).load(imageUrl)
     .placeholder(R.drawable.placeholder)
     .circleCrop().into(ivProfile);
```

6. CheckBox

6.1 Overview

CheckBox allows users to select one or more options independently from a set. Each checkbox operates independently.

6.2 Key Methods

```
boolean checked = cbAgree.isChecked();   // Read state
cbAgree.setChecked(true);                // Set state
cbAgree.toggle();                        // Toggle state

cbAgree.setOnCheckedChangeListener((buttonView, isChecked) -> {
    btnSubmit.setEnabled(isChecked);
});
```

7. RadioButton & RadioGroup

7.1 Overview

RadioButton allows selection of a single option from a group. RadioGroup is required to group RadioButtons so only one can be selected at a time.

7.2 XML Structure

```
<RadioGroup android:id="@+id/rgGender"
            android:orientation="vertical">
    <RadioButton android:id="@+id/rbMale"   android:text="Male" />
    <RadioButton android:id="@+id/rbFemale" android:text="Female" />
    <RadioButton android:id="@+id/rbOther"  android:text="Other" />
</RadioGroup>
```

7.3 Key Methods

```
int selectedId = rgGender.getCheckedRadioButtonId();  // Get selected ID
RadioButton rb = findViewById(selectedId);             // Get the button
String text = rb.getText().toString();                 // Get its text
rgGender.check(R.id.rbFemale);                        // Set selection
rgGender.clearCheck();                                 // Clear selection

rgGender.setOnCheckedChangeListener((group, checkedId) -> {
    RadioButton selected = findViewById(checkedId);
    Toast.makeText(this, selected.getText(), Toast.LENGTH_SHORT).show();
});
```

8. Spinner

8.1 Overview

Spinner provides a drop-down list for selecting one item from multiple options. It is Android's equivalent of an HTML select element.

8.2 Basic Setup

```
// From array resource
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
    this, R.array.countries_array, android.R.layout.simple_spinner_item);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinner.setAdapter(adapter);
```

```
// From a List
List<String> items = Arrays.asList("Java", "Kotlin", "C++");
ArrayAdapter<String> adapter2 = new ArrayAdapter<>(this,
    android.R.layout.simple_spinner_item, items);
spinner.setAdapter(adapter2);

// Selection listener
spinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override public void onItemSelected(AdapterView<?> parent,
            View view, int position, long id) {
        String selected = parent.getItemAtPosition(position).toString();
    }
    @Override public void onNothingSelected(AdapterView<?> parent) {}
});
```

9. Layouts Overview

Layout Type	Description	Best For
LinearLayout	Arranges children linearly (horizontal or vertical)	Simple linear structures
RelativeLayout	Positions children relative to parent or each other	Complex layouts with relative positioning
ConstraintLayout	Flexible layout with constraints (Google recommended)	Complex, flat layouts — best performance
FrameLayout	Single child view; stacks children on top of each other	Fragment containers, overlays
TableLayout	Arranges children in rows and columns	Tabular data display
GridLayout	Arranges children in a grid	Grid patterns
CoordinatorLayout	Advanced layout for Material Design scrolling	Complex scrolling behavior

Layout Optimization Tips

- Use ConstraintLayout for complex layouts to keep the hierarchy flat
- Avoid deeply nested layouts — they hurt rendering performance
- Use <merge> to reduce view hierarchy
- Use <include> to reuse layouts across screens
- Use ViewStub for views that may not always be needed (lazy loading)

10. LinearLayout

10.1 Overview

LinearLayout arranges its children in a single column (vertical) or row (horizontal). Child views are stacked one after another in the specified direction.

10.2 layout_weight

layout_weight distributes extra space proportionally among children. Set layout_width (or height) to 0dp for weighted distribution.

