Flutter Fundamentals
Stateful Widgets

Peter Kassenaar –
info@kassenaar.com

Stateful widgets

Changing data / state in your widgets, over time

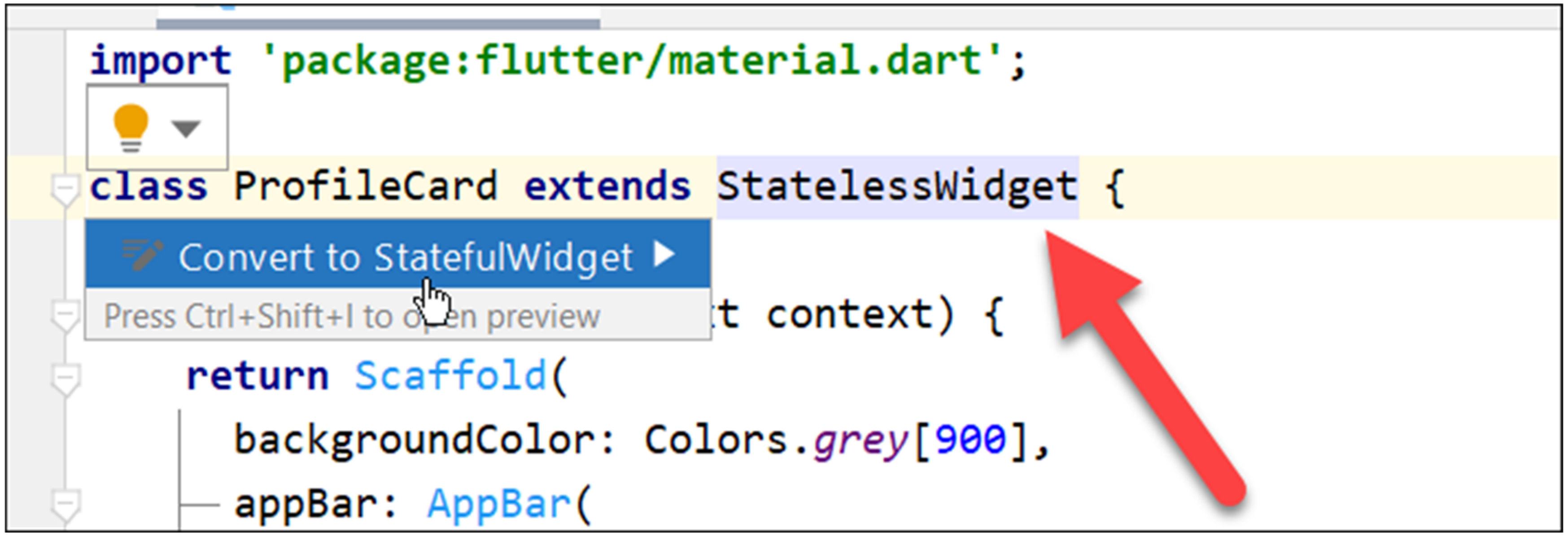
Creating stateful widgets from scratch

- Use the snippet `stful <Tab>` in IntelliJ
- It creates actually two classes:

```
class TestClass extends StatefulWidget {
  @override
  _TestClassState createState() => _TestClassState();
}

class _TestClassState extends State<TestClass> {

  // Define data (or 'state') over here and change it over time

  @override
  Widget build(BuildContext context) {
    return Container(
      // Define and return the user interface
      // it runs every time the state is changed!
    );
  }
}
```

Converting stateless widgets

- You *can* cut/paste it into newly created components
  - However – time consuming/error prone
- Use the `Convert to StatefulWidget` action menu



```
import 'package:flutter/material.dart';

class ProfileCard extends StatelessWidget {
                                     t context) {
    return Scaffold(
      backgroundColor: Colors.grey[900],
      appBar: AppBar(
```

Result

```dart
import 'package:flutter/material.dart';

class ProfileCard extends StatefulWidget {
  @override
  _ProfileCardState createState() => _ProfileCardState();
}

class _ProfileCardState extends State<ProfileCard> {

  // go ahead and create variables/state here...

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.grey[900],
      appBar: AppBar(
       …,
      ),
   …
}
```

Using state / variables

- Define variable as `var`, `int`, `String`, `List`, `Map`, ...
- Use it inside a string, prefixed with dollarsign ($)
- For instance:

```
double cutenessLevel = 0;
```

```
Text('$cutenessLevel',
    style: TextStyle(
    …
))
```

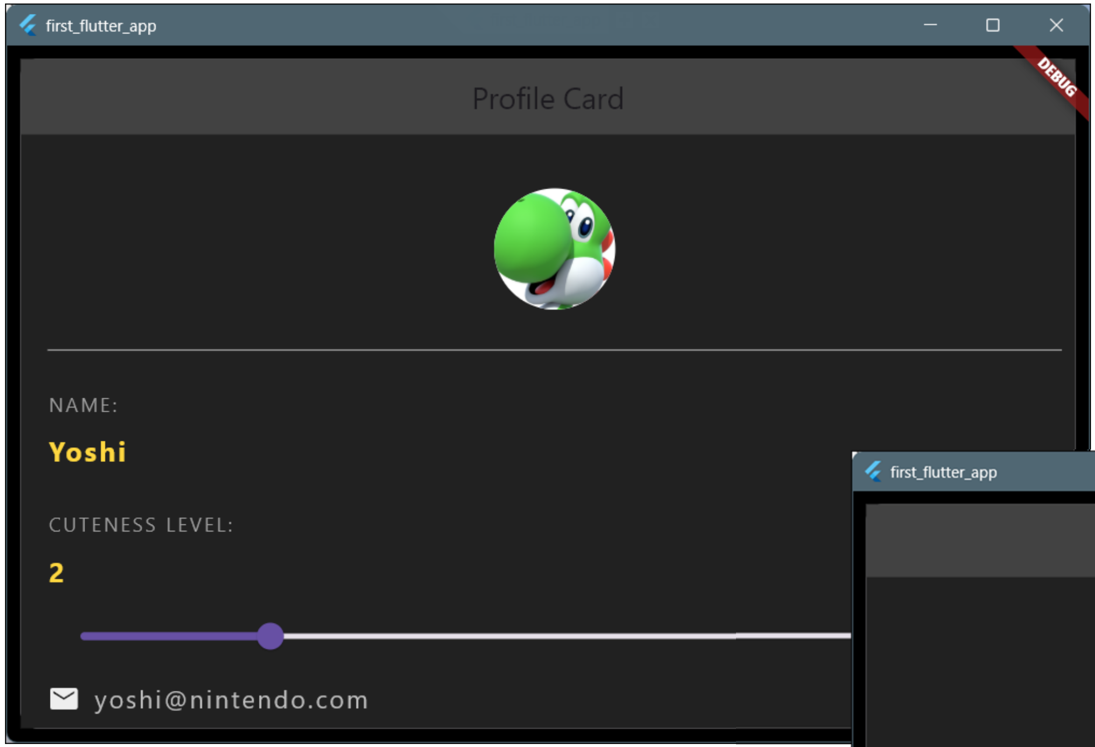
If you want to show the variable *as is* (not in a string), do not use ‘…’ and $.

