Flutter: .of(context)

What are you afraid .of?

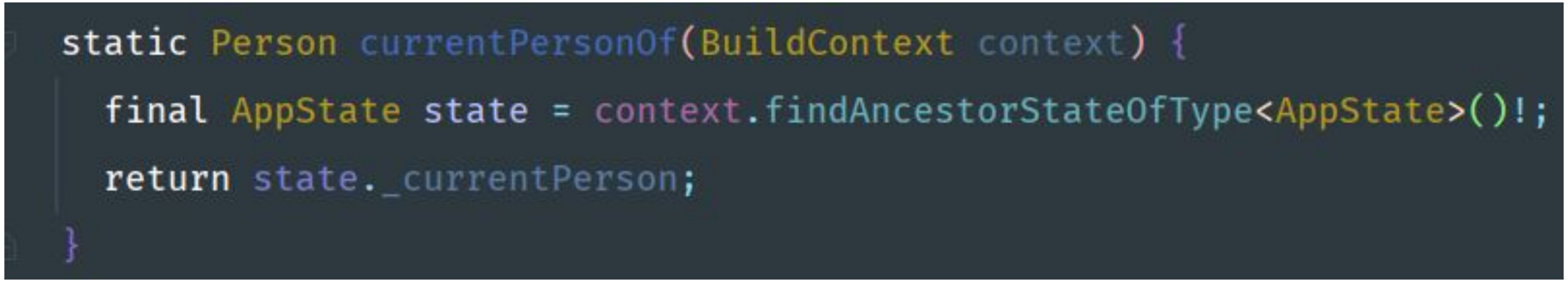
by Scott Stoll

.of(context)

"I see it everywhere,
what is it?"

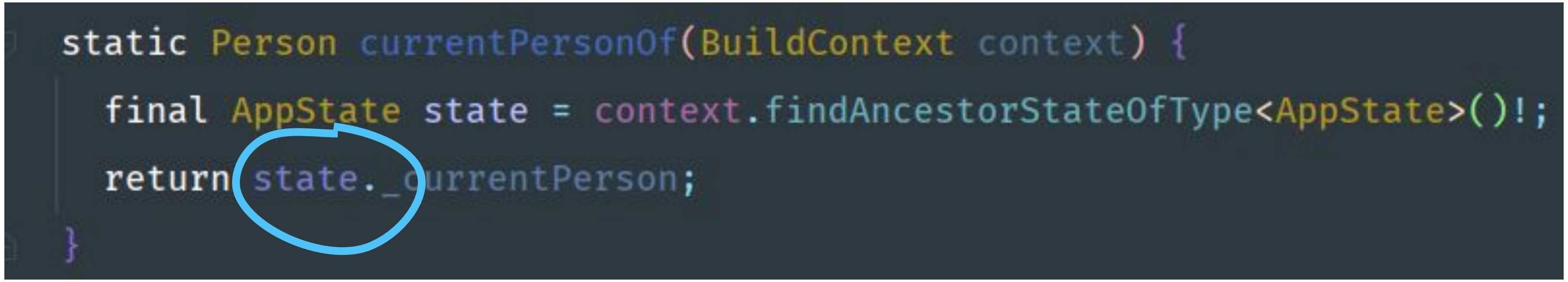
Provider

.of(context)



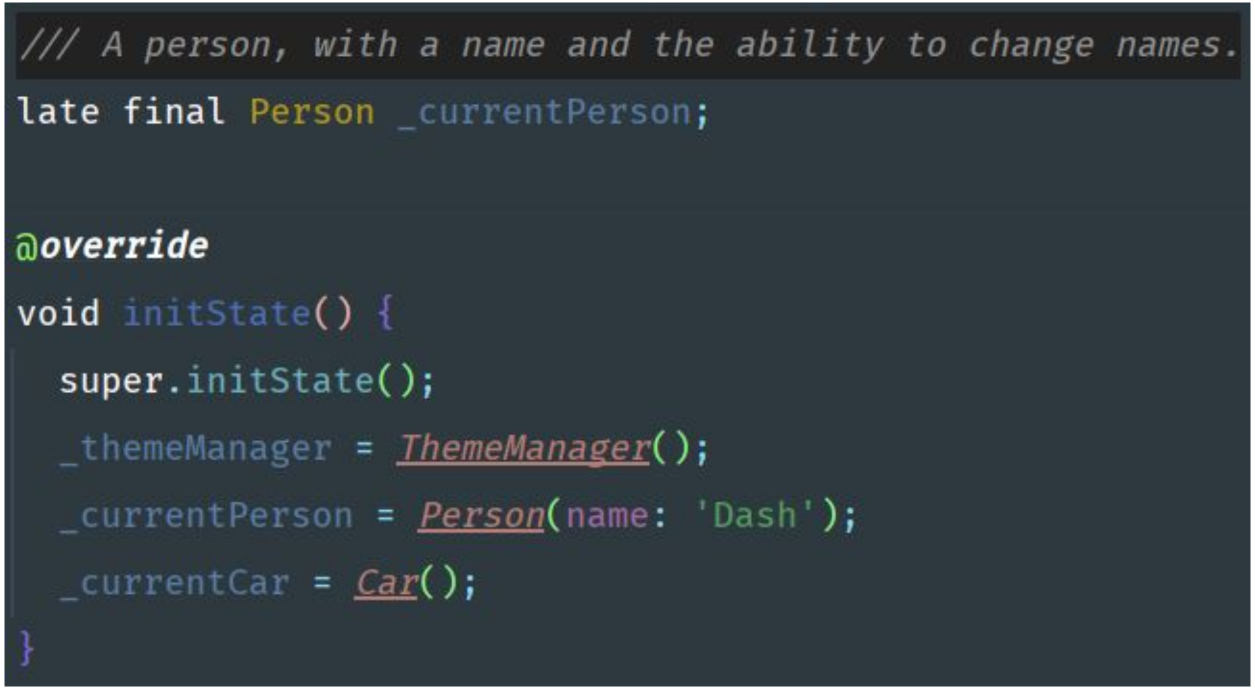
```
static Person currentPersonOf(BuildContext context) {
  final AppState state = context.findAncestorStateOfType<AppState>()!;
  return state._currentPerson;
}
```

