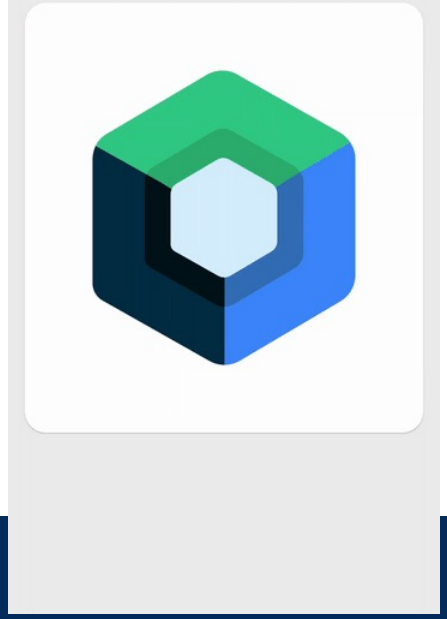
Mobile Applications
CSCI 448
Lecture 05



Stateful

Composables

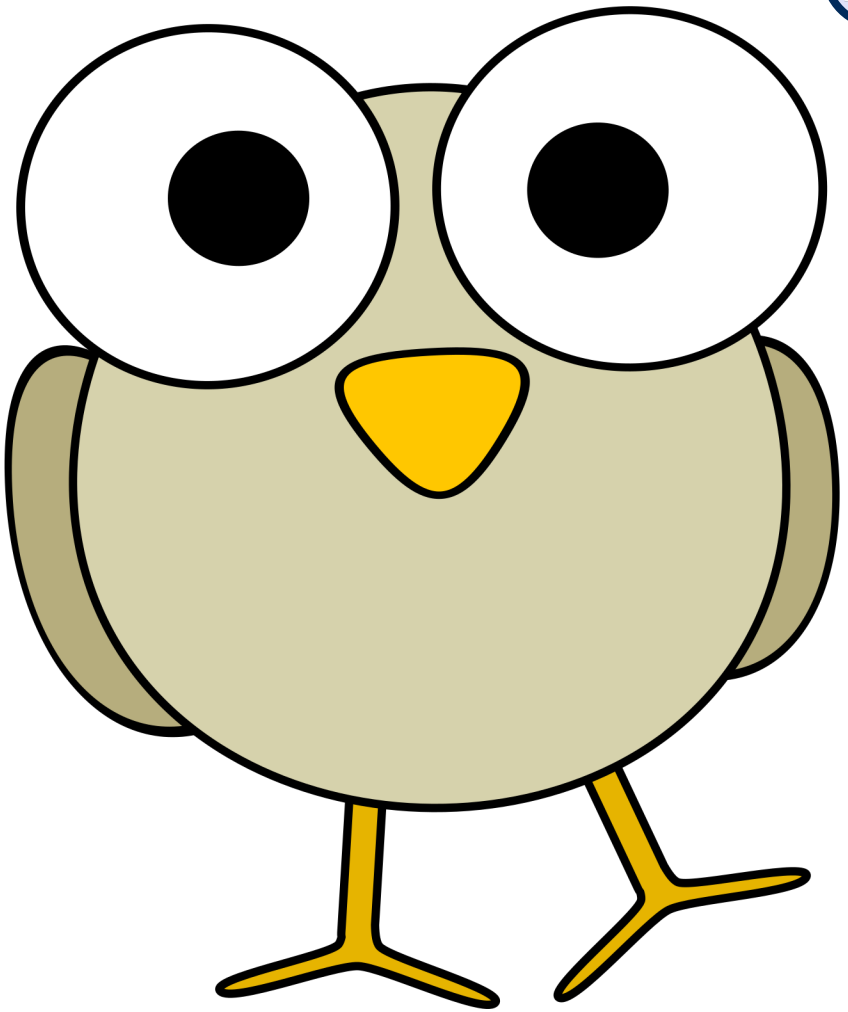
Have TempConverter

Loaded To Continue Implementing

Previously in CSCI 448

- Composables: Functions emit UI
  - Declarative paradigm
  - Slot API
- Toast
- Single source of truth
  - Events “flow” up

Questions?



