Todo App – Data and State Management (with provider)

This tutorial builds upon the previous application built in session 4 for the Todo App. You need to complete this part before attempting to complete this session.

Task 1 – Todo List Data Model

Previously we pilled our data into one place, the main stateful widget. This worked because we only really changed our data in one place. Now we want to add a few new features:

- A completed counter.
- The ability to complete Todo Tasks
- Move to the provider model to manage our state.

Steps

1. Create a new folder under your lib folder called models.
2. Create a new dart file called todo_list.dart.
3. Inside create a class called TodoList that extends ChangeNotifier
    a. This will need to import a package that includes this class (Foundation/Widgets/Material)
4. This is our data model, which holds our data which is a list of Todo's. Create a final private List<Todo> called _todos and initialise it empty.
5. Add: UnmodifiableListView<Todo> get todos => UnmodifiableListView(_todos);
    a. This is an immutable returned list for presenting in the UI.
6. Int get todoCount => _todos.length;
    a. A total or count property getter

Exercise 1 – Complete the class by adding the necessary functions for:

- Add(Todo todo);
- RemoveAll();
- Remove(Todo todo);

Inside each of these functions, at the end of the function call the NotifyListeners(); function. This is what triggers the update in the UI by the consumer objects.

Remember this class is the Data Model for the whole App. It defines the data state which reflects in each widget throughout the app.

Now add the final function, which is a little trickier, for updating a model in the list with new data. It looks like this:

```
void updateTodo(Todo todo) {
  Todo listTodo = _todos.firstWhere((t) => t.name == todo.name);
  listTodo = todo;
  notifyListeners();
}
```

Task 2 - Todo List Item Widget

Next, we're going to create a new widget object that represents one item in our list. It will take a single Todo item and populate and manage everything about that item.

Step

1. Copy the code from inside the list builder widget that you return nothing from the item builder. You should only be left with this or something close to this:

```
return ListView.builder(
    itemCount: list.todoCount,
    itemBuilder: (BuildContext context, int i) {
      return
    }); // ListView.builder
```

2. Create a new folder for your widgets, this can be called anything really; views is fine.
3. Inside create a new dart file called todo_widget.dart
4. Create a new flutter stateful widget (Recommend the shortcut to create all the parts "st") called TodoWidget.
5. Inside the state object's build method, paste in the single item builder code which represents one todo in the list.

```
  _TodoWidgetState createState() => _TodoWidgetState();
}

class _TodoWidgetState extends State<TodoWidget> {
  @override
  Widget build(BuildContext context) {
    return Container(
```

6. Inside the TodoWidget class update it to have a new todo object, and a required parameter in its constructor:

```
class TodoWidget extends StatefulWidget {

  final Todo todo;
  const TodoWidget({Key? key, required this.todo}) : super(key: key);
```

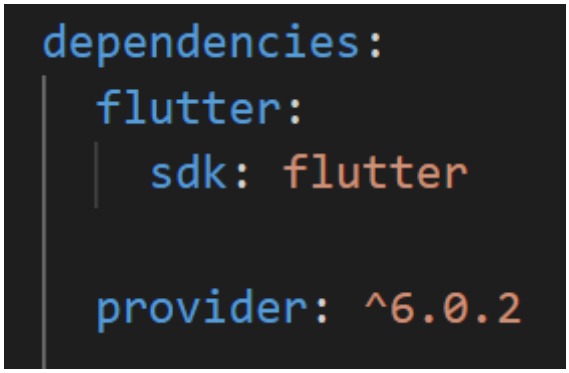
7. Now we need to update any references to the TodoList. One thing to note here is the data is passed through to the widget, not the state object which makes it inaccessible.
8. Well not directly! We can use the widget.todo to retrieve the reference from the widget within the state object.
9. At this point the Todo Widget does not contain any state changes so we can leave it as is, we'll come back and update it with code for toggling the completed state on and off.
10. We also have the error for the ListView.builder but we'll fix that soon.

Task 3 – Adding a Provider and Consumer

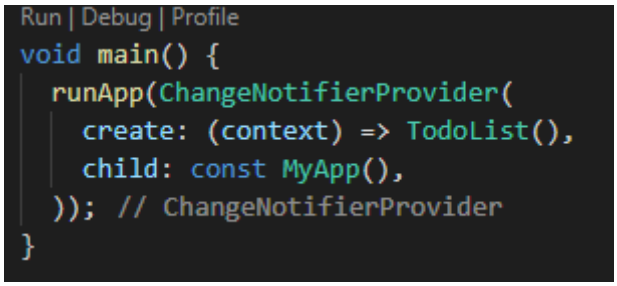
The final steps are to link this all together. We now have a state object (or ChangeNotifier) which will let us know when our data model changes. But this needs to be registered. Remember it must be high enough to cover any widgets that will interact with it. As later we'll be using it within the appbar, I'm going to add it directly around the TodoApp() definition. Then we need to add each consumer where we're pulling the data from.

Steps

1. To add the ChangeNotifierProvider we need to add the provider state package. Go to the pubspec.yaml file in the main directory.
2. Under dependencies, in line with flutter add "provider: ^6.0.2" without the quotes. Indenting is important here.



3. Now, Inside the runApp() function call add a new ChangeNotifierProvider. It needs two parameters, the create (The ChangeNotifier youre going to be providing) and the child (The app).
4. It should look something like this:



5. Now we can wrap our listview in a Consumer class.
6. Down in the HomePageState build method, where the listview builder is, wrap this in a Consumer<TodoList>().
7. This widget takes a builder that defines three parts:
    a. Context, model reference, the child
8. The builder function then returns the widget using the context and reference as if it has been fetched.
9. Inside the ListView builder we can use the model reference to access the data.
10. The final code including consumer and model access should look like this:

```
child: Consumer<TodoList>(builder: (context, model, child) {
  return ListView.builder(
      itemCount: model.todoCount,
      itemBuilder: (BuildContext context, int i) {
        return TodoWidget(todo: model.todos[i]);
      }); // ListView.builder
}), // Consumer
), // Center
```

Exercise 2

Add an app bar text widget that displays the Todo list total not completed. This is done using the app bar actions. To get the total not completed you can also use the todos.where() function shown in the Model's update() function.

Note: At this point we cannot change the value of the Todo completed status, all Todo's added should be counted as the default value is false for completed.

Exercise 3

For this exercise, attempt to update the list using the updateTodo() function in the model. This needs to be completed from the todo_widget.dart class. I recommend using a checkbox with an onchanged function call. Notes:

- Use Checkbox and onChanged.
- Remember to set the value of the todo item to the new value from the checkbox onChanged.
- Also make sure to call the setState() function to update the widget itself.
- In this case we need to call a function of the model without retrieving the data model itself. This can be done using the provider class of() method like this

```
Provider.of<TodoList>(context, listen: false).updateTodo(widget.todo);
```

- The listen: false parameter is essential to using the provider model like this when not needing the model as a point of reference.