.of(context)



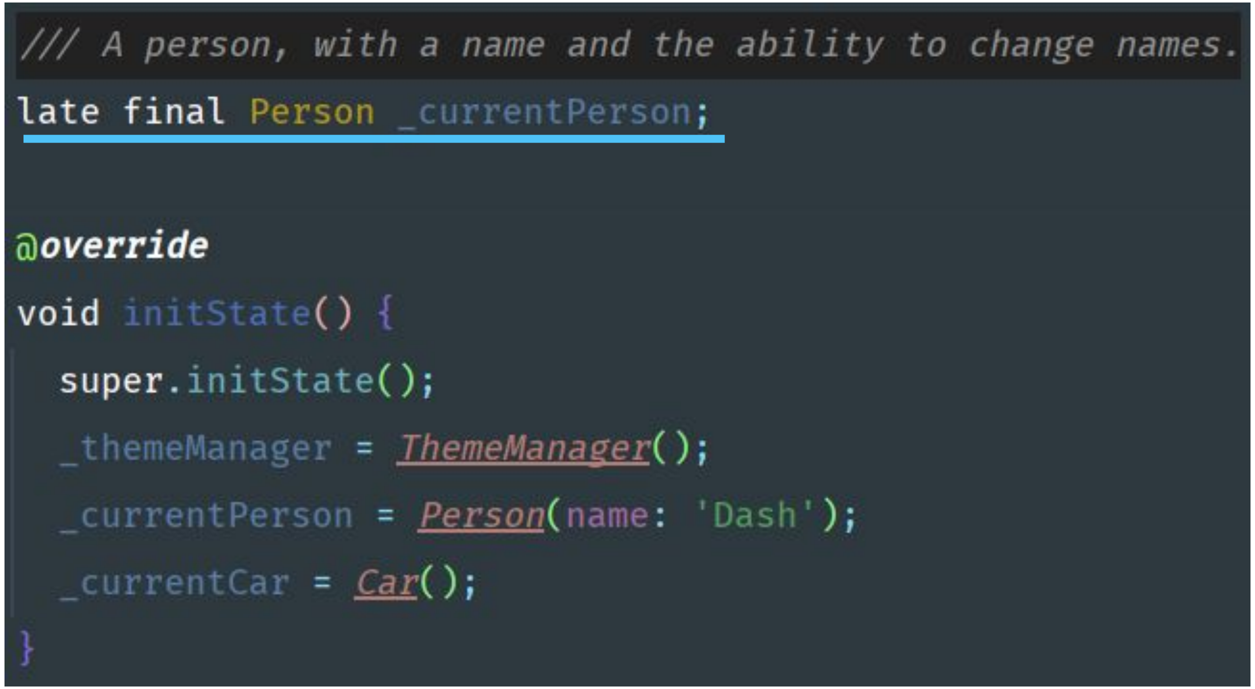
```
static Person currentPersonOf(BuildContext context) {
  final AppState state = context.findAncestorStateOfType<AppState>()!;
  return state._currentPerson;
}
```

.of(context)



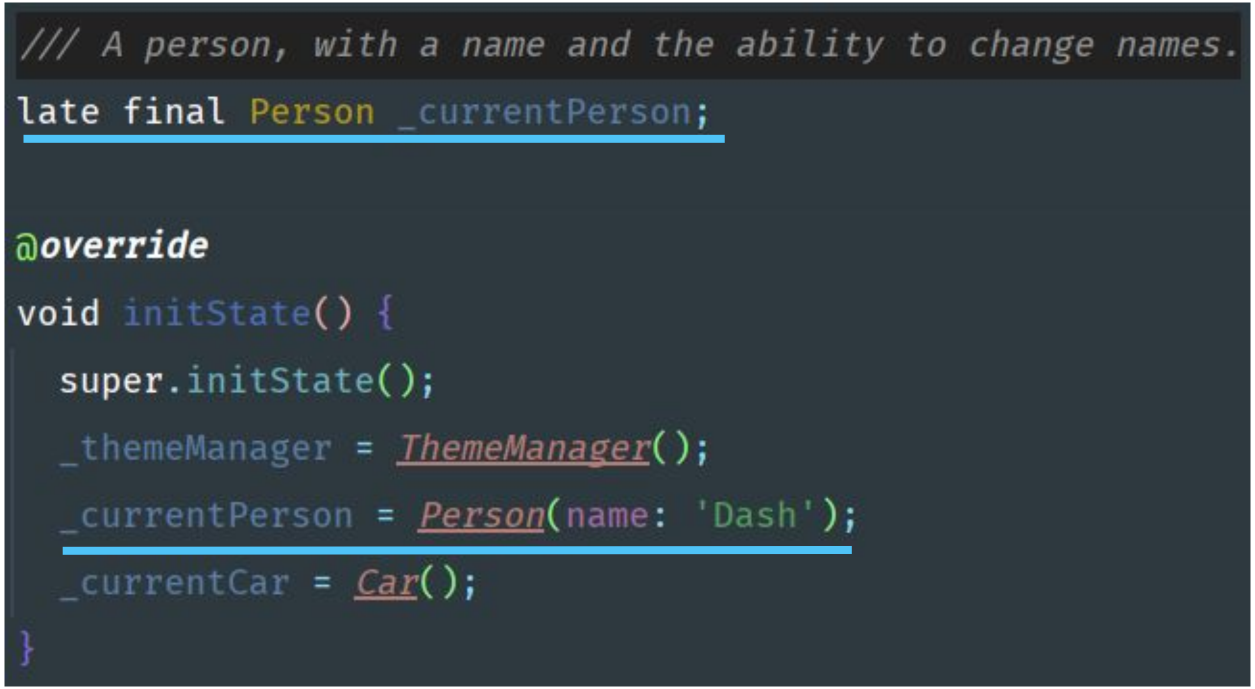
```
/// A person, with a name and the ability to change names.
late final Person _currentPerson;

@override
void initState() {
  super.initState();
  _themeManager = ThemeManager();
  _currentPerson = Person(name: 'Dash');
  _currentCar = Car();
}
```

