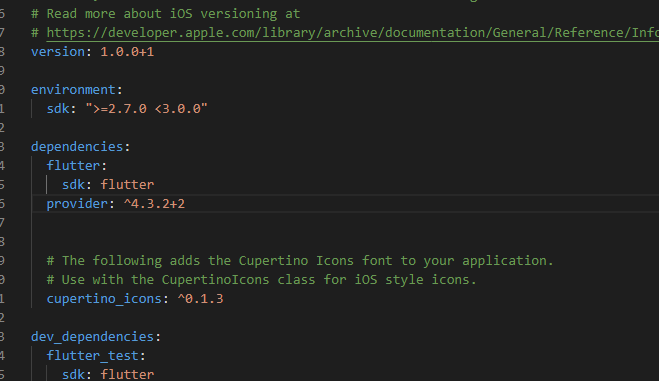
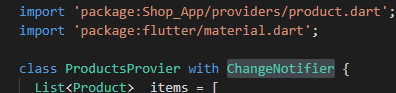
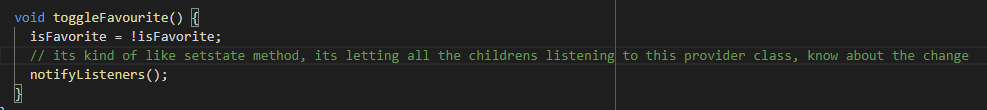
* Importing the provider package



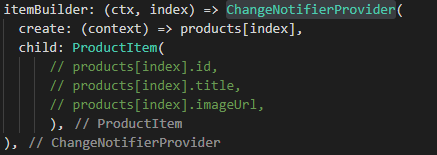
* Next step is creating the provider class. We do this by **Mixin**(it shares property with the class but don’t define the type like inheritance or interface) the created class with **ChangeNotifier.**



* Inside the class we can make methods which we can do things and then use the **notifyListeners()** , this will let all the widget who are listening to this provider update and rebuild. Its kind of similar to **setState()** .



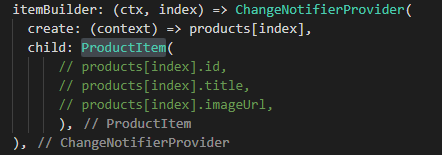
* Now that the provider class is all set, we need to wrap the topmost parent widget of the all the child widget that want to use the notifier with **ChangeNotifierProvider()** widget. Here we need to specify **create** (**build** for provider package version below 4.0.0) and create and instance of the provider class. Now we can specify the child for this widget and every child listening to this will get notified of the changes made to the provider class



* Final step is to use the provider class inside the child widget where it needs. For that we can use the **Provider.of<T>(context)** generics to get the data. This will return the instance of the provider of the nearest parent widget where it can find the provider instance. For example,



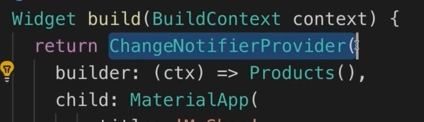
It will get the **Product** provider instance from its parent where it was instantiated.

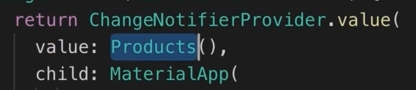


One more thing, if we just interested in the data, but don’t want the widget to change once something is changed on the data, we can set the **listen** to **false (**by default its set to **true).**



* An alternative to the **ChangeNotifierProvider(),**  we can use the **ChangeNotifierProvider.value().**  In this case we don’t need to use build/create. Instead we use the **value.** For example,





* There is an important feature we must remember. We must use the **ChangeNotifierProvider.value** if we use it to something that is part of a list or grid. Because the widget is recycled by flutter and the data only changes. In these cases, the create/builder approach will be buggy cause the changed data will be attached to the provider

In short, whenever we reuse an existing product (for example, cycle through a list). We should use the .value approach. When we create a new instance of an object we should use the create/build approach.

* When we use **Provider.of<T>(context)** it will rebuild the whole widged if the provider data is changed. But we can improve on this with clever widget management. If we wrap the widgets that use the data with **Consumer<T>()**  and only that part will rebuild once the provider data is changed**.**