Updating the state

- Use the function setState() to update the state
- Rerun of build() is automatically triggered
- For instance:

```
Slider(
    value: cutenessLevel,
    min: 0,
    max: 10,
    label: cutenessLevel.round().toString(),
    onChanged: (double value) {
      setState(() {
        cutenessLevel = value.floorToDouble();
      });
    }),
```

Result


first_flutter_app

Profile Card

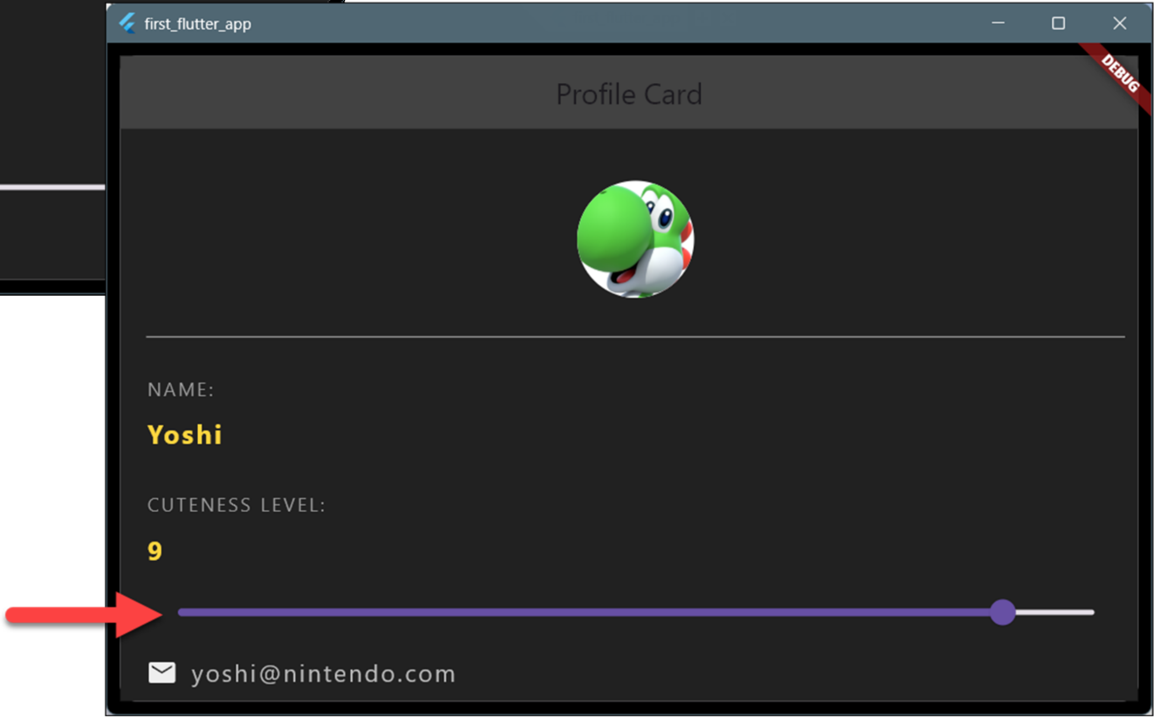
NAME:

Yoshi

CUTENESS LEVEL:

2

yoshi@nintendo.com


first_flutter_app

Profile Card

NAME:

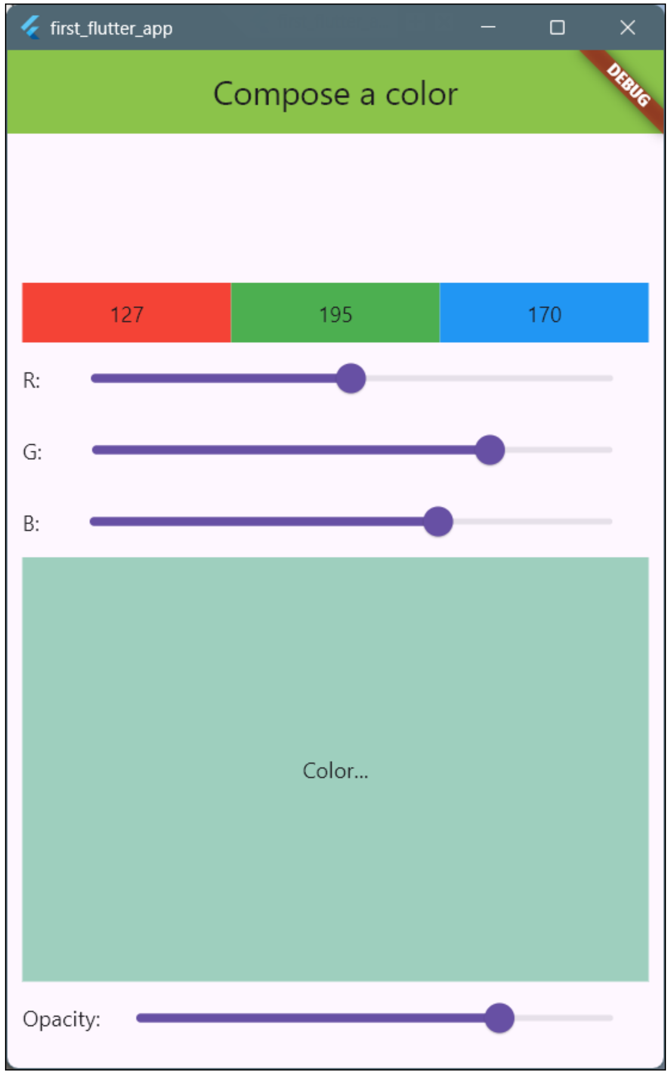
Yoshi

CUTENESS LEVEL:

