Abstract-Factory-BuildingTheme

Source
Github

Flutter Build Theme with Abstract Factory Method

Firstly it needs a page design such as also this page could connect to service. (I created this endpoint for this sample page)

- Background
    - App bar
      -Search Bar
      — Search Icon
      — Search Text
      — Microphone Icon
    - ListView
      — Product Card
    - TabBar
      — TabBar Icons

As a result of we need a color palette for using this project. If your design kit has a color palette, you can get all colors in design kits.

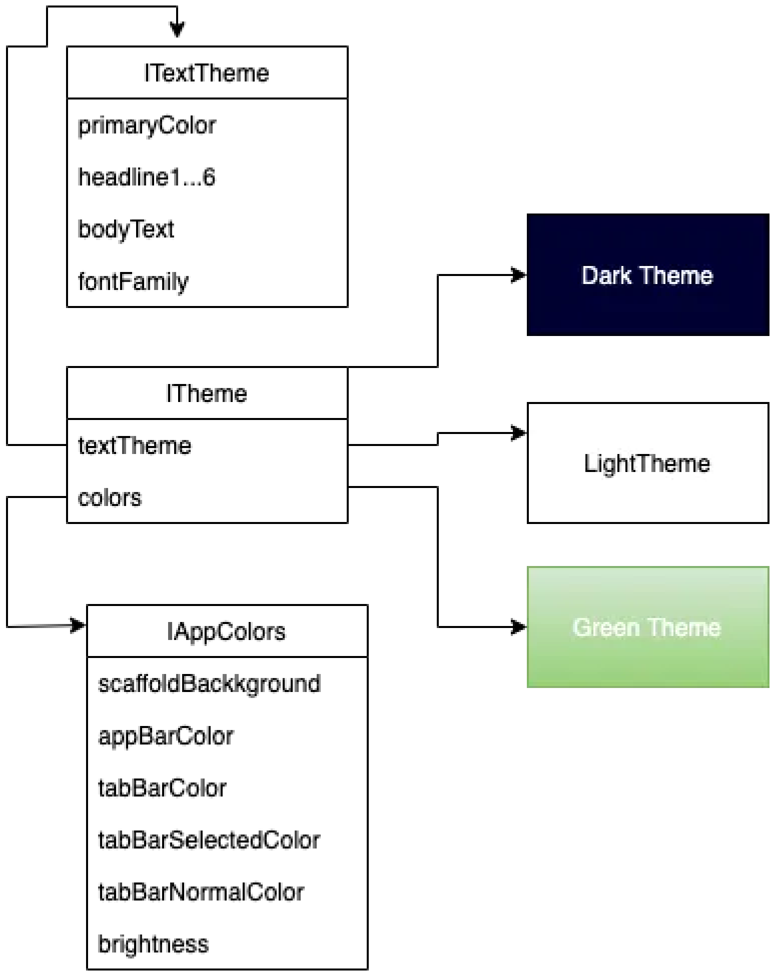
The project has to use this color palette in the event of need new widget. Actually, projects grow up easier owing to the theme manager. Finally, we ready for the hacking time thus we'll use both factory method pattern and page atomic design.

Hacking Time

Firstly, I prefer firstly write core features that’s why we don’t double work whenever code complete. I’m creating this list:

- ITheme abstract class for using different colors and styles.
- ThemeFactory class using to manage different themes from one point.

Factory design is one of the creational patterns. This pattern provides advanced objects due to the fact that clients don’t know anything. Now that pattern creates a custom object so the project can use this scheme.



Now we know what we need for this structure as we can go write an interface with both text and colors. This interface provides a central point of view so the project needs. Let’s write these points.

Text Theme Interface

Every project needs this as most usages point to text guidelines to the project. So we create base styles guideline after very easy to use from view. Sometimes we need to custom text styles as not mean you don't use current styles. We can use copyWith functions so view use like headline5 as well as can add a custom property like text color.

```
abstract class ITextTheme {

final Color? primaryColor;

late final TextTheme data;

TextStyle? headline1;

TextStyle? headline3;

TextStyle? headline4;

TextStyle? headline5;

TextStyle? headline6;

TextStyle? subtitle1;

TextStyle? subtitle2;

TextStyle? bodyText1;

TextStyle? bodyText2;

String? fontFamily;

ITextTheme(this.primaryColor);

}
```

If your project design has a kit, you can use zeplin tool. This tool gets all text styles in the style guidelines tab.

Color Theme Interface

It’s very important to point for projects because you know that color shows everywhere. So we how to manage more projects very control easier. Every project has a certain color schema and you have to use this scheme in your code. If you don't use schema and the project has a static color code, you won't add multi theme options additionally you cannot manage color problems.

```
abstract class IColors {

_AppColors get colors;

Color? scaffoldBackgroundColor;

Color? appBarColor;

Color? tabBarColor;

Color? tabbarSelectedColor;

Color? tabbarNormalColor;

Brightness? brightness;

ColorScheme? colorScheme;

}
```

I said like up to paragraph about zeplin. Again you can use this and you able to all color properties.