.of(context)



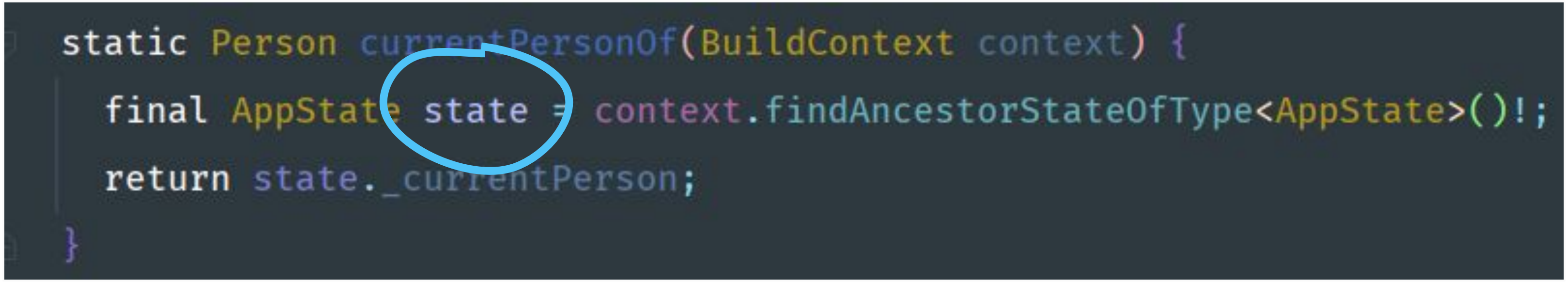
```
/// A person, with a name and the ability to change names.
late final Person _currentPerson;


@override
void initState() {
  super.initState();
  _themeManager = ThemeManager();
  _currentPerson = Person(name: 'Dash');
  _currentCar = Car();
}
```

.of(context)



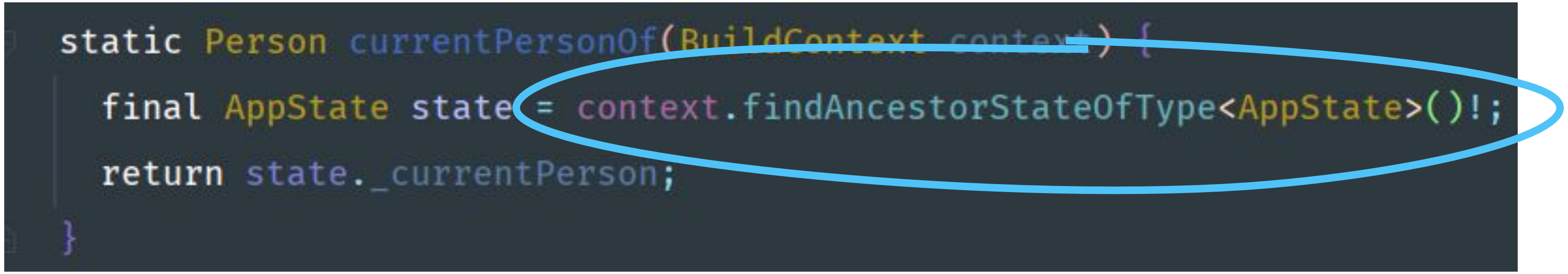
```
/// A person, with a name and the ability to change names.
late final Person _currentPerson;


@override
void initState() {
  super.initState();
  _themeManager = ThemeManager();
  _currentPerson = Person(name: 'Dash');
  _currentCar = Car();
}
```

.of(context)



```
static Person currentPersonOf(BuildContext context) {
  final AppState state = context.findAncestorStateOfType<AppState>()!;
  return state._currentPerson;
}
```

.of(context)



```
static Person currentPersonOf(BuildContext context) {
  final AppState state = context.findAncestorStateOfType<AppState>()!;
  return state._currentPerson;
}
```

.of(context)

```
/// Returns the [State] object of the nearest ancestor [StatefulWidget] widget
/// that is an instance of the given type `T`.
```

.of(context)

```
/// Returns the [State] object of the nearest ancestor [StatefulWidget] widget
/// that is an instance of the given type `T`.
```

.of(context)

```
/// Returns the [State] object of the nearest ancestor [StatefulWidget] widget
/// that is an instance of the given type `T`.
```

```
/// Calling this method is relatively expensive (O(N) in the depth of the
/// tree). Only call this method if the distance from this widget to the
/// desired ancestor is known to be small and bounded.
```

.of(context)

```
/// Returns the [State] object of the nearest ancestor [StatefulWidget] widget
/// that is an instance of the given type `T`.
```

```
/// Calling this method is relatively expensive (O(N) in the depth of the
/// tree). Only call this method if the distance from this widget to the
/// desired ancestor is known to be small and bounded.
```

.of(context)

“Okay,
what do I do with it?”