Learning Outcomes For Today

- Explain when a composable gets recomposed.
- Explain how Compose preserves unidirectional data flow.
- Explain how a stateful composable stores and modifies state.
- Create an app that uses stateful composables.

On Tap For Today

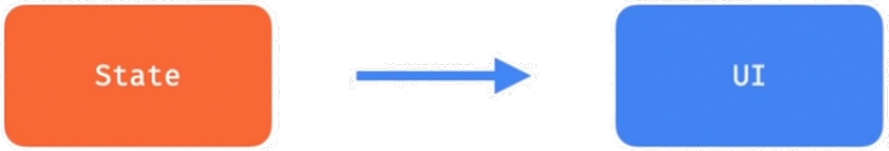
- Recomposing
- Unidirectional Data Flow
- Stateful Composables

On Tap For Today

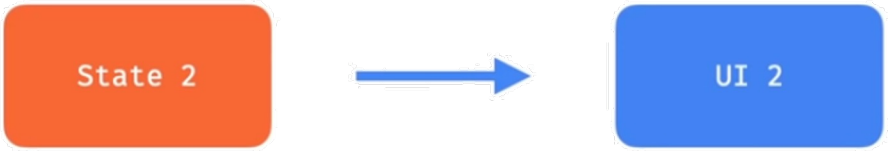
- Recomposing
- Unidirectional Data Flow
- Stateful Composables

Composing

- Given state, composable emits corresponding UI

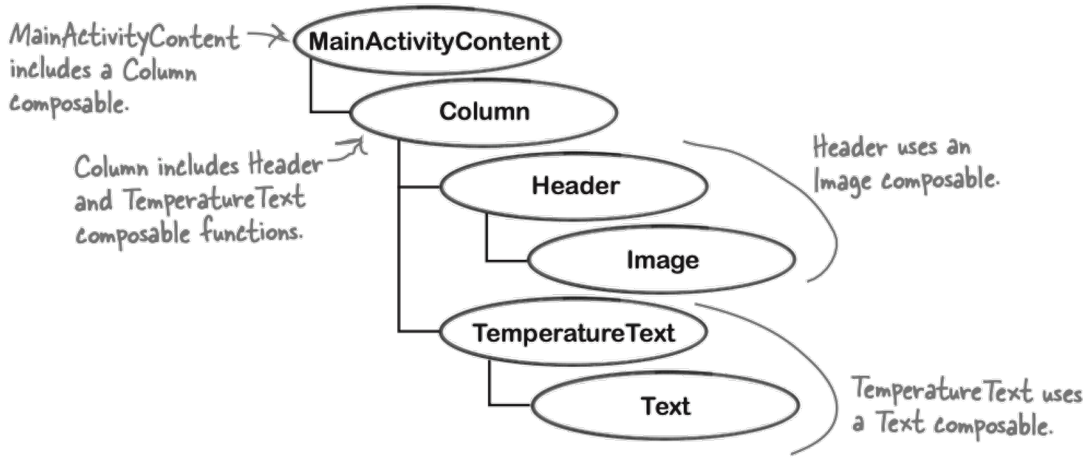


- UI is
  - Idempotent & immutable: there are no objects
  - Dynamic: different inputs → Different UI



Composable Tree

- When first building UI, hierarchical tree of all composables created



- When the input to `TemperatureText` changes, we need to replace it with the new UI
  - This is known as **recomposing**

Updating Temperature

- Concept:

Demo Time!

```
@Composable
fun UI() {
  Column {
    var celsius = 0.0
    TemperatureText(celsius)
    Button(onClick = { celsius = 100.0 }) {
      Text("Boil!")
    }
  }
}

@Composable
fun TemperatureText(celsius: Double) {
  val fahrenheit = (celsius*9.0/5.0) + 32.0
  Text("$celsius C = $fahrenheit F")
}
```