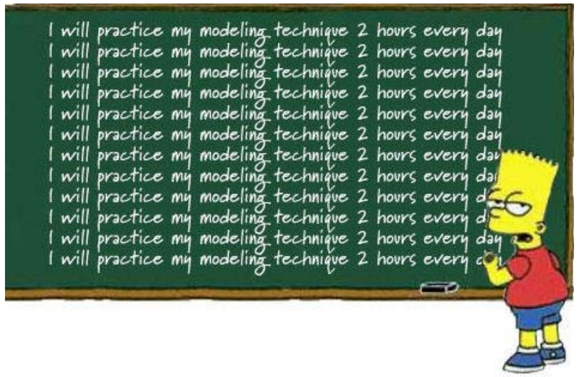
9

yoshi@nintendo.com

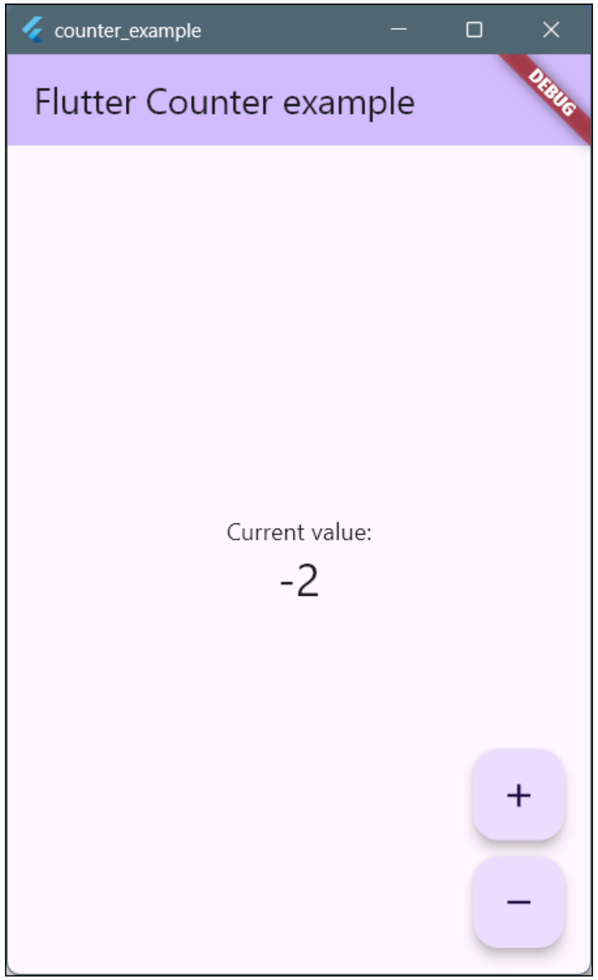
Workshop



- Implement the Slider() shown in prev slides
- AND/OR:
- Create a new `StatefulWidget` in your app
- Create three sliders, for Red, Green, and Blue
- Create an additional slider for Opacity
- Create a container, that uses the combined values as its color
- Use `Color.fromRGBO(….)` to mix the colors
- Official docs: `api.flutter.dev/flutter/material/Slider-class.html`

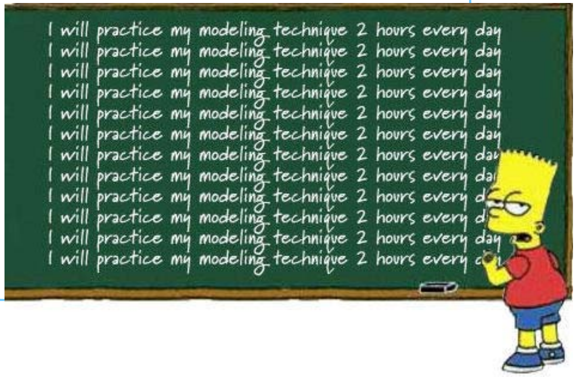


Workshop #2



Current value:
-2

- Create a new Flutter Default Application (the Counter example)
- Now study the code. You should see and understand how the counter value is retained between repainting the screen.
- Create an app with multiple Floating Action Buttons (tip: wrap the buttons in a `Column()`)
  - Create buttons for add, `subtract`, `reset`



Checkpoint

- If we want dynamic UI, we need `statefulWidget()`
- Stateful widgets are actually 2 classes
- We can *only* update data in a statefulWidget by using the `setState()` method
  - (Much like in React)

Using lists of data

Cycling through data and display it on the screen

First approach – using simple strings

- Create a list of strings, loop over it.

```
List<String> cities = ['Amsterdam', 'Berlin', 'New York', 'Sidney', 'Tokyo'];
```

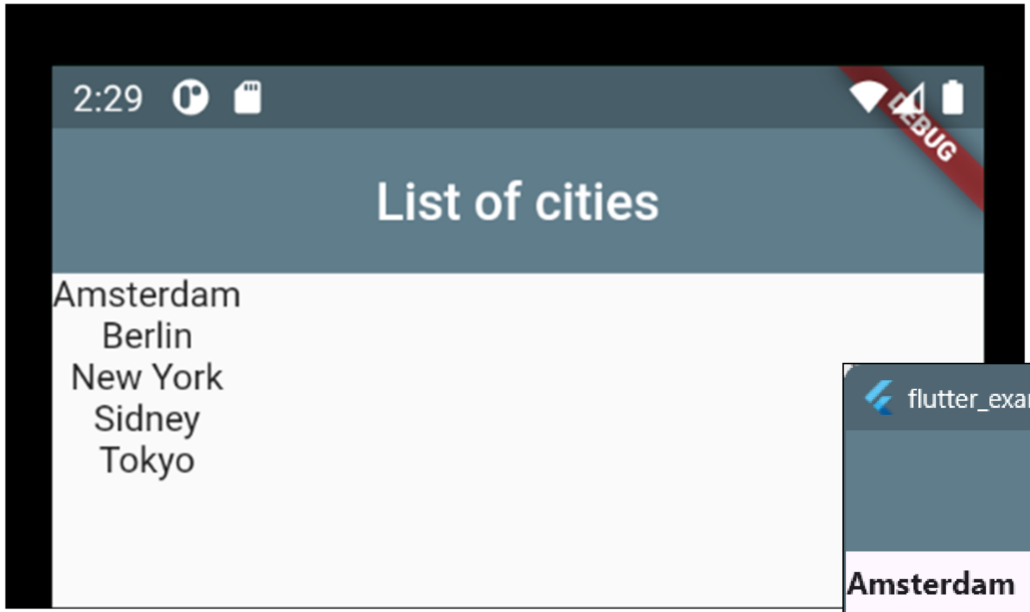
```
return Scaffold(
    appBar: AppBar(…),
    body: Column(
      // Later on: use a ListView here
      children: cities.map((city) {
        return Text(city);
      }).toList(),
    )
);
```

Classic function syntax

```
children: cities.map((city) => Text(city)).toList(),
```

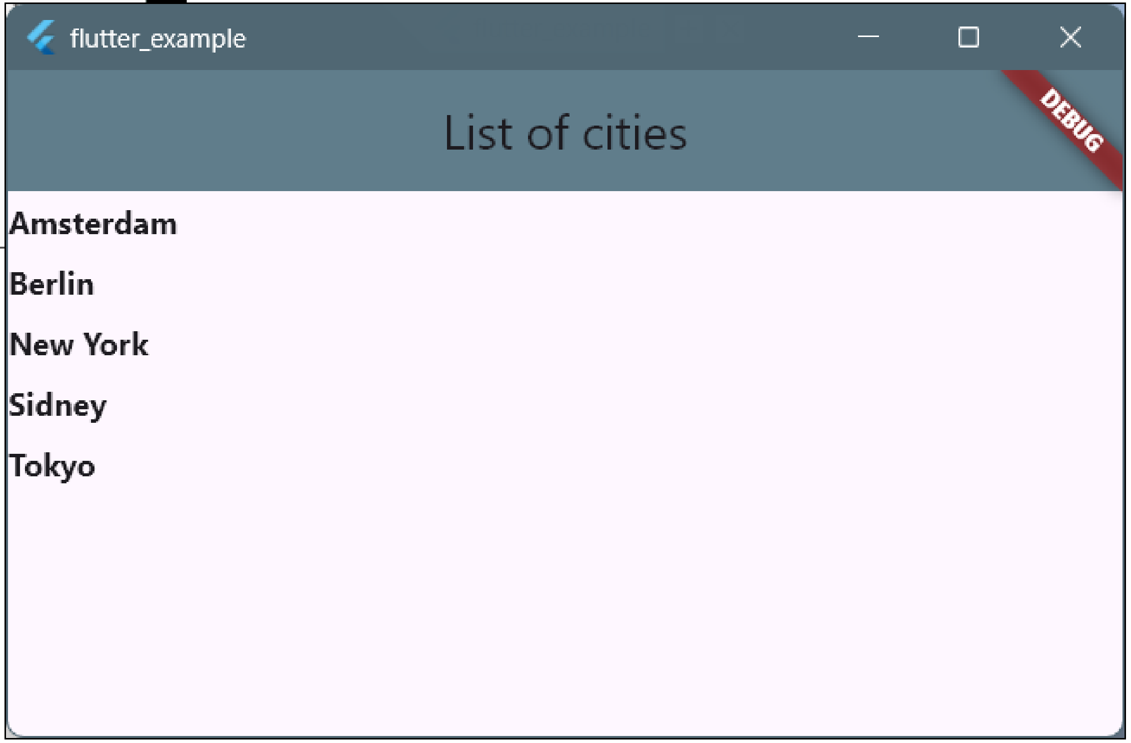
Arrow function syntax

First result – it works!