.of(context)

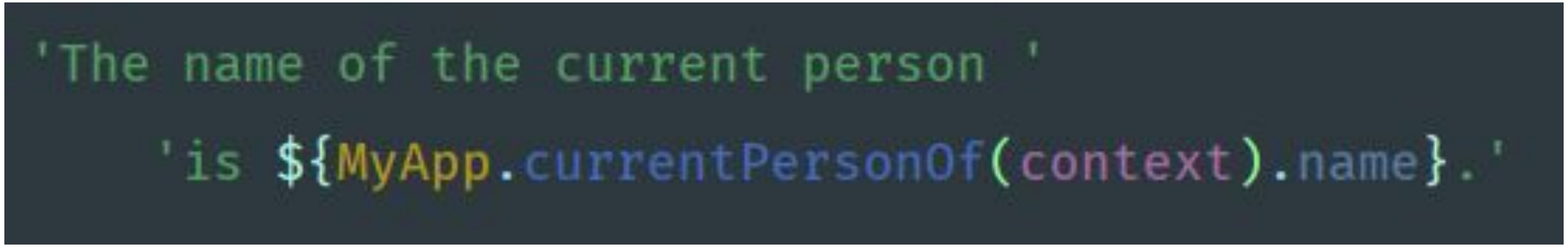
You get stuff!

.of(context)

You get stuff!
Even if it’s not your birthday!

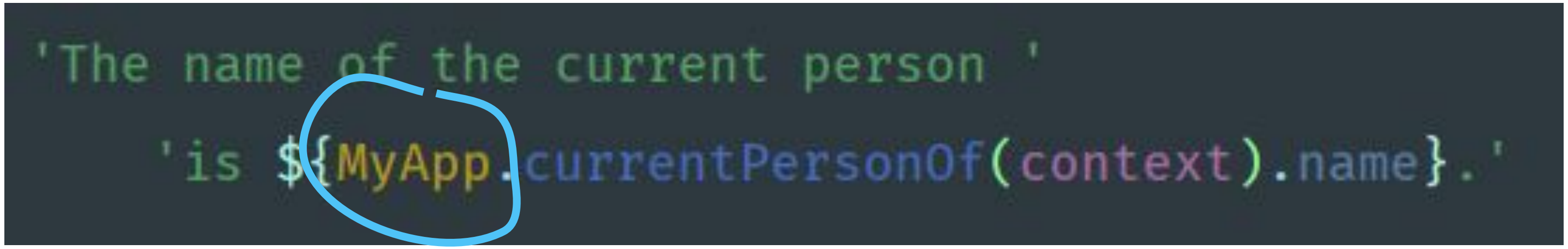
.of(context)

What’s the person’s name?



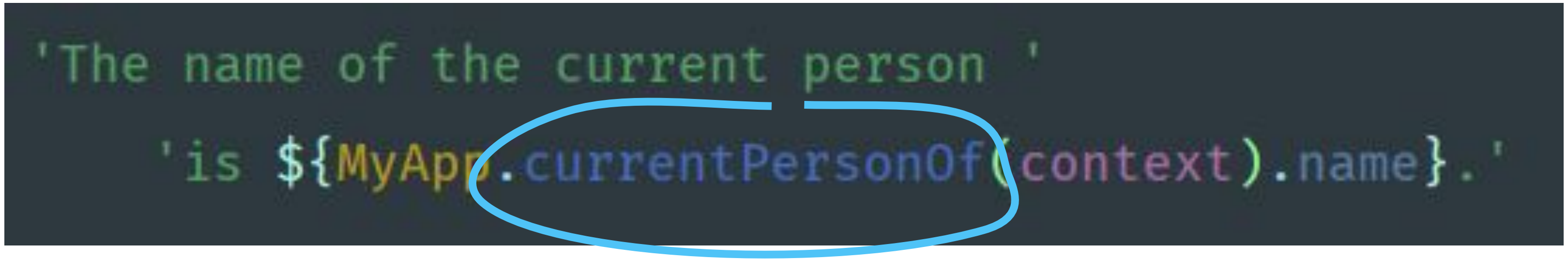
```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

.of(context)



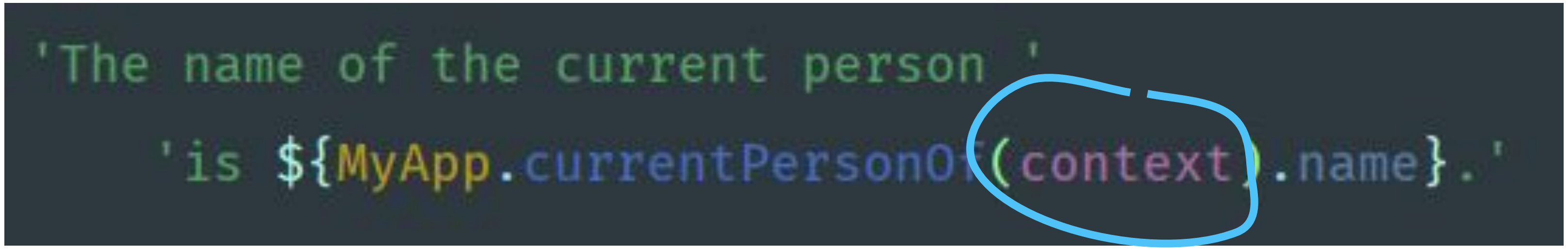
```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

.of(context)