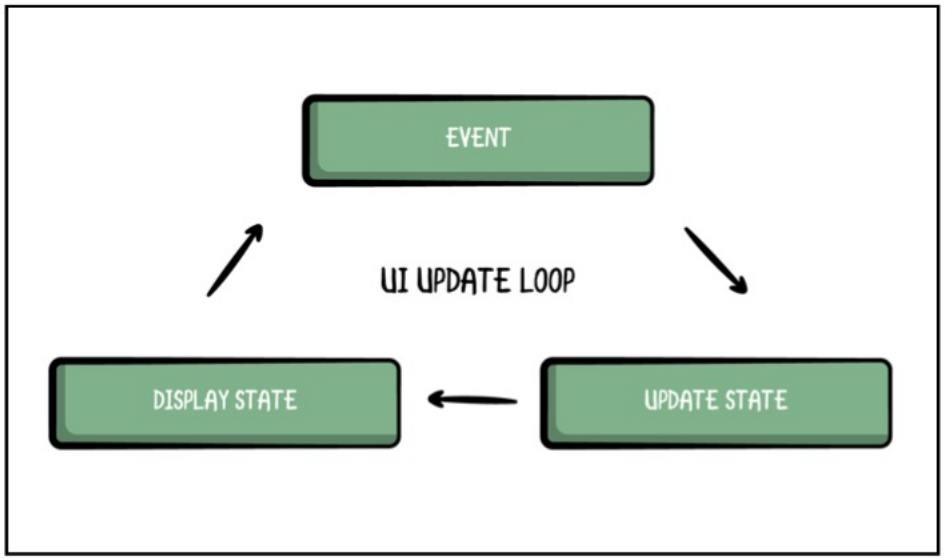
- But how does `TemperatureText` know `celsius` has changed?

On Tap For Today

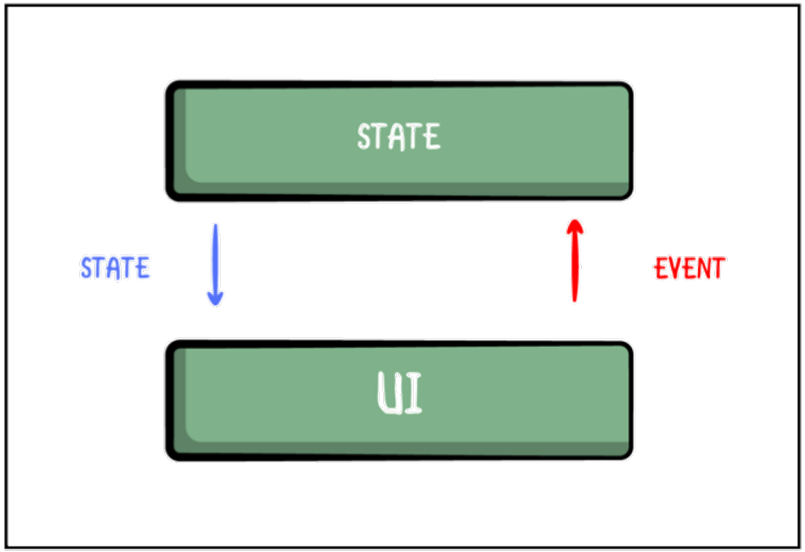
- Recomposing
- Unidirectional Data Flow
- Stateful Composables

Unidirectional Data Flow

- External events trigger change in state



Single Source of Truth



Unidirectional Data Flow

- Keep one state
- State “flows” down
- Events “flow” up
- UI “observes” the state

Where to Store State???

- What is the single source of truth?
  1. Composable - In the composable itself
     - A **stateful** composable
     - Can change state itself
  2. ViewModel - “Hoist” the state to the caller of the composable
     - A **stateless** composable
     - Composable requires parameter and event
  3. StateHolder
     - Separate class that stores UI logic & UI element states

On Tap For Today

- Recomposing
- Unidirectional Data Flow
- Stateful Composables

Stateful Composables

- Composable stores a state variable
- Composable observes the state variable
- When state changes, composable gets recomposed

- Composables create a tree
  - Observer could be self node
  - Observer could be child node