List of cities

Amsterdam
Berlin
New York
Sidney
Tokyo



List of cities

Amsterdam
Berlin
New York
Sidney
Tokyo

Using a little more lay-out by wrapping the `Text()` widgets in containing widgets like `Container()`, `Padding()`, etc. Experiment with this for yourself!

Custom classes – ‘lists of objects’

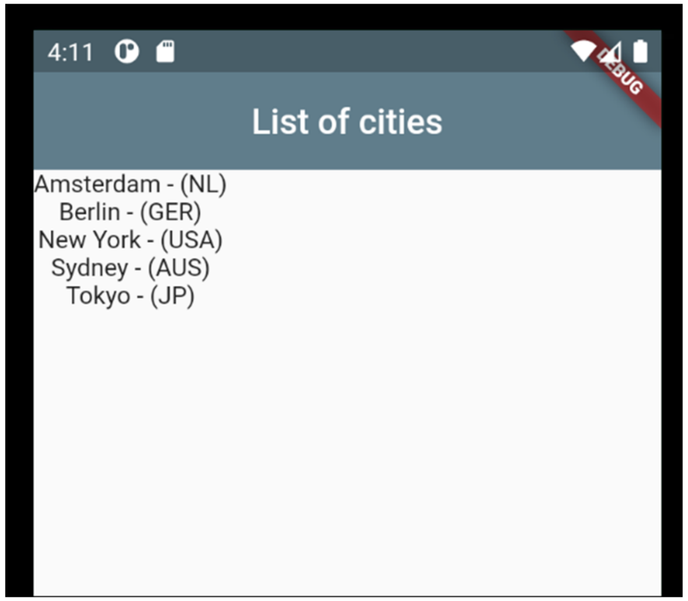
- Create a custom `City` class, holding all properties for a specific city

```
class City {
  late int id;
  late String name;
  late String country;
  late int population;

  // Option 1: constructor of our class - more verbose, not recommended anymore
  City(int id, String name, String country, int population) {
    this.id = id;
    this.name = name;
    this.country = country;
    this.population = population;
  }
}
```

```
  // Using Named Parameters. Notice the {...} notation
  // When not using `late`, we HAVE to use `required`
  // here, b/c the values may not be null.
  City({
    required this.id,
    required this.name,
    required this.country,
    this.population = -1});
}
```

Workshop

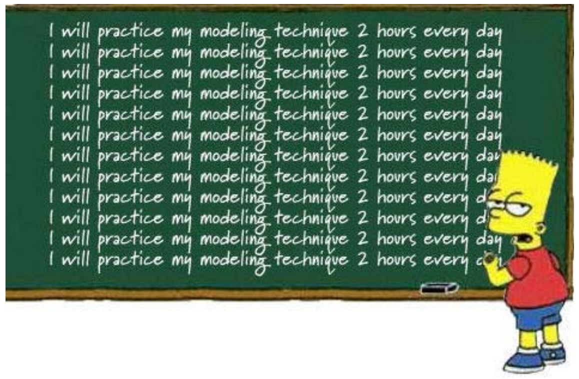
- Create a new `Class` for *your* data.
- Import the class in the widget that loops over it
- Display the class data in a list using a `.map()` function



List of cities

Amsterdam - (NL)
Berlin - (GER)
New York - (USA)
Sydney - (AUS)
Tokyo - (JP)

`../_220-custom-class`

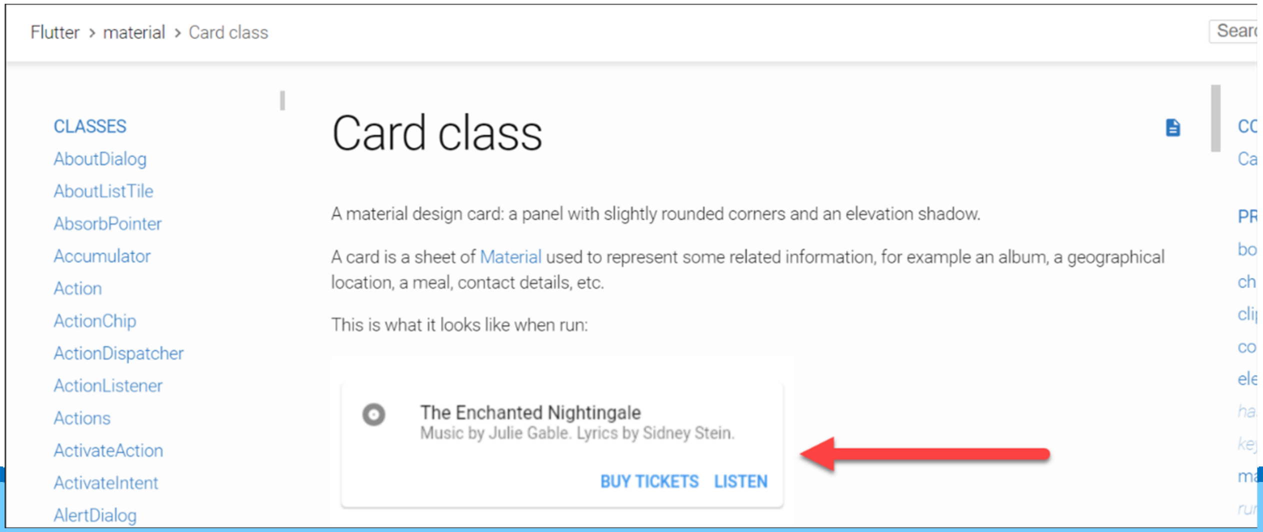

I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day
I will practice my modeling technique 2 hours every day

Using Cards

Displaying data in a `Card()` to make it look better – using a function to compose the card

Creating a function

- We're now creating a function that returns a Widget.
- This can then be used in our .map( ) function
- There *are* alternative ways, but this way you'll learn to be flexible and compose a widget tree
- api.flutter.dev/flutter/material/Card-class.html



Card class

A material design card: a panel with slightly rounded corners and an elevation shadow.

A card is a sheet of Material used to represent some related information, for example an album, a geographical location, a meal, contact details, etc.

