**✅ Section 14: More Widgets in Android**

This section explains how to use common UI widgets in Android to collect user input and display states visually. Covered widgets:

* CheckBox
* RadioButton
* Spinner (Dropdown)
* TimePicker
* DatePicker
* ProgressBar

**🔹 1. CheckBox**

**🔑 Key Concept:**

* A CheckBox allows toggling between two states: **checked** and **unchecked**.
* Useful for multi-select input (e.g., selecting toppings, options).
* A binary UI widget for toggling between checked/unchecked states.
* Ideal for optional selections (e.g., "Add extra cheese").
* Automatically handles visual state changes.

**✅ Steps to Implement:**

**XML:**

xml

CopyEdit

<CheckBox

android:id="@+id/checkbox"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

android:text="Pizza"

android:textSize="24sp"/>

**Java:**

java

CopyEdit

CheckBox checkbox;

checkbox = findViewById(R.id.checkbox);

checkbox.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener() {

@Override

public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {

if (isChecked) {

Toast.makeText(getApplicationContext(), "Checkbox is checked", Toast.LENGTH\_SHORT).show();

} else {

Toast.makeText(getApplicationContext(), "Checkbox is not checked", Toast.LENGTH\_SHORT).show();

}

}

});

**💡 Notes:**

* isChecked boolean indicates checkbox state.
* Set appropriate constraints in XML.

**APIs/Tools**:

* CheckBox class, setOnCheckedChangeListener, Toast.
* Attributes: android:text, android:textSize.

**Best Practices**:

* Use concise labels (e.g., "Enable notifications").
* Group related checkboxes (e.g., pizza toppings).
* **Alternative**: Switch for on/off settings.

**🔹 2. RadioButton + RadioGroup**

**🔑 Key Concept:**

* RadioButtons are used for **mutually exclusive choices**.
* RadioGroup ensures only one button is selected at a time.
* Mutually exclusive selection (only one option chosen at a time).
* Must be grouped inside a RadioGroup.
* Common in forms, quizzes, or settings.

**✅ Steps to Implement:**

**XML:**

xml

CopyEdit

<RadioGroup

android:id="@+id/radioGroup"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<RadioButton

android:id="@+id/radioButton1"

android:text="Tomato" />

<RadioButton

android:id="@+id/radioButton2"

android:text="Cheese" />

</RadioGroup>

**Java:**

java

CopyEdit

RadioGroup radioGroup = findViewById(R.id.radioGroup);

radioGroup.setOnCheckedChangeListener(new RadioGroup.OnCheckedChangeListener() {

@Override

public void onCheckedChanged(RadioGroup group, int checkedId) {

RadioButton radioButton = findViewById(checkedId);

Toast.makeText(getApplicationContext(), "You selected " + radioButton.getText(), Toast.LENGTH\_SHORT).show();

}

});

**💡 Notes:**

* checkedId gives the selected RadioButton's ID.
* Don’t hardcode radioButton1, radioButton2; use checkedId.

**APIs/Tools**:

* RadioGroup, RadioButton, setOnCheckedChangeListener.

**Best Practices**:

* Always wrap radio buttons in a RadioGroup.
* Prefer horizontal layouts for 2–3 options; vertical for more.
* **Alternative**: Spinner for space-constrained UIs.

**🔹 3. Spinner (Dropdown Menu)**

**🔑 Key Concept:**

* Spinner shows a **dropdown list** for selecting one option.
* Similar to a RadioGroup but more compact.
* Dropdown menu for single-item selection from a list.
* Requires an Adapter to populate data.
* Useful for forms (e.g., country selection).

**✅ Steps to Implement:**

**XML:**

xml

CopyEdit

<Spinner

android:id="@+id/spinner"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

**Java:**

java

CopyEdit

Spinner spinner = findViewById(R.id.spinner);

String[] courses = {"C++", "Java", "Kotlin", "Python"};

ArrayAdapter<String> adapter = new ArrayAdapter<>(this,

android.R.layout.simple\_spinner\_dropdown\_item, courses);

spinner.setAdapter(adapter);

**💡 Notes:**

* Adapter acts as a bridge between data and UI.
* You can customize the item layout using a custom XML layout.

