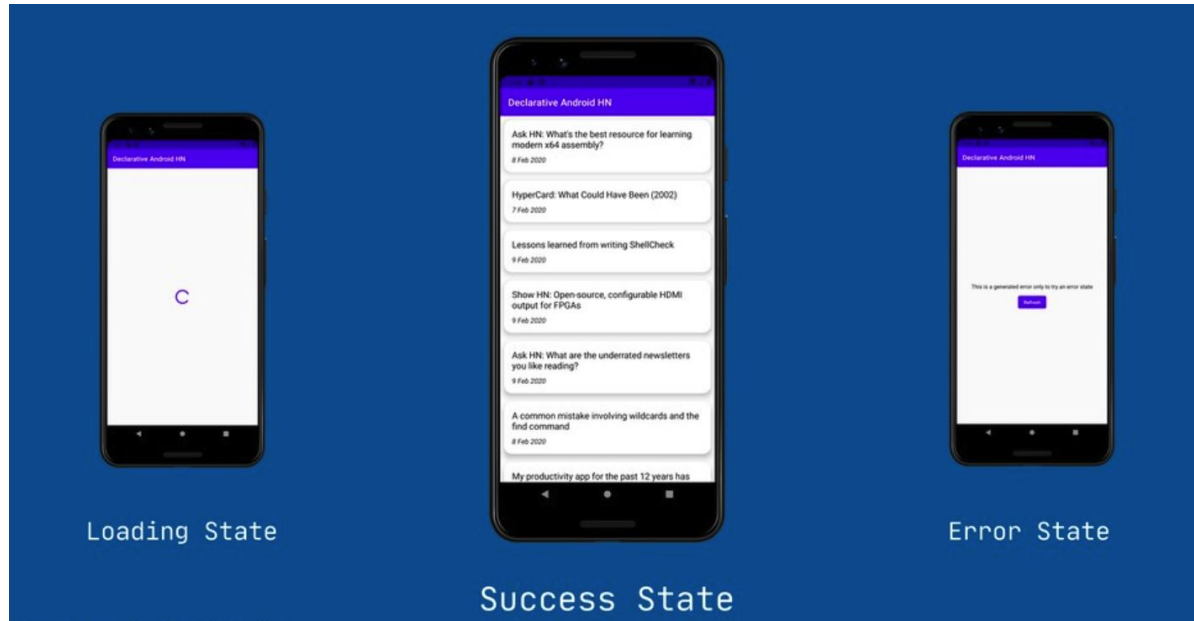




Imperative or Declarative

If we want to realize it...



Imperative



XML or Code

```
<androidx.constraintlayout.widget.ConstraintLayout
... />

<ProgressBar
  android:id="@+id/progress_bar"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  ... />

<TextView
  android:id="@+id/error_message"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:visibility="gone"
  ... />

<Button
  android:id="@+id/error_button"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:text="Refresh"
  android:visibility="gone"
  ... />

<androidx.recyclerview.widget.RecyclerView
  android:id="@+id/recycler_view"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:visibility="gone"
  ... />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Imperative



```
val progressBar = findViewById<ProgressBar>(R.id.progress_bar)
```

Imperative (handle list)



RecyclerView.Adapter

```
class NewsAdapter(var items: List<News> = listOf()) :  
    RecyclerView.Adapter<NewsAdapter.ViewHolder>() {  
    ...  
}
```

Imperative (logic)



Kotlin

```
when (appState.newsState) {  
    is NewsState.Loading → {  
        progressBar.visibility = View.VISIBLE  
        ...  
    }  
  
    is NewsState.Error → {  
        progressBar.visibility = View.GONE  
        errorMessage.visibility = View.VISIBLE  
        ...  
    }  
  