Abstract Factory Manager

Manager created for multi-interface. This manager will create a ThemeData instance for the project. You can create a new theme instance owing to this interface. This new theme only needs a color scheme etc.

```
abstract class ITheme { ITextTheme get textTheme; IColors get
colors;}
```

Yeah, it seems very simple and show useful for any project. Finally, we ready to use the core theme draw operation therefore the project can be able to declare a custom theme with this structure. Maybe, these theme interfaces can be improved more advanced version. Now it’s enough for this project.

Finally need to factory creator and we use this manager for the project theme

```
abstract class ThemeManager {

static ThemeData craeteTheme(ITheme theme) => ThemeData(

fontFamily: theme.textTheme.fontFamily,

textTheme: theme.textTheme.data,

cardColor: theme.colors.colorScheme?.onSecondary,

floatingActionButtonTheme: FloatingActionButtonThemeData(

foregroundColor: theme.colors.colors.white,

backgroundColor: theme.colors.colors.green),

appBarTheme: AppBarTheme(backgroundColor:
theme.colors.appBarColor),

scaffoldBackgroundColor: theme.colors.scaffoldBackgroundColor,

colorScheme: theme.colors.colorScheme);

}
```

I planed only specific areas as it project only has two pages as you know this sample. You have to create other areas both text style & color scheme area. Let's create custom themes with this structure and we’ll show this usage advantage.

Ligh Theme on Project

Actually, we have a structure and projects on how to create a light theme.

```
class AppThemeLight extends ITheme {

@override

late final ITextTheme textTheme;

AppThemeLight() {

textTheme = TextThemeLight(colors.colors.mediumGrey);

}

@override

IColors get colors => LightColors();

}
```

> Of course, Dark theme to create like this therefore just change style guidelines and the project can use directly. You can access the dark theme code here.

TextTheme light needs to base colors for draw text default color and Light Colors has created from zeplin styles.

```
class TextThemeLight implements ITextTheme {

@override

late final TextTheme data;

@override

TextStyle? bodyText1;
```

```
@override

TextStyle? bodyText2;

@override

TextStyle? headline1;

@override

TextStyle? headline3;

@override

TextStyle? headline4;

@override

TextStyle? headline5;

@override

TextStyle? headline6;

@override

TextStyle? subtitle1;

@override

TextStyle? subtitle2;

final Color? primaryColor;

TextThemeLight(this.primaryColor) {

data = TextTheme(
```

```dart
headline6: TextStyle(fontSize: 20, fontWeight:
FontWeight.normal),

subtitle1: TextStyle(fontSize: 16.0),

).apply(bodyColor: primaryColor);

fontFamily = GoogleFonts.arvo().fontFamily;

}

@override

String? fontFamily;

}
```

if we want to look at light colors theme instance, that shows this.
```dart
class LightColors implements IColors {

@override

final _AppColors colors = _AppColors();

@override

ColorScheme? colorScheme;

@override

Color? appBarColor;

@override

Color? scaffoldBackgroundColor;

@override
Color? tabBarColor;
@override
Color? tabbarNormalColor;
@override
Color? tabbarSelectedColor;
LightColors() {
appBarColor = colors.white;
scaffoldBackgroundColor = colors.white;
tabBarColor = colors.green;
tabbarNormalColor = colors.lighterGrey;
tabbarSelectedColor = colors.darkerGrey;
colorScheme = ColorScheme.light()
.copyWith(onPrimary: colors.green, onSecondary: colors.white);
brightness = Brightness.light;
}
@override
Brightness? brightness;
}
```

```
Sometimes need to prepared styles because don't have enough
style knowledge. At this time you could use a color scheme
instance for your project so you get the material color scheme
therefore add your custom business layer.
```

~~~
And light theme ready is to use. The project just needs the
theme factory method and you can write this class instance.
That’s okay for everything your project colors.
```dart
class MyApp extends StatelessWidget {

@override

Widget build(BuildContext context) {

return MaterialApp(

title: '@VB10',

theme: ThemeManager.craeteTheme(AppThemeLight()),

home: SampleView(),

);}}
~~~

Yeees, we can start drawing to the search result screen. Especially don't forget this method and let’s see how to create a theme instance for this project.

```dart
abstract class ThemeManager {

static ThemeData craeteTheme(ITheme theme) => ThemeData(

fontFamily: theme.textTheme.fontFamily,

textTheme: theme.textTheme.data,

cardColor: theme.colors.colorScheme?.onSecondary,

tabBarTheme: TabBarTheme(
```

```
indicator: BoxDecoration(),

labelColor: theme.colors.tabbarSelectedColor,

unselectedLabelColor: theme.colors.tabbarNormalColor,

),

floatingActionButtonTheme: FloatingActionButtonThemeData(

foregroundColor: theme.colors.colors.white,

backgroundColor: theme.colors.colors.green),

appBarTheme: AppBarTheme(backgroundColor:
theme.colors.appBarColor),

scaffoldBackgroundColor: theme.colors.scaffoldBackgroundColor,

colorScheme: theme.colors.colorScheme);

}
```

Now project depends on all theme instances directly as we just change theme value after this project going to a new color scheme additionally project never needs any code on design time. This point means your project design has all the most done