CMPUT 301

Lab 3 Presentation

Overview

- Dynamic Lists in Compose
- Reusable Row Composables
- Callbacks for User Actions
- State and Recomposition
- Adding Data with Callbacks
- Code Conventions
- Lab Demo

Dynamic Lists in Compose

- The code displays a fixed list of City objects
- LazyColumn displays each city on screen
- itemsIndexed(cities) creates one row for each city
- CityRow controls how each row looks
- Later, the list will become dynamic when users add new cities

```
@Composable  2 Usages
fun CityListScreen(
    cities: List<City>,
    modifier: Modifier = Modifier
) {
    LazyColumn(modifier = modifier) {
        itemsIndexed(cities) { index, city ->
            CityRow(city = city)

            if (index < cities.lastIndex) {
                HorizontalDivider()
            }
        }
    }
}
```

Reusable Composable Design

- Write small composable functions for specific UI pieces
- Give composables the data they need as parameters
- Include a `modifier: Modifier = Modifier` parameter
- Use callback parameters for user actions
- Keep the composable flexible instead of hard-coding all behaviour

```
@Composable
fun CityRow(
    city: City,
    modifier: Modifier = Modifier
) {
    Row(
        modifier = modifier
            .fillMaxWidth()
            .padding(horizontal = 20.dp, vertical = 16.dp)
    ) {
        Text(
            text = city.name,
            fontSize = 30.sp,
            modifier = Modifier.weight(1f)
        )

        Text(
            text = city.province,
            fontSize = 30.sp,
            modifier = Modifier.weight(1f)
        )
    }
}
```

Callbacks for User Actions

- A callback is a function passed a parameter
- The parent receives that event and decides what to do with it
- Example:
  - CityListScreen can call onAddCity(...)
  - The parent connects that callback to CityRepository

State and Recomposition

- State is data that can change while the app is running
- Compose updates the UI when state changes
- This update is called Recomposition
- For a changing list, use state-backed data
- Adding a city should update the list shown on screen

```
class CityRepository {  1 Usage
    private val _cities = mutableStateListOf(  1 Usage
        City( name = "Edmonton",  province = "AB"),
        City( name = "Vancouver",  province = "BC"),
        City( name = "Toronto",  province = "ON")
    )
```

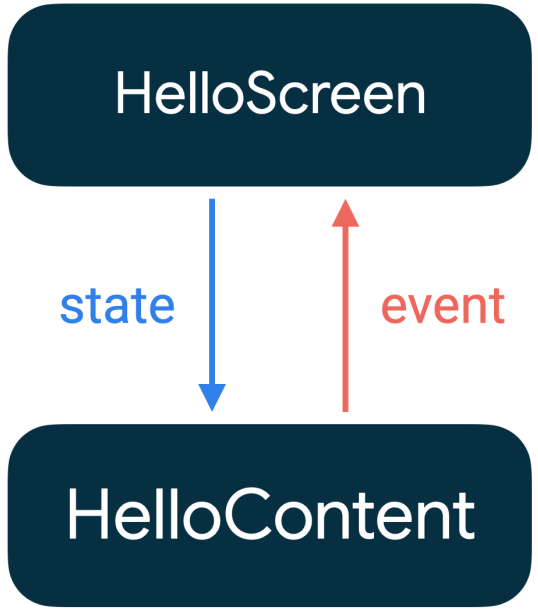
List data that can change and update the UI
UI recomposes when list changes

```
class CityRepository {  1 Usage
    private val _cities = listOf(  1 Usage
        City( name = "Edmonton",  province = "AB"),
        City( name = "Vancouver",  province = "BC"),
        City( name = "Toronto",  province = "ON")
    )
```

Fixed data
UI does not need to update from list changes

Adding Data with Callbacks

- Compose apps commonly use unidirectional data flow
- State goes down into composables
- Events go up through callback functions
- CityListScreen receives the current city list
- CityListScreen reports the Add City event through onAddCity(...)




```
@Composable  2 Usages
fun CityListScreen(
    cities: List<String>,
    onAddCity: (String) -> Unit,
    modifier: Modifier = Modifier
) {
    // Add City button will call onAddCity(newCity)
}
```

Source: https://developer.android.com/develop/ui/compose/state

Code Conventions

- Code conventions are important to programmers for a number of reasons:
  - ~80% of the lifetime cost of a piece of software goes to maintenance.
  - Hardly any software is maintained for its whole life by the original author
  - Coding standards improve software readability by allowing developers to understand new code faster and better
  - If you ship your source code as a product, you need to make sure it is as well
  - Like any other product, the software must be “well packaged” and clean

Source: https://svitla.com/blog/why-where-and-when-to-use-coding-conventions/