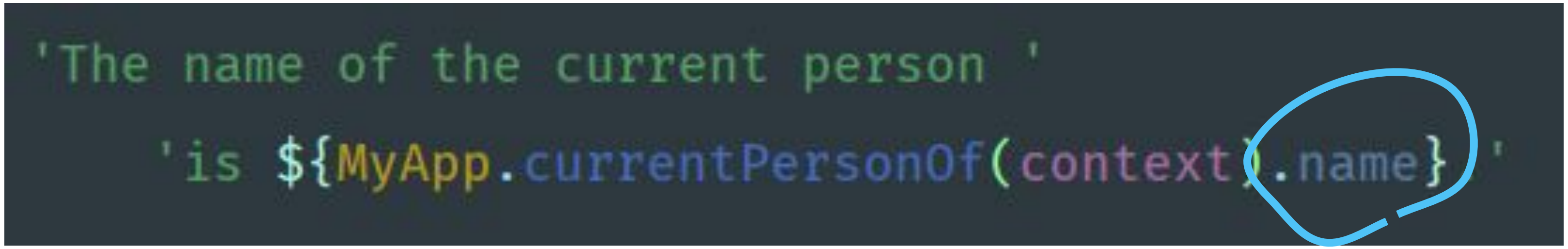
```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

.of(context)



```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

.of(context)

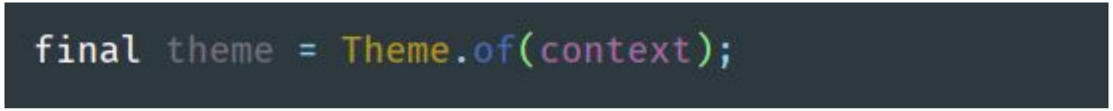


```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

.of(context)

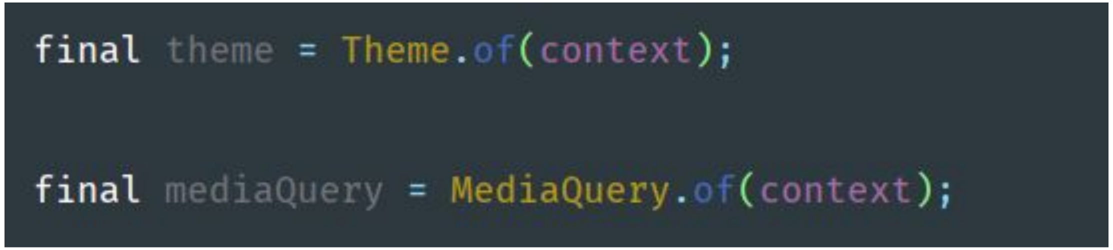
This is used everywhere!

.of(context)



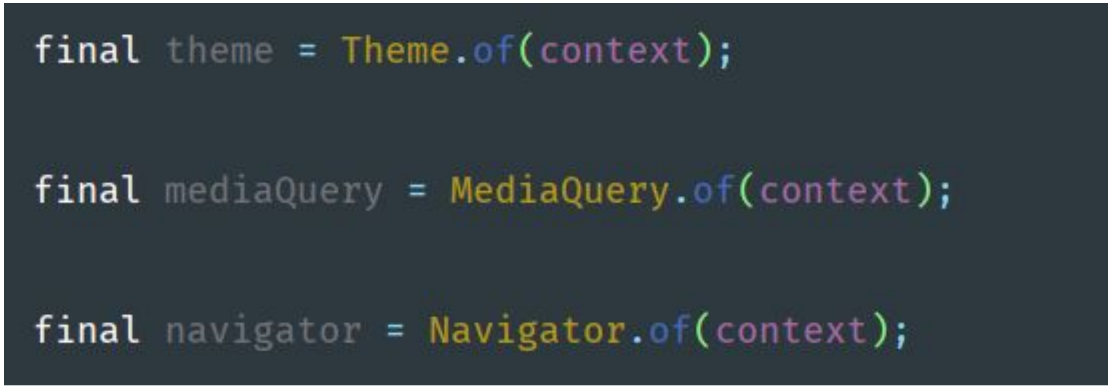
```
final theme = Theme.of(context);
```

.of(context)



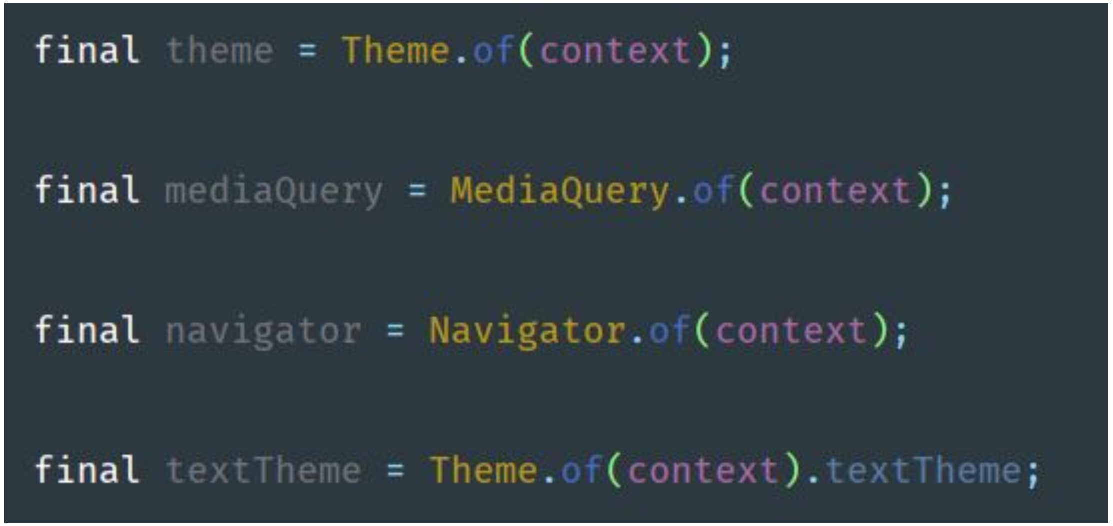
```
final theme = Theme.of(context);


