Washtenaw Community College
ANN ARBOR, MICHIGAN

CPS251

Android Development

by Scott Shaper

Custom Modifiers

Introduction

Have you ever found yourself copying and pasting the same styling code over and over? Or wished you could give your favorite combination of modifiers a cool name? That's exactly what custom modifiers are for! Think of them like creating your own special recipe - you combine different ingredients (modifiers) once, give it a name, and then you can use it anywhere you want.

Quick Reference

Concept	**What It Is**	**When to Use It**
Custom Modifier	A reusable function that combines modifiers	Repeating the same styling
Extension Function	A way to add new functions to existing types	Creating modifier functions
Parameterized Modifier	A custom modifier that accepts options	Creating flexible styling

When to Create Custom Modifiers

- When you're repeating the same modifier chain in multiple places
- To make your code more readable and maintainable

- When you want to create consistent styling across your app
- To simplify complex modifier chains
- When you want to create a reusable design system

Common Options

Feature	What It Does	When to Use It
Basic Custom Modifier	Combines fixed modifiers	Consistent styling
Parameterized Modifier	Accepts custom values	Flexible styling
Default Parameters	Provides fallback values	Optional customization

Creating a Basic Custom Modifier

Let's create a custom modifier for styling tags, like the ones you might use for skills or categories:

Copy Code

```
fun Modifier.tagStyle(): Modifier {
    return this
        .background(Color.LightGray)  // Add background color
        .padding(horizontal = 8.dp, vertical = 4.dp)  // Add padding
}
```

This creates a reusable style that you can apply to any element. It's like creating a template for your tags!

Using Your Custom Modifier

Here's how to use your custom modifier in a real layout:

Copy Code

```
@Composable
fun TagExample() {
    Row(horizontalArrangement = Arrangement.spacedBy(8.dp)) {
        Text("Compose", modifier = Modifier.tagStyle())  // Apply the style
        Text("Kotlin", modifier = Modifier.tagStyle())   // Apply the style
        Text("UI", modifier = Modifier.tagStyle())       // Apply the style
    }
}
```

Notice how much cleaner and more readable this is! If you want to change how all tags look, you only need to update the tagStyle function.

Adding Parameters to Custom Modifiers

Want to make your custom modifier more flexible? Add parameters!

Copy Code

```
fun Modifier.tagStyle(
    color: Color = Color.LightGray  // Default color if none provided
): Modifier {
    return this
        .background(color)  // Use the provided color
        .padding(horizontal = 8.dp, vertical = 4.dp)
}
```

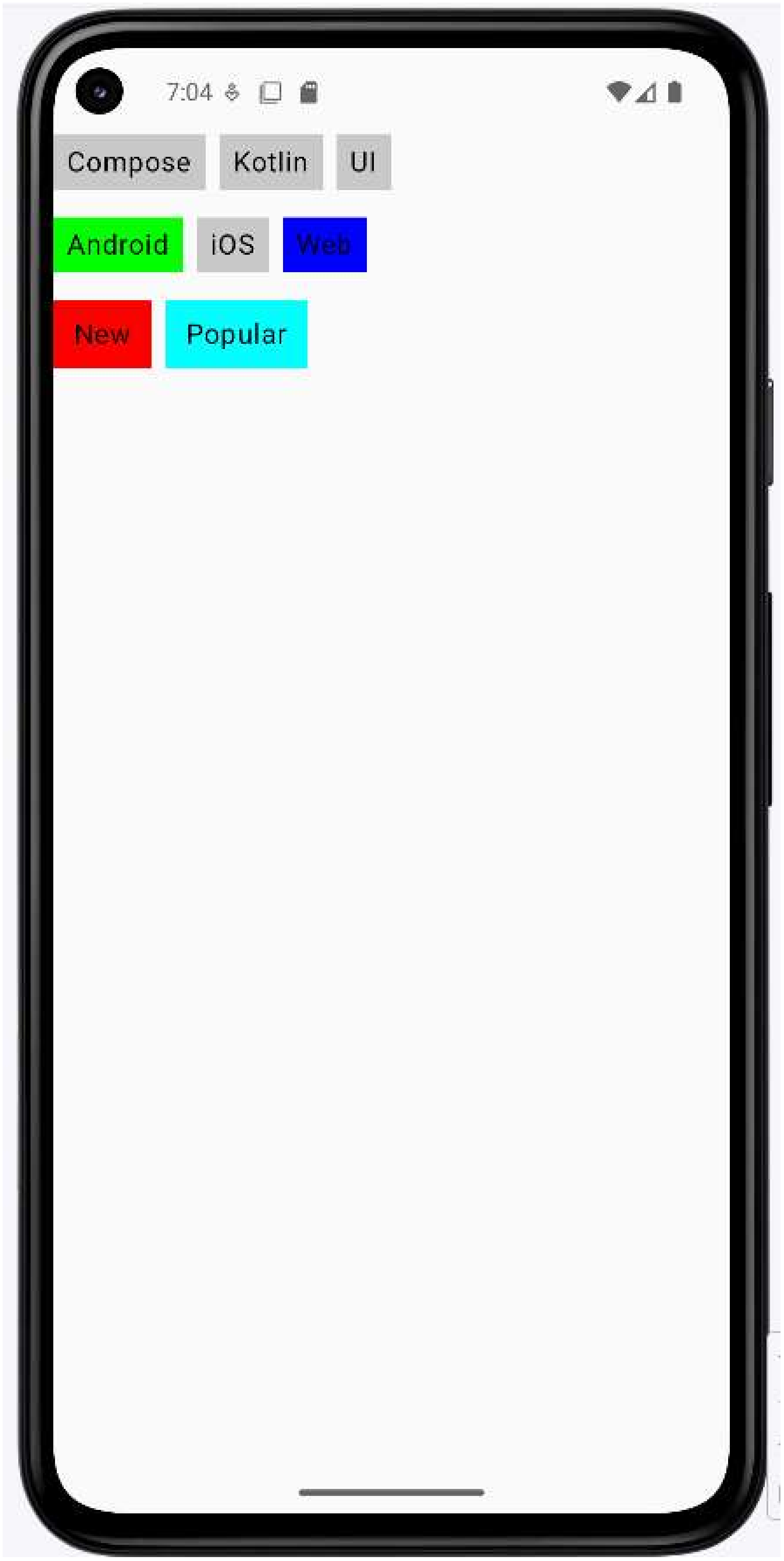
Now you can customize the color when you use it:

Copy Code

```
Text("Android", modifier = Modifier.tagStyle(Color.Green))  // Custom color
Text("iOS", modifier = Modifier.tagStyle())  // Default color
```

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 **chapter5 cust_modifier.kt file**.



Tips for Success

- Give your custom modifiers clear, descriptive names
- Start with basic modifiers before adding parameters
- Use default values for optional parameters
- Keep your modifier functions focused and simple

Common Mistakes to Avoid

- Creating too many parameters in one modifier
- Making modifiers too specific to one use case
- Forgetting to return the modified chain
- Not using default values for optional parameters

Best Practices

- Create modifiers for commonly used style combinations
- Use meaningful parameter names
- Document your custom modifiers with comments
- Test your modifiers with different content