Washtenaw Community College
ANN ARBOR, MICHIGAN

CPS251

Android Development

by Scott Shaper

Layout Basics

Think of layouts in Jetpack Compose as the invisible containers that organize your app's user interface. Just as you use different types of containers to organize items in your home—shelves for stacking books, drawers for side-by-side items, and boxes for layered objects—layouts help you arrange UI elements in your app.

In this lesson, we'll explore the three most common layout types in Compose: Column, Row, and Box. You'll learn how to use these layouts to create well-organized, visually appealing screens for your Android apps.

Quick Reference Table

Layout	Description	When to Use
Column	Stacks elements vertically	When you need to arrange items from top to bottom
Row	Arranges elements horizontally	When you need to place items side by side
Box	Overlays elements on top of each other	When you need to layer items or position them precisely

When to Use

- When you need to organize multiple UI elements on a screen
- When you want to create structured, visually appealing layouts

- When you need to position elements relative to each other
- When you want to create responsive designs that adapt to different screen sizes

You also use something called a Modifier with layouts (and their children) to control appearance and positioning, like adding padding, setting size, alignment, or background color. We'll talk about modifiers in more detail later in the book, but for now, just see how they are used in the code examples.

Column Layout

When to Use

- When you need to stack elements vertically
- When creating forms or lists
- When displaying content that flows from top to bottom
- When you want to create a vertical navigation menu

Practical Example

Copy Code

```
@Composable
fun ColumnExample() {
    Column(
        modifier = Modifier
            .padding(16.dp)  // Add space around the column
            .border(2.dp, MaterialTheme.colorScheme.secondary)  // Add a border
    ) {
        Text("Part 1")
        Text("Part 2")
        Text("Part 3")
        Text("Part 4")
        Text("Part 5")
    }
}
```

In this example, everything inside the `Column` is stacked vertically. The `Modifier.padding(16.dp)` adds space around the column so its content doesn't touch the edge of the screen. The `Modifier.border(2.dp, MaterialTheme.colorScheme.secondary)` adds a border around the column to make it easier to see its size and position.

Row Layout

When to Use

- When you need to arrange elements horizontally
- When creating navigation bars or toolbars
- When displaying items side by side
- When you want to create a horizontal list of options

Practical Example

Copy Code

```
@Composable
fun RowExample() {
    Row(
        modifier = Modifier
            .padding(16.dp)  // Add space around the row
            .border(2.dp, MaterialTheme.colorScheme.tertiary)  // Add a border
    ) {
        Text("Part 1")
        Spacer(modifier = Modifier.width(20.dp))  // Add space between elements
        Text("Part 2")
        Spacer(modifier = Modifier.width(20.dp))
        Text("Part 3")
        Spacer(modifier = Modifier.width(20.dp))
        Text("Part 4")
        Spacer(modifier = Modifier.width(20.dp))
        Text("Part 5")
    }
}
```

In the RowExample, the text elements are displayed side by side. The Spacer adds horizontal space between them. The Modifier.padding(16.dp) adds space around the row so its content doesn't touch the edge of the screen. The Modifier.border(2.dp, MaterialTheme.colorScheme.tertiary) adds a border around the row to make it easier to see its size and position.

Box Layout

When to Use

- When you need to overlay elements on top of each other
- When you want to position elements precisely within a container
- When you need to layer UI elements
- When you want to create complex layouts with overlapping components