    is NewsState.Success → {  
        progressBar.visibility = View.GONE  
        errorMessage.visibility = View.GONE  
        recyclerView.visibility = View.VISIBLE  
        ...  
    }  
}
```

Imperative

- A lot of boilerplate code
- A very heavy development
- Hard to mantain

Welcome to Declarative!

**“ a programming paradigm that
uses statements that change a
program's state ”**

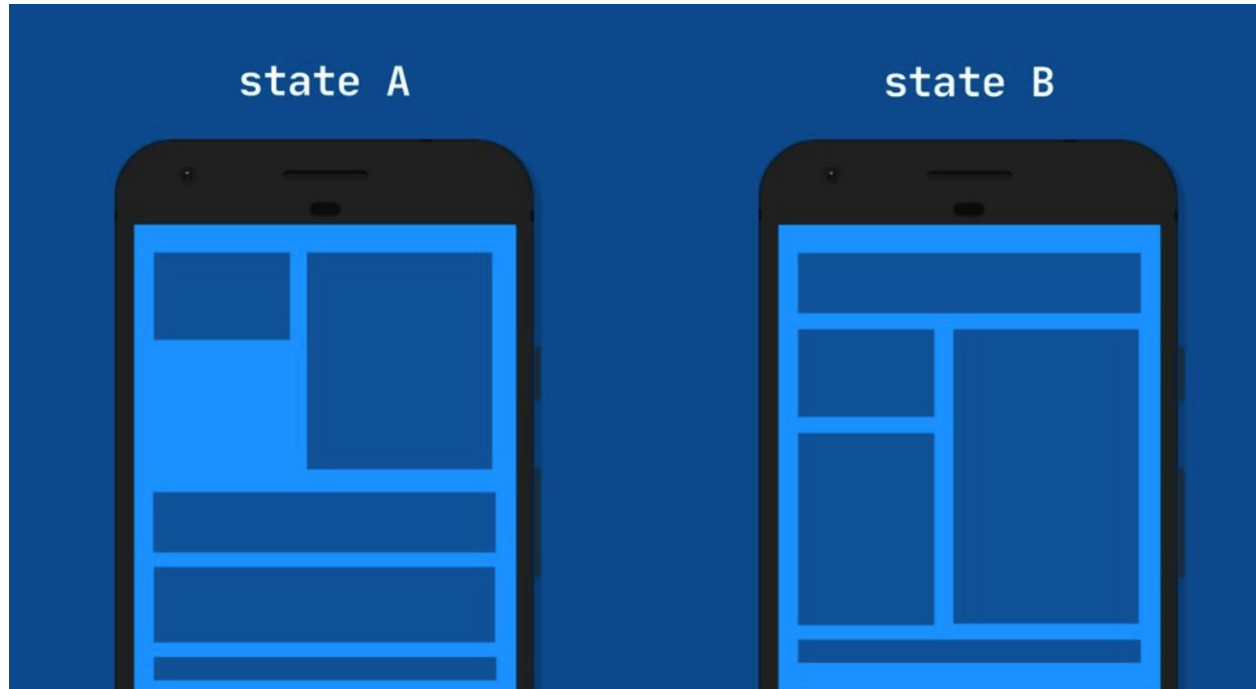
- Wikipedia

Declarative

“ what the program should accomplish without specifying how the program should achieve the result. ”

- Wikipedia

Declarative



Declarative

$$\text{UI} = f(\text{state})$$

Declarative - State

- Declare the UI at the moment
- Independent from the previous states
- If changes, the UI is redrawn

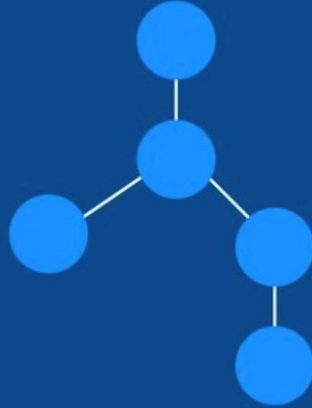
Declarative

Changes are handled by the magician system
not by the developer



Declarative

1. Creates an abstract representation of the UI and renders it



Declarative

1. Creates an abstract representation of the UI and renders it
2. When a change is made, it creates a new representation
3. Computes the differences between the two representations
4. Renders the differences

Declarative

React [Native]



- Virtual DOM as representation of the UI

```
<button class='button button-blue'>
  <b>
    OK!
  </b>
</button>
```

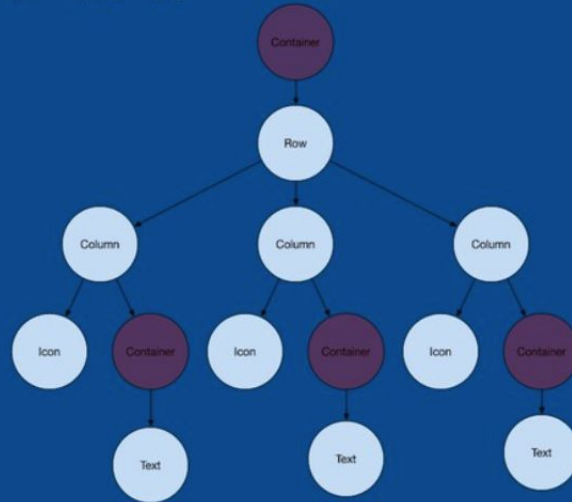


```
{
  type: 'button',
  props: {
    className: 'button button-blue',
    children: {
      type: 'b',
      props: {
        children: 'OK!'
      }
    }
  }
}
```


Declarative

Flutter

- Element Tree as representation of the UI



Declarative

Jetpack Compose



- Slot table as representation of the UI

A screenshot of a video presentation. On the left, a diagram titled "Slot Table" shows a vertical list of seven slots. The first three slots contain a red triangle, a green square, and a green triangle. The next three slots are labeled "EMPTY". The last slot contains a red circle. A red arrow points to the first "EMPTY" slot. To the right of the diagram, a vertical axis is numbered 0 to 5. A red bracket labeled "Gap" spans from index 3 to index 5. On the right side of the video frame, a man is speaking at a podium with a "KOTLIN CONF 2019" sign. Below the video frame, the YouTube player interface shows the video title "KotlinConf 2019: The Compose Runtime, Demystified by Leland Richardson" and a view count of 1,771.

Declarative - UI



Component

