4A Value widgets

We've got some basic widgets created. In this lab we're going to give them some content and learn some cool other lessons along the way. As you're building widgets, they'll look pretty ugly. Don't worry about their appearance nearly as much as their content. We'll pretty them up later.

We'll start out with an easy one...

FABs look great with an Icon

1. Edit PickSeats.

2. Add a FloatingActionButton (a FAB for short). Set the icon to a shopping cart. Later on we'll use this button to navigate the user to their cart.

3. Give your new FAB an onPressed event that just print()s something.

4. Test out how it looks by making PickSeats the startup widget in your main app.

5. Let's do the same with the Checkout widget. Add a FAB with an icon to represent "pay". Like a currency icon or something. You can decide what looks best.

6. Test your Checkout widget like you did PickSeats.

Fleshing out the Landing widget

7. Create a new folder called assets. Ask your instructor to provide you with a company logo file called daam.png. Put this file in your new folder.

8. Tell the project about this file by putting it pubspec.yaml kind of like this:

```
assets:
    - assets/daam.png
```

9. Edit Landing.dart.

10. Add the Dinner-and-a-movie logo at the top as an Image.
    Hints:
    - Use Image.asset()
    - Provide a height. 75 would be a good size.
    - Provide a fit so it will maintain the right aspect ratio.

11. Put the business name "Dinner and a Movie" in a Text().

12. Have another Text() that says " "Tap a movie below to see its details. Then pick a date to see showtimes."

13. Ensure that the DatePicker widget and the FilmBrief widget appear next.

14. Run and test Landing, making sure that everything shows up.

Reading data and writing it to state

At this point we know how to show a widget. We know how to put widgets inside of widgets, whether built-in widgets or custom widgets that you create. In other words we know how to compose widgets.

Sure, we could continue with the rest of these steps asking you to hardcode values but that would be time-consuming and not very realistic. We're trying to simulate real-world as much as possible. So we're going to give you some more pre-written code that will read a film and some showings from our API server. Explanations will be coming later in the course.

15. Edit FilmDetails. Add these imports at the top:

```
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'state.dart';
```

16. FilmDetails needs some properties. Put these in the _FilmDetailsState class:

```
Film film = Film();
DateTime selected_date = DateTime.now();
List<dynamic> showings = [];
```

17. We'll load the film and the showings when the widget first starts up. Add an initState method to the _FilmDetailsState class:

```
@override
void initState() {
  super.initState();
  film = context.read(selectedFilmProvider).state ?? Film()
    ..id = 1;
  selected_date = context.read(selectedDateProvider).state ?? DateTime.now();
  fetchFilm(id: 1).then(
      (f) => setState(() => context.read(selectedFilmProvider).state = f));
  fetchShowings(film_id: film.id, date: selected_date)
      .then((s) => setState(() => context.read(showingsProvider).state = s));
}
```

18. Look for the build method in FilmDetails. Add these two lines:

```
film = context.read(selectedFilmProvider).state ?? Film();
showings = context.read(showingsProvider).state;
```

19. Still in the build method, print out film and showings and make sure you're getting some data. If so, you can move on to the real exercise.

Making the data show in FilmDetails

Now that we're reading in some realistic data, let's work on displaying it.

20. Show the movie poster using Image.network.
    Hint:
    - When you test this out, you'll see a big poster which is awesome! But the screen has overflowed -- you can't see everything. To fix that, wrap your outermost widget with a SingleChildScrollView().
    - This widget wasn't taught in the book or lecture. That's okay. In the real world it's normal to come across some obscure widget that you must look up. This is a good opportunity for you

to practice looking at the Flutter documentation to learn a new widget. Go look up a SingleChildScrollView at flutter.dev and see if you can get your arms around it.

21. Put your instance of ShowingTimes after the image.

22. Show the movie title.

23. Show the tagline.

24. Show the movie homepage.

25. Show the movie overview.

26. Add a Text() that says "Rating: X/10 Y votes" where X is the movie rating and Y is the number of votes cast.

27. Add an ElevatedButton that says "Done" and when pressed, prints something to the debug console.

28. Run and test. When you can see all of that great information, you can move on.

ShowingTimes

29. Edit the ShowingTimes widget. Make its constructor receive a film object, a selected_date, and an array of showings.
    Hints:
    - Make these properties all required
    - Create class properties to match. You probably want to mark them as final.
    - *film* should be a Film
    - *selected_date* should be a DateTime
    - *showings* will be a List<Showings> if you created that class earlier or a List<dynamic> if not.

30. Format the date string:

```
String selectedDateString =
    new DateFormat.MMMMEEEEd().format(widget.selected_date);
```

31. Give it a text like so: Text("Showing times for $selectedDateString for ${widget.film.title}"), You don't need to do anything with showings yet. We'll need that later in the Gestures lab.

32. As things now stand you should be able to test ShowingTimes by looking at FilmDetails. Go ahead and do so.
    Hints:
    - Since the constructor of ShowingTimes now requires three parameters, you'll need to provide those values in FilmDetails.

How to show multiple instances of a widget

It is very common to show multiple instances of the same widget in a parent. That's what we're going to do next.

Let's read all of the films from our data server and show each one to our user, then allow him/her to scroll through all of the films. We'll start by reading from the server. We'll dig into the details of how all this works later in the course.

33. Edit Landing. Add these imports at the top:

```
import 'package:flutter_riverpod/flutter_riverpod.dart';
import 'state.dart';
```

34. Add this to the State class:

```
@override
void initState() {
  super.initState();
  fetchFilms().then((films) => context.read(filmsProvider).state = films);
}
```

35. And add this to the top of the build() method:

```
List<Film> films = context.read(filmsProvider).state;
```

36. Find where you had one FilmBrief():

```
FilmBrief(),
```

and replace it with multiple FilmBrief()s, one for each entry in our new films array.

```
Column(
  children: films.map((f) => FilmBrief(film: f,)).toList(),
),
```

Hints:

- Yes, that is a Column() within a Column().
- The .map() method returns a new array based on an original array (films in this case). It receives a transformation function.

37. We're now passing a Film object into FilmBrief here so we'll need to change FilmBrief before our app will compile or run. Edit FilmBrief to receive a Film object in its constructor. Go ahead and print() it out just to make sure you have data and that data is being received into FilmBrief.

38. If you test this out, you've got multiple films showing up which is awesome! But the screen may have overflowed. A SingleChildScrollView() widget is one of several ways to fix that.

Displaying data in FilmBrief

The FilmBrief widget will be important. Its mission is to show brief information about a film, hence the name.

39. In FilmBrief, get the movie poster to show up. Give it a height of 100.

40. Show the title in a text

41. Show the tagline in a text.

Congratulations, you're finished! Take a well-earned break.