AB Dart's Unexpected things

Now that you know about the unique features of Dart, let's refactor your existing classes using the best practices. This lab leaves a lot of wiggle room -- you can make choices on which Dart features you want to use and which you don't.

Conciseness of code

1. Go through your code. Look for anywhere you might have used "this.". Remove "this."

2. Again, look through your code for any instantiations. If you used the word "new", get rid of it.

3. See any opportunities to use var? Change a variable or two to var. Convince yourself that they're still strongly typed.

4. How about const? Look for some variables you've created that will never change. If you mark them as const, you'll make your code faster.

5. Change some of your positional parameters to named parameters in constructors and function definitions.

Null safety

The issue of null safety may have already been solved for you. As you were writing, you probably saw the compiler complain about certain variables needing to be initialized.

6. If you see some properties that make sense to be null, go ahead and put a "?" after the type. This marks them as nullable. For example, maybe you think that not all films will have a home_page. When you do, the IDE is likely to complain about the use of that variable.

7. On the other hand if there are any that were previously marked as nullable and you remove the "?", you'll get even more warnings requiring you to provide a fallback value or mark it as required since it can't be null anymore.

8. Add default values where it makes sense.

Prepping to read API data

As mentioned, we're trying to approximate real world in these labs and we want to get you reading data from an API server ASAP. In this next section, we're asking you to add some pre-written code that will be explained later in the course. Please don't feel like you need to understand all of this. Just trust us for a little while; it'll all become clearer when we cover state management and reading from APIs later.

9. Edit main.dart. At the top, add this line:

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

10. Find the main function. Wrap MainApp in a ProviderScope widget. Change this:

```
void main() {
  runApp(MyApp());
}
```

to look like this:

```
void main() => runApp(ProviderScope(child: MyApp()));
```

11. Find MyApp's build method. It currently returns a MaterialApp Widget. Wrap that return with a Consumer Widget. Change this:

```
Widget build(BuildContext context) {
  return MaterialApp(
    title: 'Dinner And A Movie',
    theme: ThemeData(
      primarySwatch: Colors.blue,
    ),
    home: MyHomePage(title: "Some title"),
  );
}
```

to look like this:

```
Widget build(BuildContext context) {
  return Consumer(builder: (context, watch, _) {  // <-- Add this ...
    watch(filmsProvider).state;                   // <-- ... and this ...
    watch(showingsProvider).state;                // <-- ... and this ...
    watch(selectedFilmProvider).state;                // <-- ... and this ...
    watch(selectedDateProvider).state;                // <-- ... and this ...
    films = films.length == 0 ? [Film()] : films; // <-- ... and this ...
    return MaterialApp(
      title: 'Dinner And A Movie',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: "Some title"),
    );
  }); // <-- ... and this
}
```

12. Once you've got all that written, you won't see any change but you should be able to compile and run your app. Give it a try.