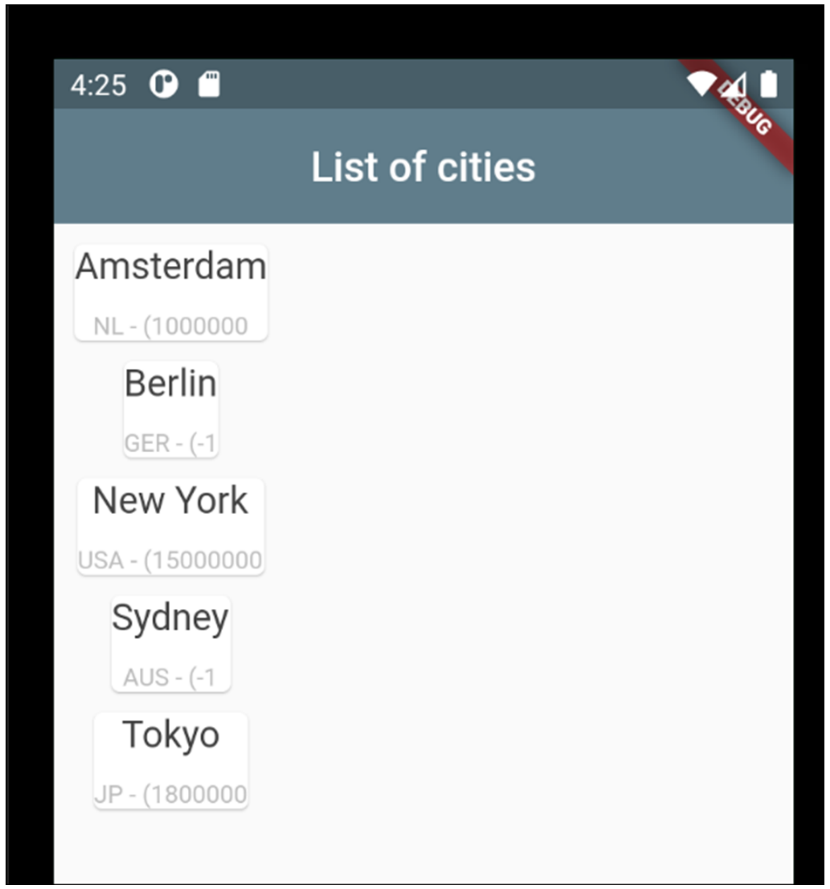
This is what it looks like when run:

1st approach

```dart
// We call this function for each city and it returns a Card widget with the
// properties of the city inside.
Widget cityTemplate(City city) {
  return Card(
    margin: EdgeInsets.fromLTRB(10, 10, 10, 0),
    child: Column(
      children: <Widget>[
        Text(city.name, style: TextStyle(
            fontSize: 18,
            color: Colors.grey[800]
        )),
        SizedBox(height: 12,),
        Text('${city.country} - (${city.population}',
          style: TextStyle(
            fontSize: 12,
            color: Colors.grey[400]),
        )
      ],
    ),
  );
}
```

Call the function

```
cities.map((city) => cityTemplate(city)).toList(),
```


4:25

DEBUG

List of cities

Amsterdam
NL - (1000000

Berlin
GER - (-1

New York
USA - (15000000

Sydney
AUS - (-1

Tokyo
JP - (1800000

It works, but it doesn't look really good at the moment.

Solution: add more lay-out stuff to the `Card()`.

2nd approach

- Add Padding( ), adjust fontSize and stretch out cards to fill the entire width of the column

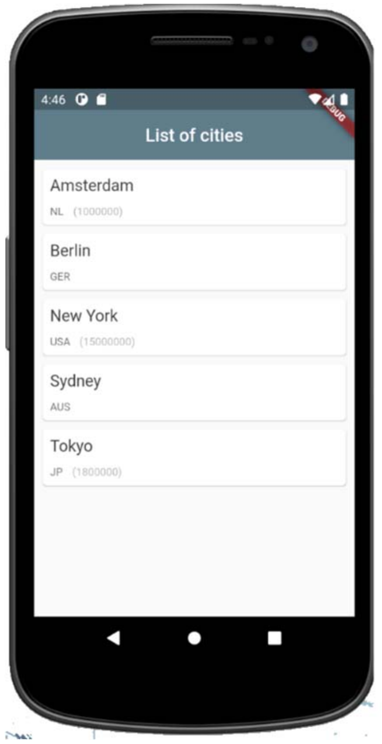
```
child: Padding(
  padding: const EdgeInsets.all(8.0),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.stretch,
    children: <Widget>[
      Text(
          city.name,
          style: TextStyle(fontSize: 18, color: Colors.grey[800])),
      SizedBox(
        height: 12,
      ),
    …
)
```

Problem – conditional rendering

- We only want to render the `population` (which is an optional property) if there is one (i.e. it is not -1)
- Solution: use the `if()`-statement *inside* the widget tree
- Also: wrap the line in a `Row()` to render multiple `Text()` widgets
- Notice: the (weird) way to group multiple widgets below the if-statement with …[ ]

Conditional rendering

```
Row(
  children: [
    Text(
      '${city.country}',
      style: TextStyle(fontSize: 12, color: Colors.grey[600]),
    ),
    // conditional rendering of a part of the widget tree.
    if (city.population != -1) ...[
      SizedBox(
        width: 10,
      ),
      Text(
        '(${city.population})',
        style: TextStyle(fontSize: 12, color: Colors.grey[400]),
      )
    ]
  ],
```


4:46

List of cities

Amsterdam
NL (1000000)

Berlin
GER

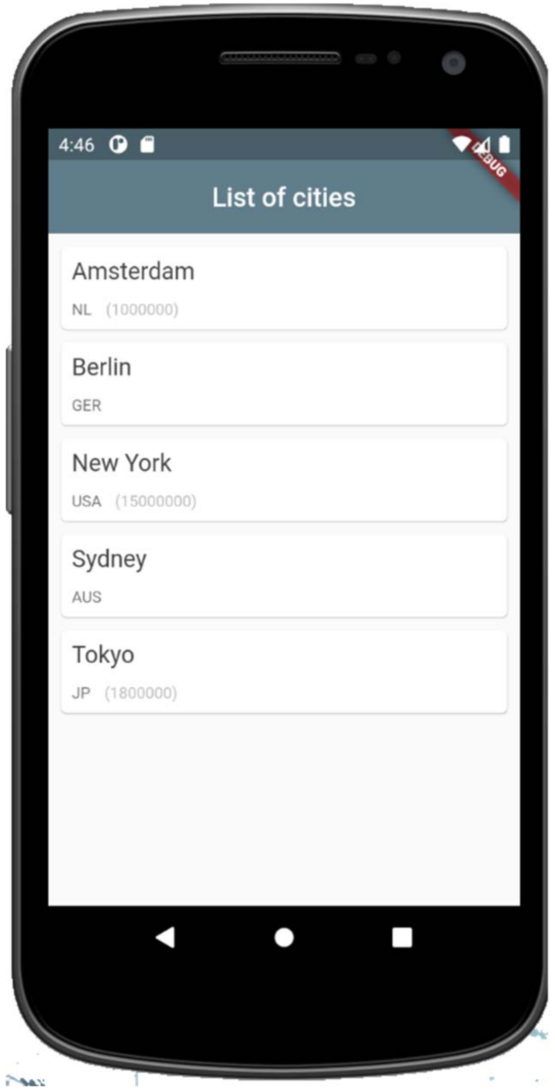
New York
USA (15000000)

Sydney
AUS

Tokyo
JP (1800000)

Much better!

Workshop


4:46

List of cities

Amsterdam
NL (1000000)

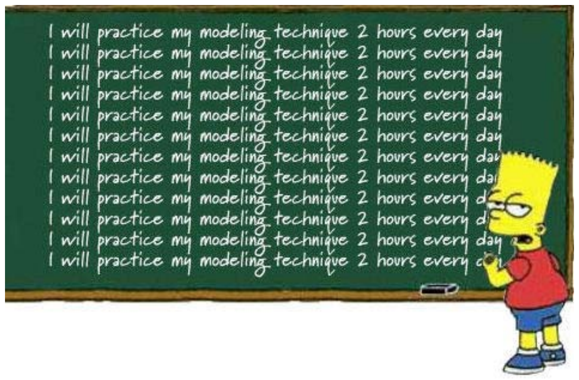
Berlin
GER

New York
USA (15000000)

Sydney
AUS

Tokyo
JP (1800000)

- Display your data in a `Card()`
- Extract the functionality into its own function
- Make sure it renders correctly, using conditionals and properties
- Optional: spice up your card with extra text, images, and so on.