final mediaQuery = MediaQuery.of(context);
```

.of(context)



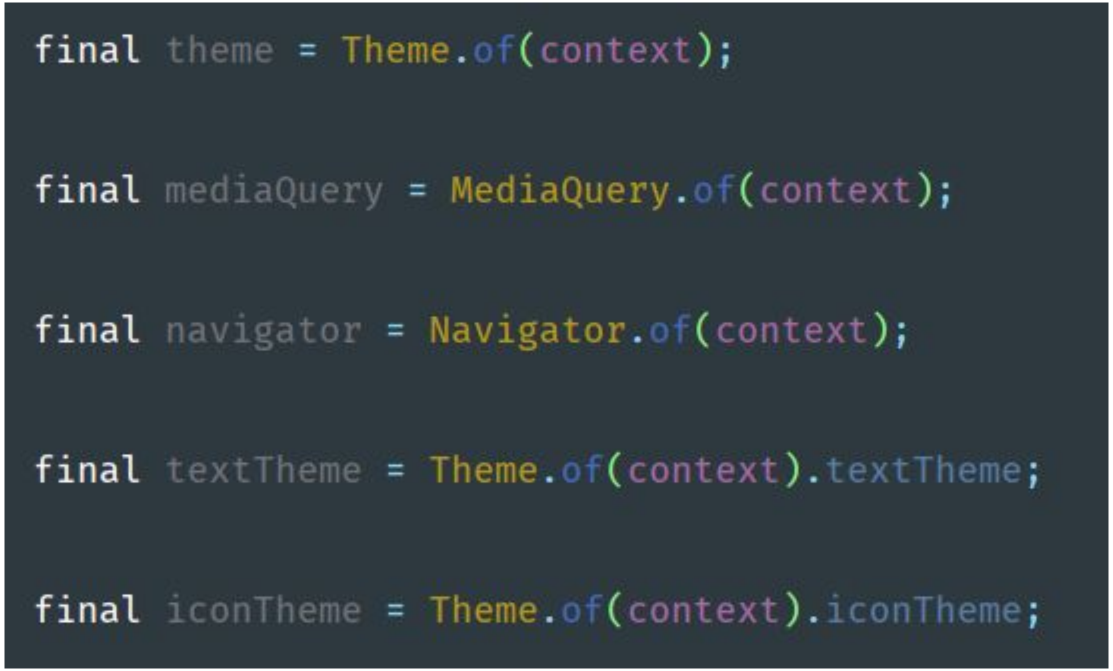
```
final theme = Theme.of(context);

final mediaQuery = MediaQuery.of(context);

final navigator = Navigator.of(context);
```

.of(context)



```
final theme = Theme.of(context);

final mediaQuery = MediaQuery.of(context);

final navigator = Navigator.of(context);

final textTheme = Theme.of(context).textTheme;
```

.of(context)



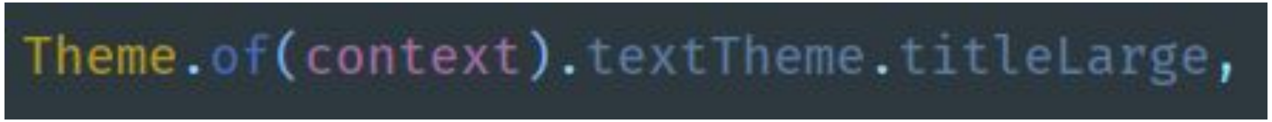
```
final theme = Theme.of(context);

final mediaQuery = MediaQuery.of(context);

final navigator = Navigator.of(context);

final textTheme = Theme.of(context).textTheme;

final iconTheme = Theme.of(context).iconTheme;
```

.of(context)

Material Design 3 Compliant text size?

.of(context)

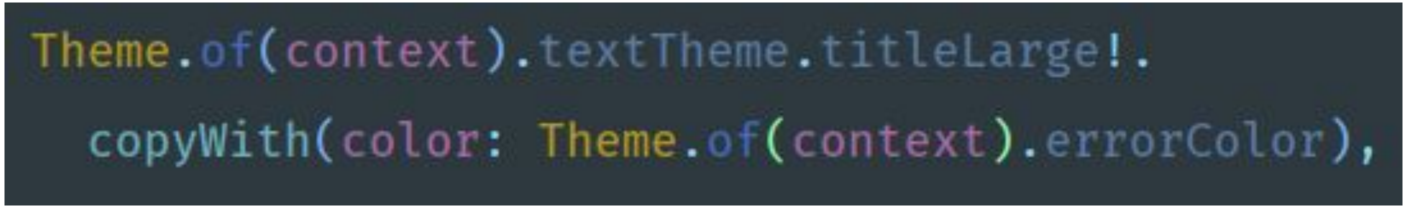
Material Design 3 Compliant text size?



```
Theme.of(context).textTheme.titleLarge,
```

.of(context)

Material Design 3 Compliant text size?
To show an error?