**APIs/Tools**:

* Spinner, ArrayAdapter.
* Built-in layouts: android.R.layout.simple\_spinner\_dropdown\_item.
* **Context**: Represents app environment (used in ArrayAdapter constructor).

**Best Practices**:

* Use androidx.appcompat.widget.AppCompatSpinner for backward compatibility.
* Preload default selection (e.g., spinner.setSelection(0)).
* **Alternative**: RecyclerView for complex lists.

**🔹 4. TimePicker**

**🔑 Key Concept:**

* TimePicker allows users to select a **specific time** (hour & minute).
* Often used in scheduling, reminders, alarms.
* Supports 12-hour or 24-hour formats.

**✅ Steps to Implement:**

**XML:**

xml

CopyEdit

<TimePicker

android:id="@+id/timePicker"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

**Java:**

java

CopyEdit

TimePicker timePicker = findViewById(R.id.timePicker);

timePicker.setOnTimeChangedListener(new TimePicker.OnTimeChangedListener() {

@Override

public void onTimeChanged(TimePicker view, int hourOfDay, int minute) {

Toast.makeText(getApplicationContext(), "Hour: " + hourOfDay + " Minute: " + minute, Toast.LENGTH\_SHORT).show();

}

});

**💡 Notes:**

* Values are in 24-hour format by default.
* Can be switched to 12-hour via XML: android:is24HourView="false".

**APIs/Tools**:

* TimePicker, setOnTimeChangedListener.

**Best Practices**:

* Use TimePickerDialog for a pop-up version.
* Validate time ranges (e.g., disallow past times).

**🔹 5. DatePicker**

**🔑 Key Concept:**

* DatePicker lets users choose a **date** (day, month, year).
* Ideal for bookings, forms, calendars.
* Months are 0-indexed (January = 0).

**✅ Steps to Implement:**

**XML:**

xml

CopyEdit

<DatePicker

android:id="@+id/datePicker"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content" />

<Button

android:id="@+id/btnSubmit"

android:text="Submit" />

**Java:**

java

CopyEdit

DatePicker datePicker = findViewById(R.id.datePicker);

Button button = findViewById(R.id.btnSubmit);

button.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

int day = datePicker.getDayOfMonth();

int month = datePicker.getMonth() + 1; // 0-based index

int year = datePicker.getYear();

Toast.makeText(getApplicationContext(),

"Day: " + day + " Month: " + month + " Year: " + year,

Toast.LENGTH\_SHORT).show();

}

});

**💡 Notes:**

* Always **add 1 to month** because it starts from 0 (January = 0).
* Use DatePickerDialog if you want a popup instead of inline view.

**APIs/Tools**:

* DatePicker, getDayOfMonth(), getMonth(), getYear().

**Best Practices**:

* Use DatePickerDialog for better UX.
* Always add +1 to month for display.
* Validate dates (e.g., future-only).

**🔹 6. ProgressBar**

**🔑 Key Concept:**

* ProgressBar visually indicates **task progress (e.g., file download)**.
* Two types: **circular (indeterminate)** and **horizontal (determinate)**.

**✅ Steps to Implement:**

**XML:**

xml

CopyEdit

<ProgressBar

android:id="@+id/progressBar"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content"

style="?android:attr/progressBarStyleHorizontal" />

<Button

android:id="@+id/btnProgress"

android:text="Click Me" />

**Java:**

java

CopyEdit

ProgressBar progressBar = findViewById(R.id.progressBar);

Button button = findViewById(R.id.btnProgress);

int[] progress = {0};

button.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

progress[0] += 10;

progressBar.setProgress(progress[0]);

}

});

**💡 Notes:**

* Use progressBar.setIndeterminate(true) for tasks with unknown duration.
* For smooth UI, use Handler, Coroutine, or AsyncTask (deprecated) for background tasks.

**APIs/Tools**:

* ProgressBar, setProgress().
* Styles: Widget.ProgressBar.Horizontal.