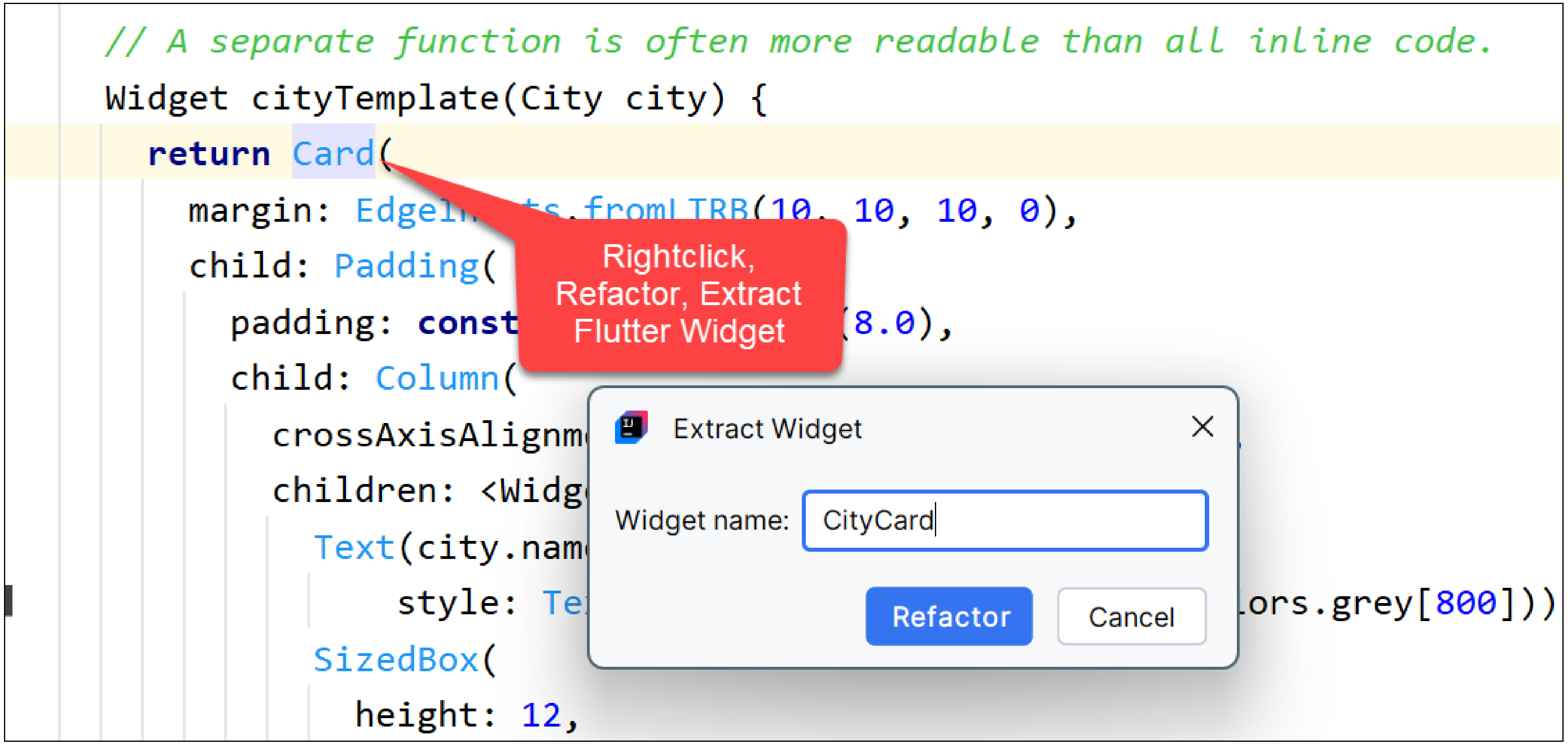
Extracting Widgets

Extracting functionality into their own, reusable widgets

Extracting reusable custom code

- The widget becomes rather large. We can extract the content of the `templateCard()` function to its own widget
- Use IntelliJ/Android Studio for that
- Panel `Flutter Outline`, rightclick the `Card()` that needs to be in its own widget.

Extracting a Widget



```
// A separate function is often more readable than all inline code.
Widget cityTemplate(City city) {
  return Card(
    margin: EdgeIn    ts.fromLTRB(10, 10, 10, 0),
    child: Padding(
      padding: const                    (8.0),
      child: Column(
        crossAxisAlignm
        children: <Widg
          Text(city.nam
              style: Te                          lors.grey[800]))
          SizedBox(
            height: 12,
```

Rightclick, Refactor, Extract Flutter Widget

Extract Widget

Widget name: CityCard

Examining the new widget

- Android Studio created a new `StatelessWidget` for us
- You can delete the constructor for now

```
class CityCard extends StatelessWidget {

  @override
  Widget build(BuildContext context) {
    return Card(
      margin: EdgeInsets.fromLTRB(10, 10, 10, 0),
      child: Padding(
        padding: const EdgeInsets.all(8.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.stretch,
        …
    );
  }
}
```

```
Widget cityTemplate(City city) {
  return CityCard();
}
```

Passing in the correct city

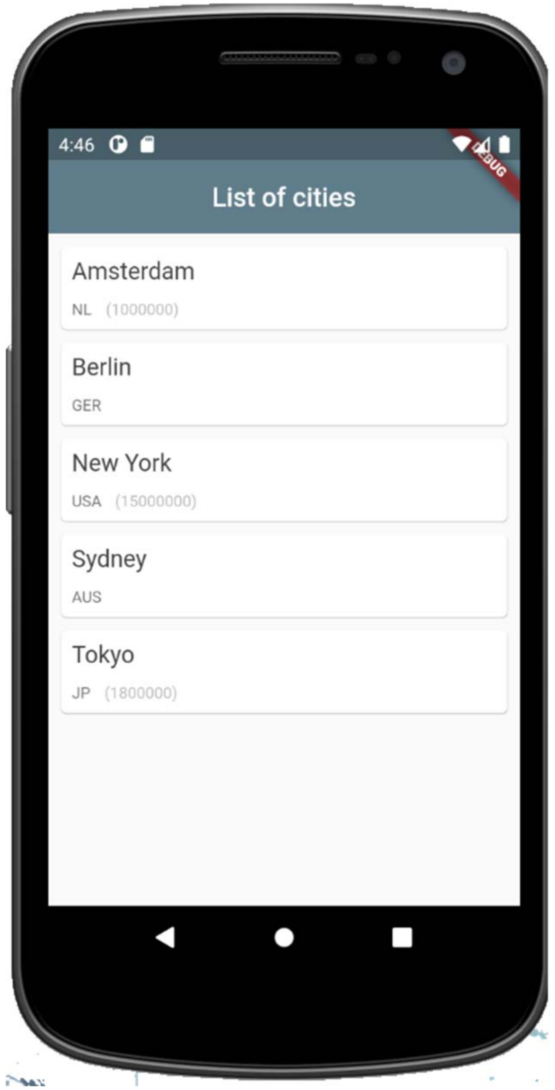
- Create local variable `final City city;`
- Add a constructor with named parameter
- `CityCard ({this.city})`

```
class CityCard extends StatelessWidget {

  final City city;
  CityCard({this.city});
  …
}
```

Final step – extract to file

- Of course we can now extract the `CityCard()` widget to its own file, to improve reusability
- Cut/paste to new file `CityCard.dart`.
- Don’t forget to import that new widget in `CityList.dart`.