```
<!-- Equal split (50/50) -->
<LinearLayout android:orientation="horizontal">
    <Button android:layout_width="0dp" android:layout_weight="1"
android:text="Left" />
    <Button android:layout_width="0dp" android:layout_weight="1"
android:text="Right" />
</LinearLayout>

<!-- Weighted distribution (1/3 and 2/3) -->
<Button android:layout_width="0dp" android:layout_weight="1" android:text="1/3" />
<Button android:layout_width="0dp" android:layout_weight="2" android:text="2/3" />
```

10.3 gravity vs layout_gravity

Attribute	Description	Example
android:gravity	Aligns content INSIDE the view	Text centered in a TextView
android:layout_gravity	Aligns the view WITHIN its parent	Button aligned to the right side

11. RelativeLayout

11.1 Overview

RelativeLayout positions children relative to the parent container or to other sibling views using declarative positioning rules.

11.2 Positioning Relative to Parent

Attribute	Description
layout_alignParentTop	Align to top of parent
layout_alignParentBottom	Align to bottom of parent
layout_alignParentLeft / Start	Align to left/start of parent
layout_alignParentRight / End	Align to right/end of parent
layout_centerHorizontal	Center horizontally in parent
layout_centerVertical	Center vertically in parent
layout_centerInParent	Center both horizontally and vertically

11.3 Positioning Relative to Other Views

Attribute	Description

layout_above	Place this view above the specified view
layout_below	Place this view below the specified view
layout_toLeftOf / toStartOf	Place this view to the left of the specified view
layout_toRightOf / toEndOf	Place this view to the right of the specified view
layout_alignTop	Align top edge with top edge of specified view
layout_alignBottom	Align bottom edge with bottom of specified view
layout_alignBaseline	Align text baseline with specified view

12. ConstraintLayout

12.1 Overview & Benefits

ConstraintLayout is a flexible and highly efficient layout that allows you to create complex UI without nested view groups. It is the Google-recommended layout for most screens.

- Flat view hierarchy — better rendering performance
- Flexible positioning using constraints between views and the parent
- Supports chains, guidelines, and barriers
- Visual drag-and-drop design in Android Studio

12.2 Basic Constraints

```
<!-- Center in parent -->
<TextView
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent" />

<!-- Relative positioning -->
<TextView android:id="@+id/tvFirst"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintStart_toStartOf="parent" />

<TextView android:id="@+id/tvSecond"
    app:layout_constraintStart_toEndOf="@id/tvFirst"
    app:layout_constraintTop_toTopOf="@id/tvFirst" />
```

12.3 Bias

Bias controls the distribution of space when a view is constrained on both sides. A value of 0.3 places the view 30% from the start.

```
app:layout_constraintHorizontal_bias="0.3"  // 30% from left
app:layout_constraintVertical_bias="0.7"    // 70% from top
```

12.4 Chains

Chains link a group of views horizontally or vertically. The first view in the chain controls the chain style.

Chain Style	Description
spread (default)	Distributes views with equal space between them
spread_inside	Distributes views but ends are flush with parent edges
packed	Views are packed together (centered)
packed + bias	Packed views with a bias position offset

12.5 Guidelines

Guidelines are invisible helper lines for aligning views at a fixed position or percentage.

```
<!-- Vertical guideline at 32dp from start -->
<androidx.constraintlayout.widget.Guideline
    android:orientation="vertical"
    app:layout_constraintGuide_begin="32dp" />

<!-- Horizontal guideline at 50% of the screen height -->
<androidx.constraintlayout.widget.Guideline
    android:orientation="horizontal"
    app:layout_constraintGuide_percent="0.5" />
```

12.6 Barriers

A Barrier groups multiple views and creates a virtual boundary aligned to the outermost edge of the group. Useful for aligning inputs after labels of varying widths.

```
<androidx.constraintlayout.widget.Barrier
    android:id="@+id/labelBarrier"
    app:barrierDirection="end"
    app:constraint_referenced_ids="tvLabel1,tvLabel2" />

<EditText
    app:layout_constraintStart_toEndOf="@id/labelBarrier"
    app:layout_constraintEnd_toEndOf="parent" />
```

13. FrameLayout

13.1 Overview

FrameLayout is a simple layout that stacks children on top of each other. It is most commonly used as a fragment container or to overlay views (e.g., a progress spinner on top of content).

```
<FrameLayout android:id="@+id/fragmentContainer"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- Fragment fills the container -->
    <Fragment ... />
```

```
    <!-- ProgressBar overlaid on top -->
    <ProgressBar
        android:layout_gravity="center"
        android:visibility="gone" />

</FrameLayout>
```

14. Toasts

14.1 Overview

Toast provides simple feedback about an operation in a small popup that automatically disappears. It does not require user interaction.

14.2 Usage

```
// Short (approx 2 seconds)
Toast.makeText(this, "Hello!", Toast.LENGTH_SHORT).show();

// Long (approx 3.5 seconds)
Toast.makeText(this, "Hello!", Toast.LENGTH_LONG).show();

// Custom position
Toast toast = Toast.makeText(this, "Centered", Toast.LENGTH_SHORT);
toast.setGravity(Gravity.CENTER, 0, 0);
toast.show();
```

15. Dialog Boxes

15.1 AlertDialog

AlertDialog displays a dialog with title, message, and up to three action buttons (Positive, Negative, Neutral).

