Todo App – Hive Offline Binary Data Store

Introduction

Hive is an offline key data store that uses binary data to store basic data in any platform – including web. There a little bit of set up and the structure of hive might take a while to wrap your head around but once it falls into place. It's very easy to use. First, we need to update our Todo class to use the hive annotations and create an adapter so hive can easily translate the objects into binary data for storage.

Hive

Add the hive packages in the pubspec.yaml file like this (please note check the latest version for changes):

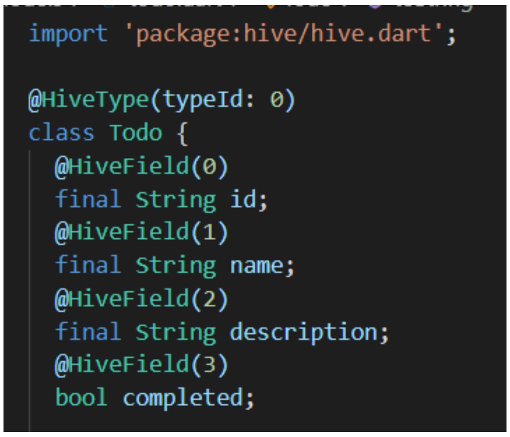
```
hive: ^[version]
hive_flutter: ^[version]
```

At the top of the todo.dart file, add the import we're going to use:

Import 'package:hive/hive.dart';

Now we can annotate our class for hive. There are two main annotations that use the '@' symbol proceeding it. HiveType and HiveField. One is for the whole class and identifies the type, so hive knows to use the correct adapter (We'll get there) and the field lets us fetch the correct binary data from the store to turn our objects into actual data.

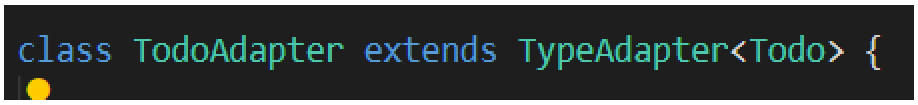
Above the class name and fields update them to be annotated like this:



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

@HiveType(typeId: 0)
class Todo {
  @HiveField(0)
  final String id;
  @HiveField(1)
  final String name;
  @HiveField(2)
  final String description;
  @HiveField(3)
  bool completed;
```

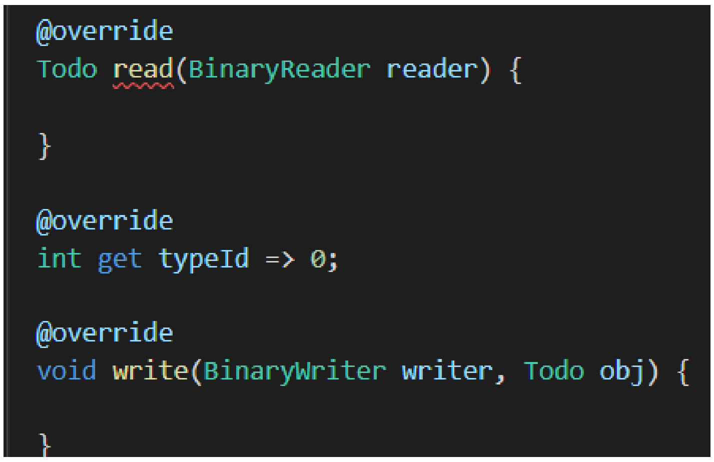
Note that the id's for each type (Class) and field (property) must be unique.

Next, we can create the adapter that allows Hive to translate the data. Inside the Todo.dart file under the class and outside its scope (The last curly bracket) add a new class definition. This is our TodoAdapter. It must extend a type-adapter of type Todo.



```dart
class TodoAdapter extends TypeAdapter<Todo> {
```

This class requires three overrides – Read, Write and typeId. To easily add this hover over the red squiggly line under Todo adapter and quick fix add overrides. Alternatively add the three methods like below:



```dart
@override
Todo read(BinaryReader reader) {

}

@override
int get typeId => 0;

