AB Dart's Unexpected things

Now that you know about the unique features of Dart, let's refactor your existing code to be more “darty” 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." Maybe you did this in _incrementCounter() in main.dart?

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. Definitely try this where you’re calling makeConsecutiveDates().

4. Convince yourself that they're still strongly typed by trying to assign a hardcoded value of the wrong type to it. The IDE should complain and refuse to compile.

5. 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.

Learning from some example code

6. Look in repository.dart. Find the getBaseUrl() function. Notice how we’re building certain variables up from hardcoded values along with the *port* variable. Do you see the $ sign, how we’re using interpolation? Pretty clean, right?

7. Bonus! If you didn’t create an overridden Film.toString() method no big deal. But if you did, change it to use string interpolation instead.

8. Look for the fetchFilms() function. See how we’re interpolating with the url variable? But this time we’re calling the getBaseUrl() function so the interpolation isn’t done on a simple variable. In this case, the curly braces are needed.

9. Same with fetchShowings. Look for the interpolations there and the necessity of using curly braces.

10. There’s a nice example of an arrow function in fetchFilms. The List.map() function converts a List of one thing into a list of another thing. The .map() method receives in as its one parameter a **function**, a convertor function in fact. Well, the arrow is defining a function. In this case, it receives in a film which is of type dynamic and returns a strongly-typed Film object. This function is run on each member of the films list thereby converting the list of dynamics into a list of bona-fide films!

11. In that same convertor arrow function, please note the use of the cascade operator, the ".." operator. We’re instantiating a Film class and setting each object property to a value from the dynamic. Cascade operators; a super useful feature of Dart.

Function parameters

12. Locate makeConsecutiveDates() in your code. Convert both of those positional parameters into named parameters. (Hint: use curly braces). Notice that you'll either have to 1) make the parameters nullable, 2) mark them as *required* or 3) give them a default value.

13. Give howMany a default value of 5. (Hint: `howMany=5` in the argument list).

14. Try to do the same with startDate, setting it to DateTime.now(). How'd that work for you?

15. Mark startDate as required sort of like this:
```
List<DateTime> makeConsecutiveDates(
    {int howMany = 5, required DateTime startDate}) {
```

16. If you try to run and test, you'll see that any calls to makeConsecutiveDates() must now provide named parameters and not positional ones. Change your call to makeConsecutiveDates() to fix the problem.

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.

17. Edit your Film class in the film.dart file. If you see some properties that make sense to be null, go ahead and put a "?" after the type. This marks them as nullable. And if something is nullable, you may also want to remove the default value. For example, maybe you think that not all films will have a homepage. When you do, the IDE is likely to complain about the use of that variable.

18. 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.

Are we having fun yet? Hopefully this lab is exposing some Dart features and giving you a feel for Dart, even though it may not all make total sense yet. Hang in there, it’ll land better with every line of Dart you read and write!