```
Theme.of(context).textTheme.titleLarge!.
  copyWith(color: Theme.of(context).errorColor),
```

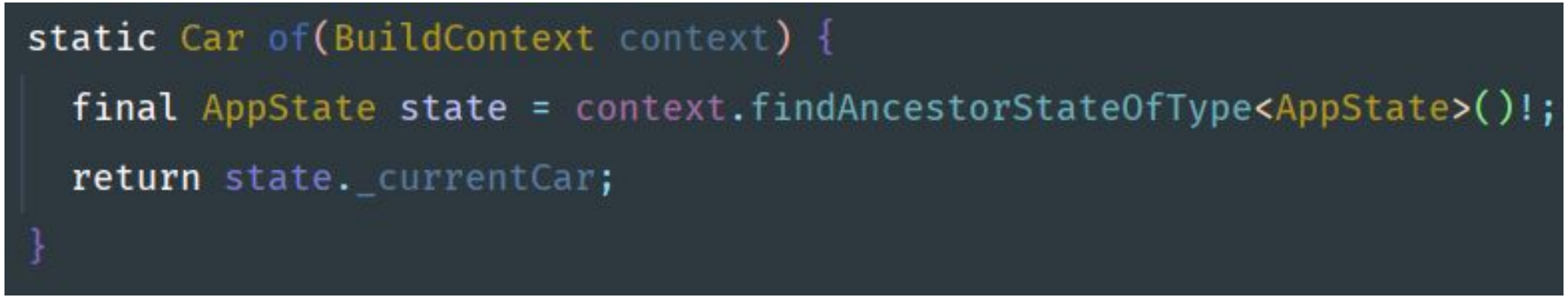
.of(context)

Wanna see something cool?

.of(context)

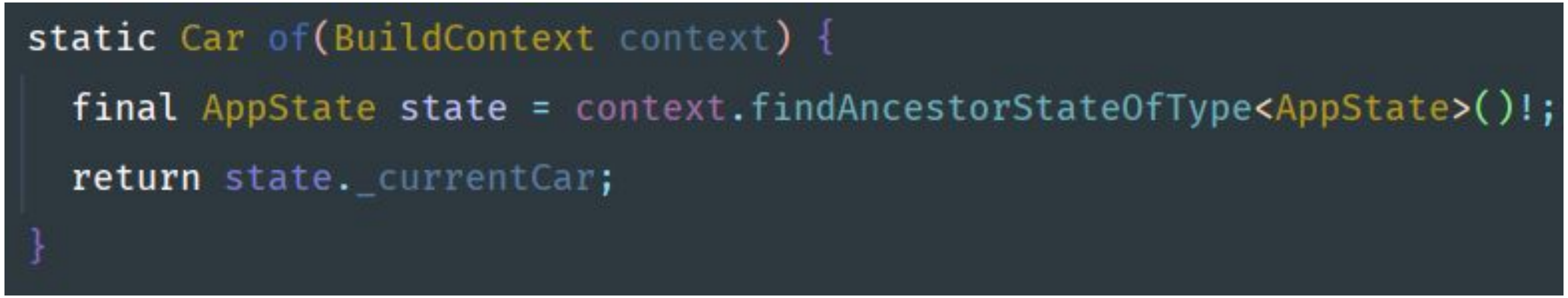
Wanna see something cool?
Do you like “clean code” naming?

.of(context)

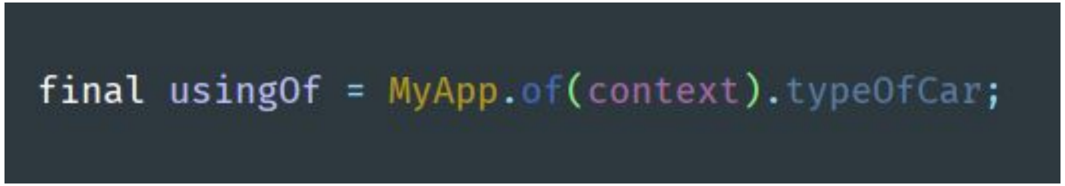


```
static Car of(BuildContext context) {
  final AppState state = context.findAncestorStateOfType<AppState>()!;
  return state._currentCar;
}
```

.of(context)

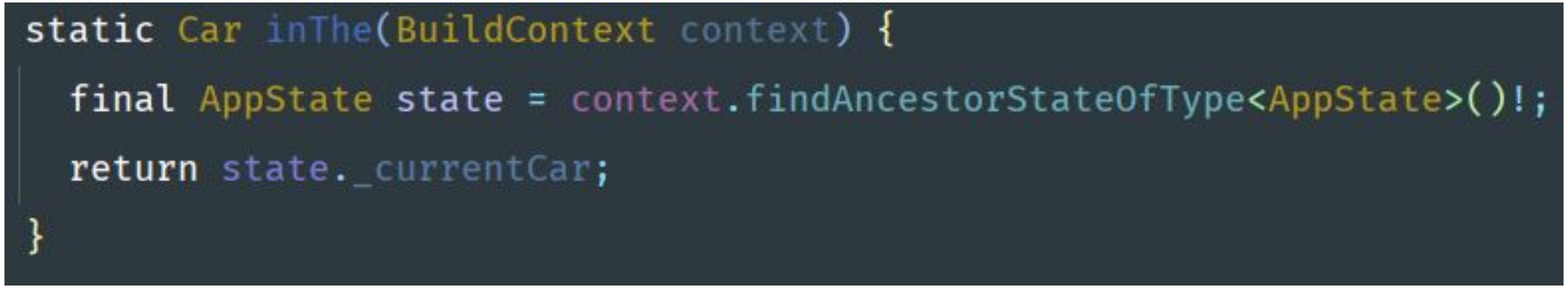