@override
void write(BinaryWriter writer, Todo obj) {

}
```

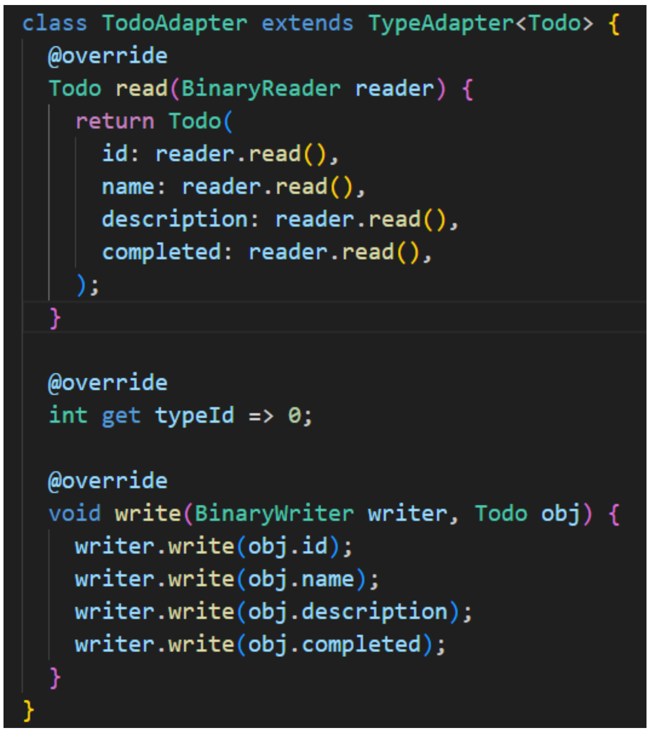
The typeId is done for you above, this should make the @HiveType value you set above for the class.

We can add the write method. We use writer.write(obj.name). By default, passing the value into this from the object also sets the field id from the class (For retrieving later).

For the read function we need to take the reader and return a new Todo object. This can be done by calling the reader.read() method. These will be read back in order of their write. Calling reader.read() will initially get back the id, then the second will be the name as it is in the write.

Return a todo by calling this reader.read method and passing each variable for the constructor.

Finally, you should end up with a class like this:

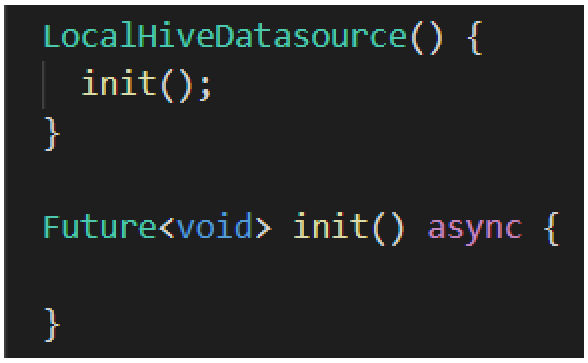


```dart
class TodoAdapter extends TypeAdapter<Todo> {
  @override
  Todo read(BinaryReader reader) {
    return Todo(
      id: reader.read(),
      name: reader.read(),
      description: reader.read(),
      completed: reader.read(),
    );
  }

  @override
  int get typeId => 0;

  @override
  void write(BinaryWriter writer, Todo obj) {
    writer.write(obj.id);
    writer.write(obj.name);
    writer.write(obj.description);
    writer.write(obj.completed);
  }
}
```

Local Hive Data Source

Create a new file for the local data source for hive. Give it a name with a similar pattern to the other classes. Extend the tododatasource class and add the implemented methods.

Add a new function called init() that's a future<void> this will allow us to async set up and await as the hive init process is also async and can take a while. Call this function from the localhivedatasource() constructor.



```
LocalHiveDatasource() {
  init();
}

Future<void> init() async {

}
```

This function has a few things to do register the adapter we just created so hive and convert it, open the box we're going to use (that's what hive call its tables/documents). And just to check, we'll also add a Todo list item with something in it.

The first call must be to initalise Hive for use by calling:

await Hive.initFlutter();

Next, call Hive.registerAdapter(TodoAdapter()); to add the adapter for hive.

Get the box using Box box = await Hive.openBox('todos_box');

Now we can call the box.put() method to add something to the box. As hive uses key value pairs, we are going to pass a name for the todo list (might be worth storing this as a const) and a list of todos with something in it. This is an async method, so we await the result.

The purpose of our offline databases is to store the data and retrieve it should the internet go down. To do this we do not need an add, update, or delete method, just get and all for offline (however eventually we can add those).

Exercise 1

Explore the hive package and create the All and Get methods of the local hive data source.