Workshop


4:46

List of cities

Amsterdam
NL (1000000)

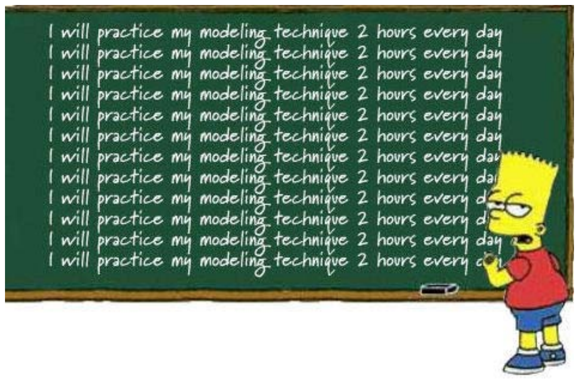
Berlin
GER

New York
USA (15000000)

Sydney
AUS

Tokyo
JP (1800000)

- Extract your function to a widget
- Make sure the application still runs
- Call the widget directly from your `.map()` statement
- If this works, extract the widget to its own file and import it.



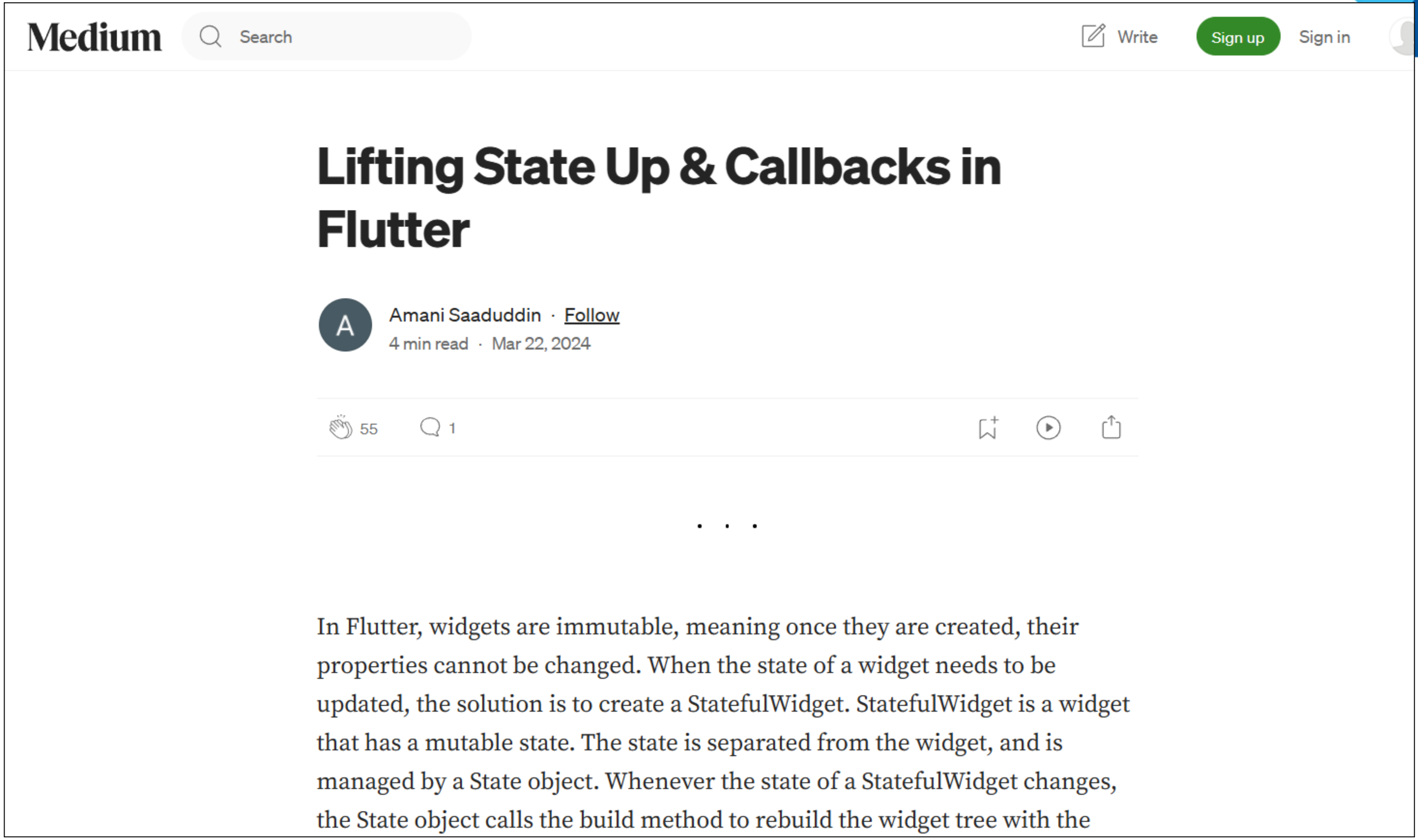
Passing functions as parameters

What if we want to retrieve functionality from a stateless widget?

Passing functionality

- Let’s say we want to delete cities from our array
- We can’t do that in the `CityCard.dart` file
  - It’s a `statelessWidget!`
  - It has no access to the array itself
- However, we *can* pass a function down to that Widget, that deletes the city in the parent widget
  - Again – much like how React handles this.

Design pattern: “Lifting state up”



Lifting State Up & Callbacks in Flutter

Amani Saaduddin · Follow
4 min read · Mar 22, 2024

. . .

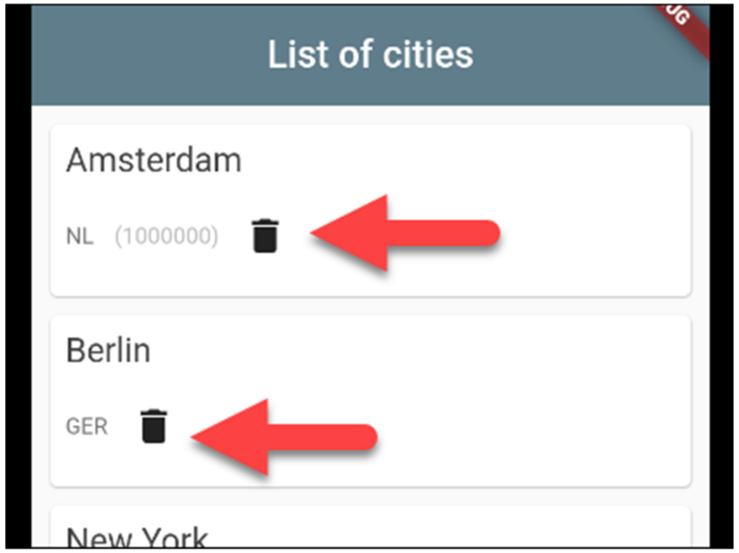
In Flutter, widgets are immutable, meaning once they are created, their properties cannot be changed. When the state of a widget needs to be updated, the solution is to create a StatefulWidget. StatefulWidget is a widget that has a mutable state. The state is separated from the widget, and is managed by a State object. Whenever the state of a StatefulWidget changes, the State object calls the build method to rebuild the widget tree with the

https://medium.com/@amani9920/lifting-state-up-callbacks-in-flutter-d36a5a3319f8

Step 1. Create UI to delete a city

- Extend the `CityCard.dart` file with a button or icon to delete the item

```
IconButton(
  onPressed: (){}, // to be filled in
  icon: Icon(Icons.delete),
),
```