```
new AlertDialog.Builder(this)
    .setTitle("Confirm")
    .setMessage("Are you sure?")
    .setPositiveButton("Yes", (dialog, which) -> { /* handle */ })
    .setNegativeButton("No", null)
    .setNeutralButton("Later", null)
    .show();

// With list items
String[] items = {"Item 1", "Item 2", "Item 3"};
new AlertDialog.Builder(this)
    .setTitle("Select")
    .setItems(items, (dialog, which) -> { /* handle position */ })
    .show();
```

15.2 Snackbar

Snackbar provides lightweight feedback at the bottom of the screen and supports an optional action button.

```
// Basic
```

```
Snackbar.make(findViewById(android.R.id.content),
    "Item deleted", Snackbar.LENGTH_LONG).show();

// With undo action
Snackbar.make(view, "Deleted", Snackbar.LENGTH_LONG)
    .setAction("Undo", v -> { /* restore */ })
    .show();
```

15.3 Toast vs Snackbar

Feature	Toast	Snackbar
Position	Custom (default bottom-center)	Bottom of screen
User action	None — auto-dismisses	Optional action button
Context needed	Application context OK	Requires a View
Interrupts UI	No	No
Recommended for	Simple info messages	Feedback with optional undo

16. Menus

16.1 Types of Menus

Menu Type	Description	How to Use
Options Menu	Action bar menu (3-dot overflow)	Override onCreateOptionsMenu() and onOptionsItemSelected()
Context Menu	Long-press menu on a view	registerForContextMenu(view) then onCreateContextMenu()
Popup Menu	Menu anchored to any view	Create PopupMenu(context, anchorView) and show()

16.2 Options Menu Example

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.main_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_search: /* handle search */ return true;
        case R.id.action_settings: /* handle settings */ return true;
        default: return super.onOptionsItemSelected(item);
    }
}
```

17. Key Concepts Summary

Component	Purpose	Key Attributes / Methods
TextView	Display text	text, textSize, textColor, gravity
EditText	User text input	hint, inputType, imeOptions, getText()
Button	Trigger an action	text, setOnClickListener()
ImageView	Display images	src, scaleType, setImageResource()
CheckBox	Multiple selection	checked, setOnCheckedChangeListener()
RadioButton	Single selection in a group	checked, inside RadioGroup
Spinner	Drop-down selection	setAdapter(), setOnItemSelectedListener()
Toast	Non-interactive feedback popup	makeText().show()
AlertDialog	Interactive dialog	Builder, setPositiveButton()
Snackbar	Bottom feedback with action	make().setAction().show()
LinearLayout	Stack views horizontally/vertically	orientation, layout_weight
RelativeLayout	Position by rules	layout_below, layout_toRightOf
ConstraintLayout	Flat, flexible layout	constraints, chains, guidelines
FrameLayout	Overlay / fragment container	layout_gravity on children

18. Interview Questions

- 1. What is the difference between match_parent and wrap_content?
- 2. Explain the different layout types in Android and when to use each.
- 3. What is the difference between gravity and layout_gravity?
- 4. How does layout_weight work in LinearLayout?
- 5. What are the advantages of ConstraintLayout over other layouts?
- 6. Explain the concept of chains in ConstraintLayout.
- 7. What is the difference between Toast and Snackbar?
- 8. How do you handle user input validation in EditText?
- 9. Explain the different input types in EditText and their purpose.
- 10. How do you create a custom AlertDialog with a custom view?

19. Gradle Dependencies

```
dependencies {
    // Material Design Components (buttons, text fields, etc.)
    implementation 'com.google.android.material:material:1.11.0'
```

```
    // ConstraintLayout
    implementation 'androidx.constraintlayout:constraintlayout:2.1.4'

    // CardView
    implementation 'androidx.cardview:cardview:1.0.0'

    // Glide (image loading)
    implementation 'com.github.bumptech.glide:glide:4.15.1'
}
```

End of Module 2 Study Material