```
export default class NewsCard extends React.Component {
  constructor(props) {
    super(props);
  }

  render() {
    const { news } = this.props;
    if (news) {
      return (
        <View style={styles.container}>
          <TouchableOpacity .. >
            <Text .. >
            <Text .. >
          </TouchableOpacity>
        </View>
      );
    }
    return null;
  }
}
```



Component

Declarative - UI

Widget

```
class NewsCardWidget extends StatelessWidget {  
  final News news;  
  
  NewsCardWidget({this.news});  
  
  @override  
  Widget build(BuildContext context) {  
    return Padding(  
      padding: const EdgeInsets.all(6.0),  
      child: Card(  
        elevation: 8.0,  
        shape: RoundedRectangleBorder(  
          borderRadius: BorderRadius.circular(16.0),  
        ),  
        child: ... ,  
      ),  
    );  
  }  
}
```

Component

 Widget

Declarative - UI



composable function

```
@Composable
fun NewsCard(news: News) {
    Card(
        shape = RoundedCornerShape(16.dp),
        elevation = 8.dp,
        modifier = ..
    ) {

        Column {
            Text(news.title)
            ...
        }
    }
}
```

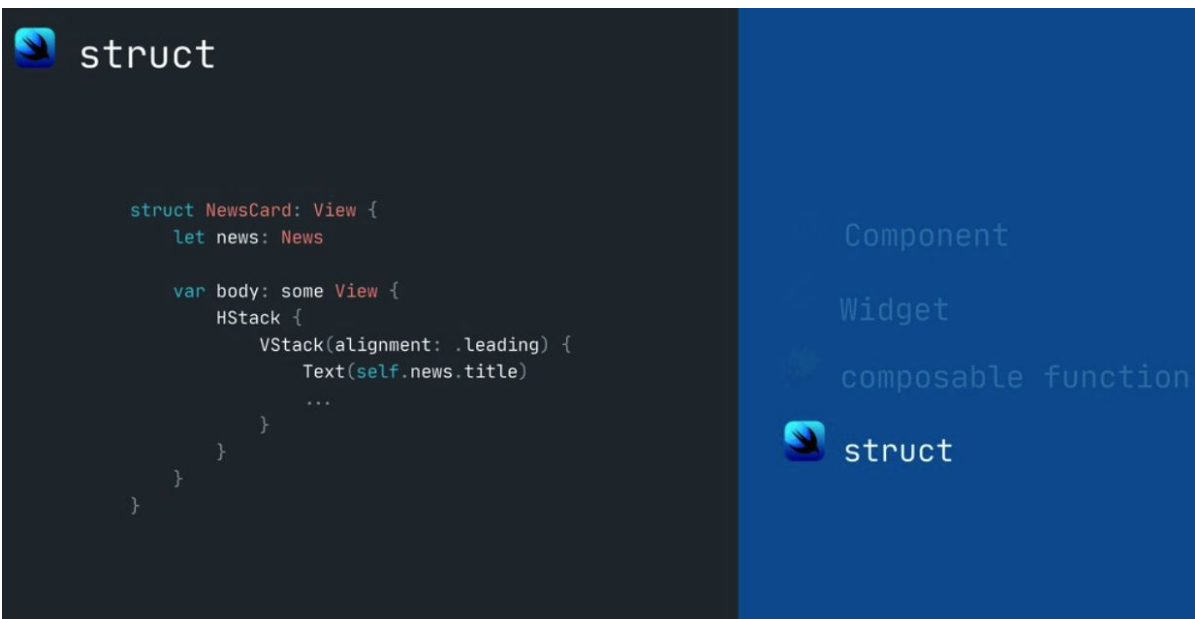
Component

Widget



composable function

Declarative - UI



The image shows a code editor window with a dark background on the left and a blue background on the right. The left pane contains Swift code for a struct named `NewsCard`. The right pane lists several UI-related terms: `Component`, `Widget`, `composable function`, and `struct`.

```
struct NewsCard: View {  
    let news: News  
  
    var body: some View {  
        HStack {  
            VStack(alignment: .leading) {  
                Text(self.news.title)  
                ...  
            }  
        }  
    }  
}
```

Component
Widget
composable function
struct

Declarative

Combine the Ui with a **pure function**! It returns the same value, given the same input and it has no side effects.



```
fun sum(a: Int, b: Int): Int {  
    return a + b  
}
```



```
fun addExpense(account: BackAccount, amount: Int) {  
    account.balance = account.balance - amount  
}
```

Thanks for being here!