Design Pattern #2: Observer

- Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
- Participants:
  - `Subject`: knows it observers, provides interface for adding/removes `Observer` objects
  - `Observer`: defines an updating interface for objects that should be notified of changes in a subject
  - `ConcreteSubject`: stores a state of interest to `ConcreteObserver` objects and sends a notification to its observers when its state changes
  - `ConcreteObserver`: maintains a reference to a `ConcreteSubject`, stores state that should stay consistent with the subject's, implements the `Observer` updating interface

Stateful Composable Observer

- `Subject` →
- `Observer` →
- `ConcreteSubject` →
- `ConcreteObserver` →

Android Design Patterns

- Behavioral Patterns
  1. Command – UI Event Handling
  2. Observer – State

Updating Temperature

- Implementation:

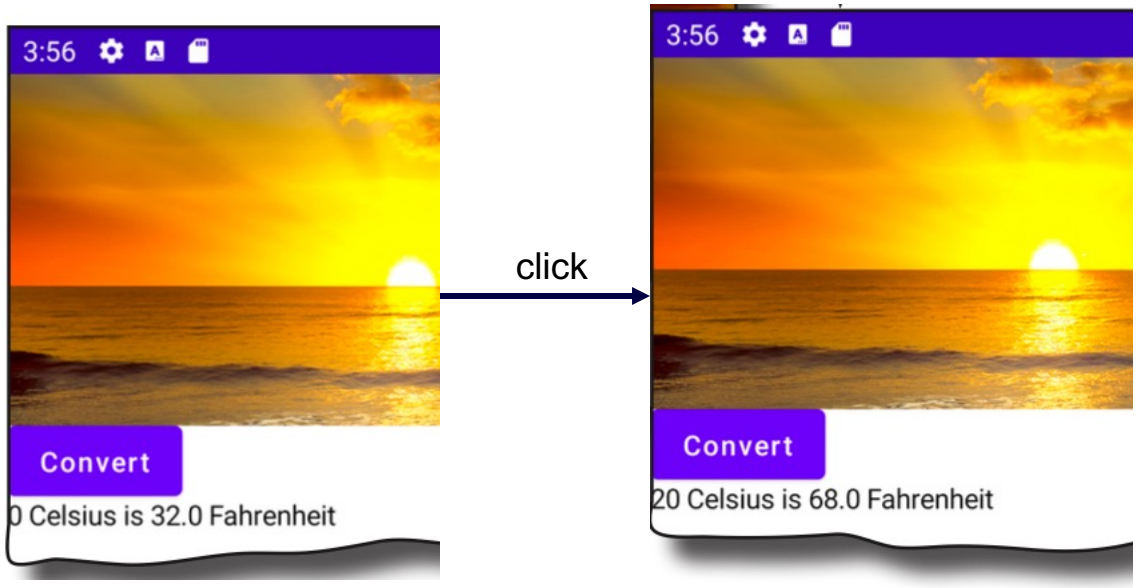
Demo Time!

```
@Composable
fun UI() {
  Column {
    val celsius = remember { mutableStateOf(0.0) }
    TemperatureText(celsius.value)
    Button(onClick = { celsius.value = 100.0 }) {
      Text("Boil!")
    }
  }
}

@Composable
fun TemperatureText(celsius: Double) {
  val fahrenheit = (celsius*9.0/5.0) + 32.0
  Text("$celsius C = $fahrenheit F")
}
```

- Question: Which composable is stateful?

Recomposing



On Tap For Today

- Recomposing
- Unidirectional Data Flow
- Stateful Composables

A Brief History…

- ~3200 BCE Egypt

A Brief History…

- ~2500 BCE Egypt

A Brief History...

- 1st Century AD

…to current

Display a List in Compose

- Use LazyColumn composable

```
@Composable
fun ListScreen() {
  val myList = List<T>
  LazyColumn {
    items(myList) { singleItem ->
      DisplayOneItem(singleItem)
    }
  }
}
@Composable
fun DisplayOneItem(item: T) {
  // display that item
}
```

To Do For Next Time

- Final Project Team Formation due Friday
  - Groups of 3
  - Each team member submits
    - Team Name
    - Team Members
    - Whose app you are creating
    - App Name
    - One paragraph description of the app