Step 2. Create function to delete City

- In the `CityList.dart` file:

```
// function to remove a city from the array
void _deleteCity(city){
  setState(() {
    cities.remove(city);
  });
}
```

And, passing extra parameter:

```
children: cities.map((city) => CityCard(
  city: city,
  delete: _deleteCity
 )
).toList(),
```

Step 3. Create additional property on Card

In `CityCard.dart` file, add extra `final` property

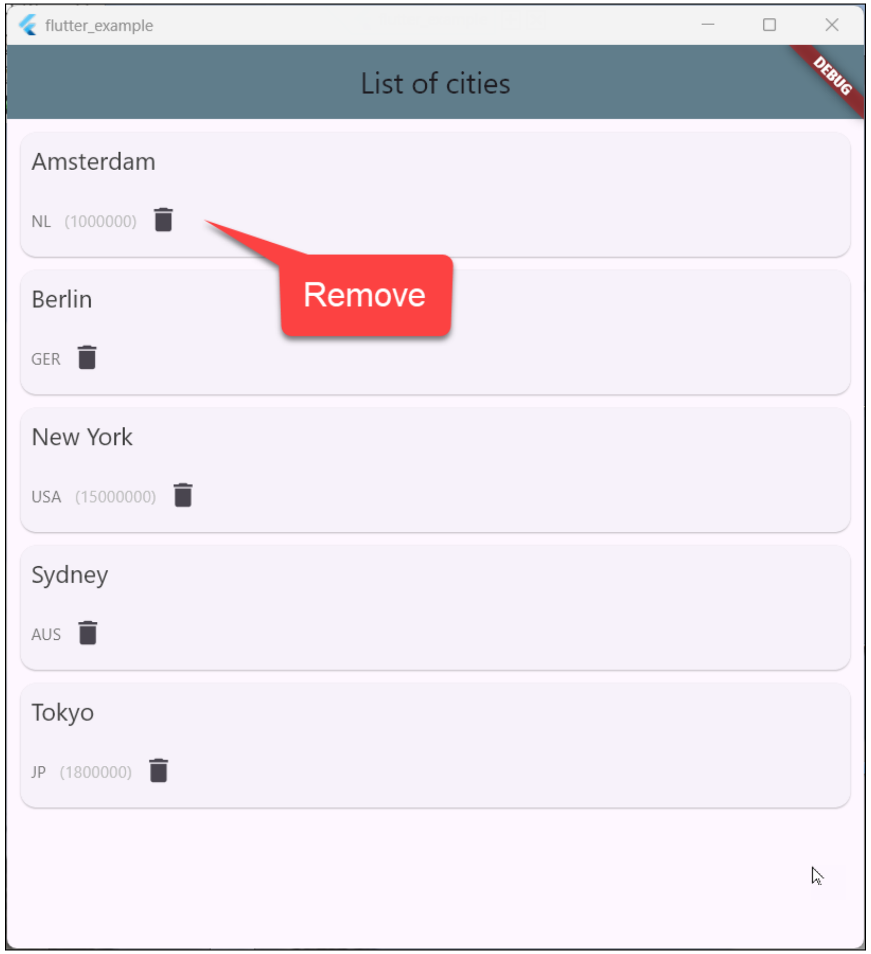
(which is the function that you pass in)

```
// use the 'final' keyword
final City city;
final Function delete;

// constructor - receiving the city and removal function.
CityCard({required this.city, required this.delete});
```

```
// Button to delete the city
IconButton(
  onPressed: () => delete(city),
  icon: Icon(Icons.delete),
),
```

Note the ( ) => ... notation. This is *delayed execution*. Try what happens if you omit it!

Result


flutter_example

List of cities

Amsterdam

NL (1000000)

Berlin

GER

New York

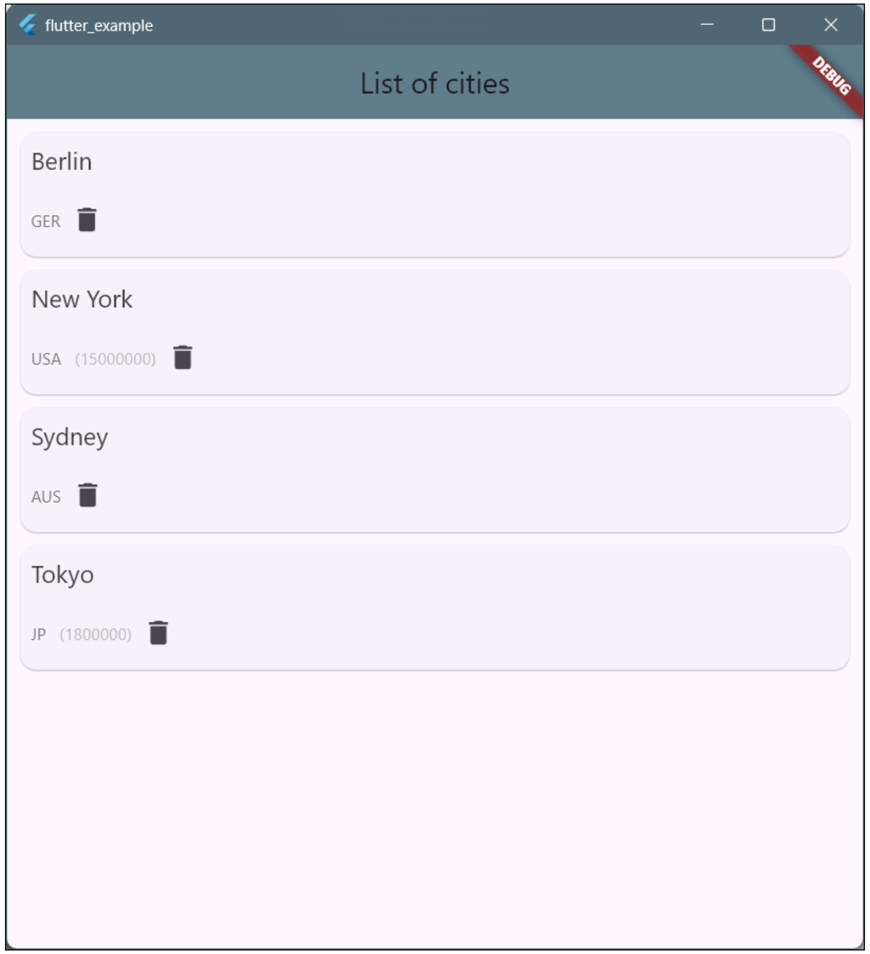
USA (15000000)

Sydney

AUS

Tokyo

JP (1800000)


flutter_example

List of cities

Berlin

GER

New York

USA (15000000)

Sydney

AUS

Tokyo

JP (1800000)

Workshop

- Create a Delete (or other) function for your widget
- Pass the function as a parameter from the parent- to the child widget.
- Make sure the application still works
- Example: `../_250-passing-functions`