**Best Practices**:

* Use indeterminate mode for unknown durations.
* Update progress from background threads (use runOnUiThread).
* **Alternative**: SwipeRefreshLayout for pull-to-refresh.

**🔧 Part B: Additional Concepts Not Covered but Important**

**🧠 1. Custom Adapters (for Spinners and Lists)**

* Use ArrayAdapter for simple string arrays.
* For complex layouts (e.g., items with images, descriptions), use:
  + BaseAdapter (for Spinner/ListView)
  + RecyclerView.Adapter (modern, recommended)
* ✅ *Best Practice*: Define a custom XML layout for each row/item and inflate it in getView().

**🦽 2. Accessibility**

* Add android:contentDescription for screen readers (especially for icons, images, custom views).
* Ensure proper touch target sizes and keyboard navigation.
* Tools:
  + Android Accessibility Scanner (Play Store or developer tools)

**✅ 3. Event Handling Best Practices**

* Prefer ViewBinding or DataBinding to avoid findViewById().
* For multiple widgets sharing the same listener (e.g., many buttons), use switch-case in onClick() or shared listener objects.

**💾 4. State Restoration & Persistence**

* Widgets like CheckBox, RadioButton, and Spinner **lose state on screen rotation**.
* Use:
  + onSaveInstanceState() / onRestoreInstanceState() to save and restore values.
  + Or better: **Jetpack ViewModel** to retain data across configuration changes.
* ✅ Also use SavedStateHandle with ViewModel if working with complex UI.

**🎨 5. Modern Alternatives (Material Design & Jetpack Libraries)**

| **Widget** | **✅ Recommended Modern Alternative** |
| --- | --- |
| Spinner | MaterialAutoCompleteTextView |
| DatePicker | MaterialDatePicker |
| TimePicker | MaterialTimePicker |
| ProgressBar | CircularProgressIndicator or Lottie animation |
| Button, CheckBox | MaterialButton, MaterialCheckBox (from Material library) |

📦 **Dependency** for Material Design:

groovy

CopyEdit

implementation 'com.google.android.material:material:<latest-version>'

**🧩 6. Dialogs for Pickers**

* Best UX: Use TimePickerDialog and DatePickerDialog instead of inline TimePicker or DatePicker.
* More compact and user-friendly, especially on small screens.

**🔐 7. Input Validation**

* Validate inputs before processing:
  + e.g., valid email format, date ranges, or required fields.
* Use TextInputLayout + setError() to show validation errors.
* Third-party validation libs: Android Validator, AwesomeValidation.

**🔄 8. ProgressBar with Async Tasks**

* Don’t update ProgressBar directly in main thread for long operations.
* Use:
  + AsyncTask (deprecated but still common in older code)
  + Kotlin Coroutines (recommended)
  + Java ExecutorService for background work
  + Combine with Handler or LiveData to update UI

**📦 9. ViewModel Integration**

* Use ViewModel to:
  + Store and observe data used in widgets (e.g., selected date, progress value).
  + Avoid data loss on screen rotation or fragment recreation.
* Pair with LiveData or StateFlow for observing changes.

**🛠 10. Jetpack Compose Counterparts**

For modern Android UI, **Jetpack Compose** is now the standard.

| **Classic Widget** | **Jetpack Compose Equivalent** |
| --- | --- |
| CheckBox | Checkbox() |
| RadioGroup | RadioButton() + State Mgmt |
| Spinner | DropdownMenu() |
| DatePicker | DatePickerDialog() |
| ProgressBar | LinearProgressIndicator(), etc. |

**✅ Benefits:**

* Declarative UI
* Built-in state management
* Cleaner, more scalable code

**🌐 11. Industry Trends**

* **Jetpack Compose**: Recommended for new Android projects.
* **Material You**: Dynamic theming (Android 12+), use Material3 library.
* **3rd-Party Libraries**:
  + [MaterialDateTimePicker](https://github.com/wdullaer/MaterialDateTimePicker) for advanced pickers.
  + Lottie for animated loading indicators.