Practical Example

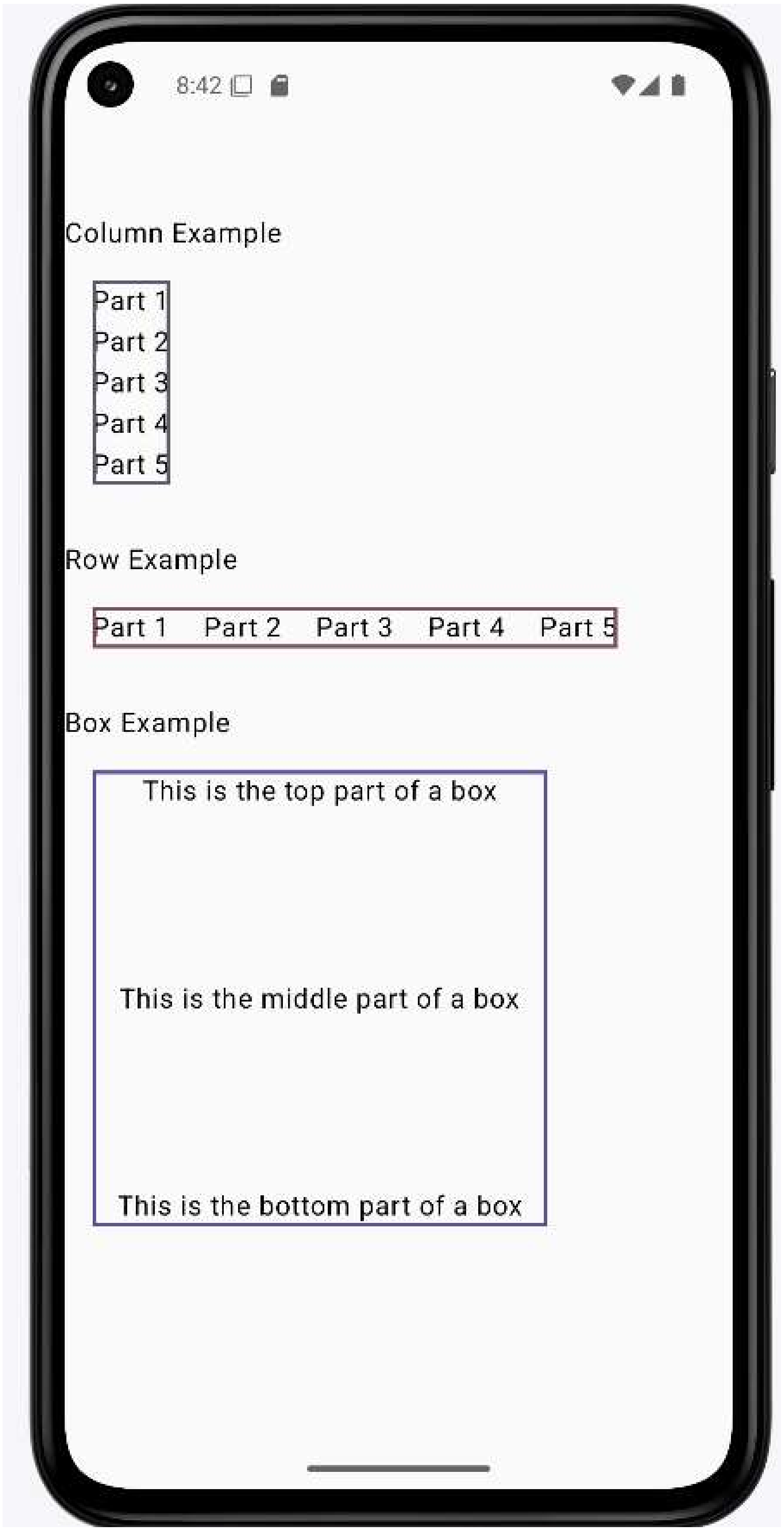
Copy Code

```
@Composable
fun BoxExample() {
    Box(
        modifier = Modifier
            .size(300.dp)  // Set the size of the box
            .padding(16.dp)  // Add space around the box
            .border(2.dp, MaterialTheme.colorScheme.primary)  // Add a border
    ) {
        Text(
            text = "This is the top part of a box",
            modifier = Modifier.align(Alignment.TopCenter)  // Position at the top center
        )
        Text(
            text = "This is the middle part of a box",
            modifier = Modifier.align(Alignment.Center)  // Position in the center
        )
        Text(
            text = "This is the bottom part of a box",
            modifier = Modifier.align(Alignment.BottomCenter)  // Position at the bottom center
        )
    }
}
```

In the BoxExample, the three text elements are displayed on top of each other. The Modifier.size(300.dp) sets the size of the box. The Modifier.padding(16.dp) adds space around the box so its content doesn't touch the edge of the screen. The Modifier.border(2.dp, MaterialTheme.colorScheme.primary) adds a border around the box to make it easier to see its size and position.

How this example renders

Above is just a snippet of the code to view the full code, you need to go to my GitHub page and look at the **chapter4 layout_code.kt file**.



Tips for Success

- Start with simple layouts and build up to more complex ones
- Use the Compose Preview feature to see your layouts as you build them
- Combine different layout types to create complex UIs
- Use modifiers to fine-tune the appearance and positioning of your layouts
- Test your layouts on different screen sizes and orientations
- Use the Compose documentation and samples as reference

Common Mistakes to Avoid

- Creating overly complex layouts that are hard to maintain
- Not using modifiers to control the appearance and positioning of layouts
- Not considering different screen sizes and orientations
- Not using the Compose Preview feature to check your layouts
- Not following Material Design guidelines for consistency
- Not breaking down complex layouts into smaller, reusable components

Best Practices

- Keep layouts simple and focused on a single responsibility
- Use meaningful names for your layout composables
- Extract reusable layout components into separate composables
- Use the Compose Preview feature to iterate quickly
- Follow Material Design guidelines for a consistent look
- Test your layouts with different screen sizes and orientations