```dart
static Car of(BuildContext context) {
  final AppState state = context.findAncestorStateOfType<AppState>()!;
  return state._currentCar;
}
```



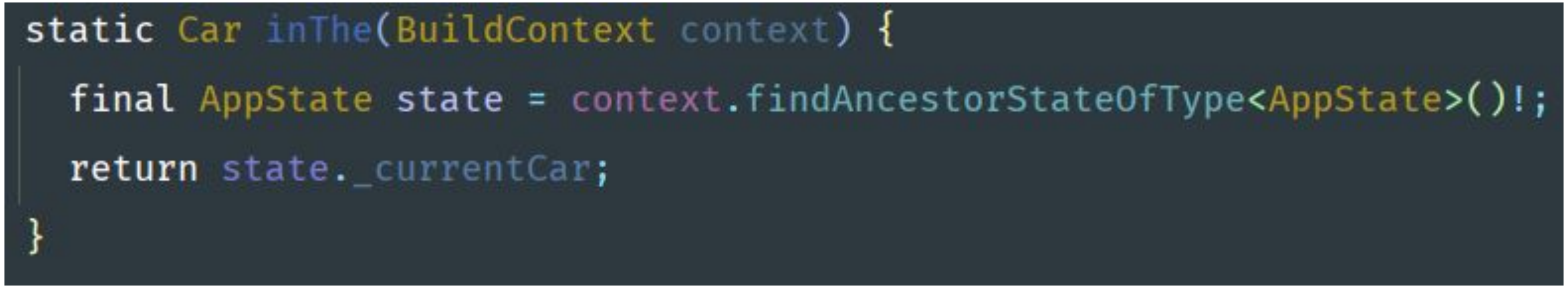
```dart
final usingOf = MyApp.of(context).typeOfCar;
```

.of(context)

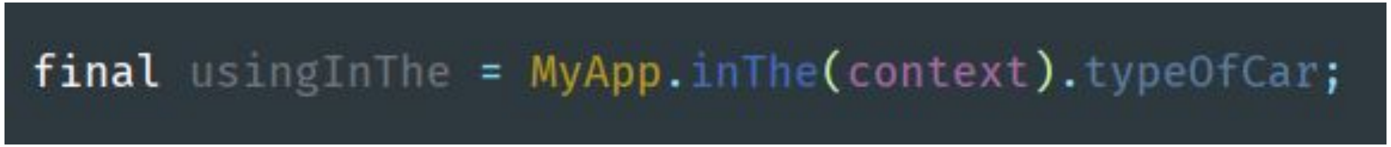


```
static Car inThe(BuildContext context) {
  final AppState state = context.findAncestorStateOfType<AppState>()!;
  return state._currentCar;
}
```

.of(context)

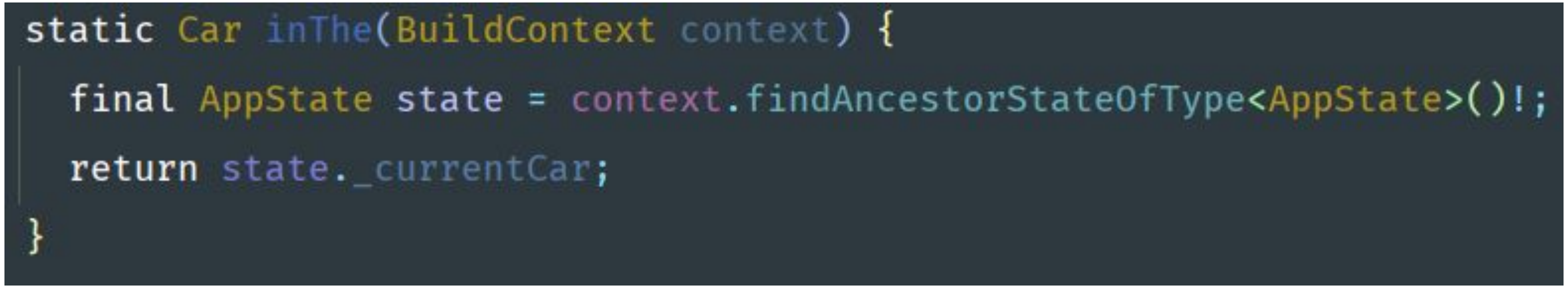


```dart
static Car inThe(BuildContext context) {
  final AppState state = context.findAncestorStateOfType<AppState>()!;
  return state._currentCar;
}
```

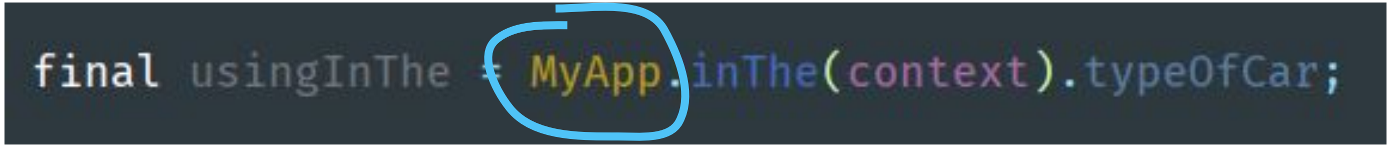


```dart
final usingInThe = MyApp.inThe(context).typeOfCar;
```

.of(context)



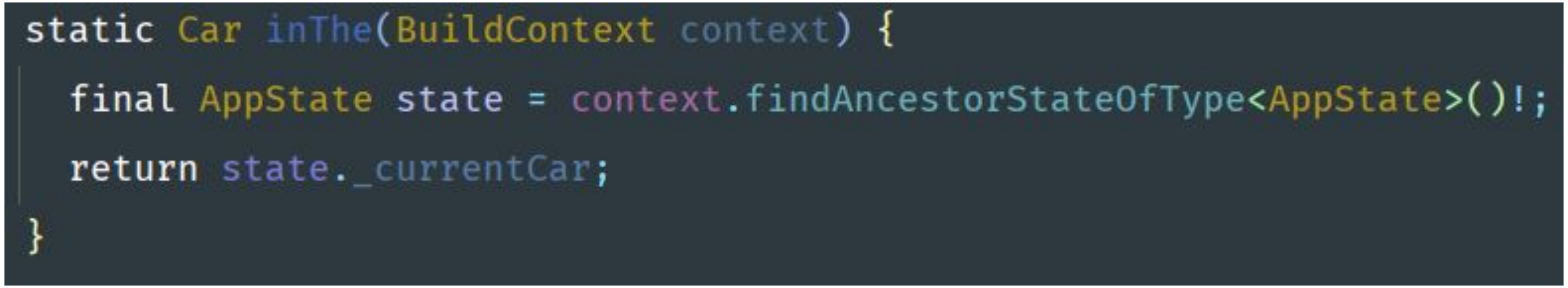
```
static Car inThe(BuildContext context) {
  final AppState state = context.findAncestorStateOfType<AppState>()!;
  return state._currentCar;
}
```



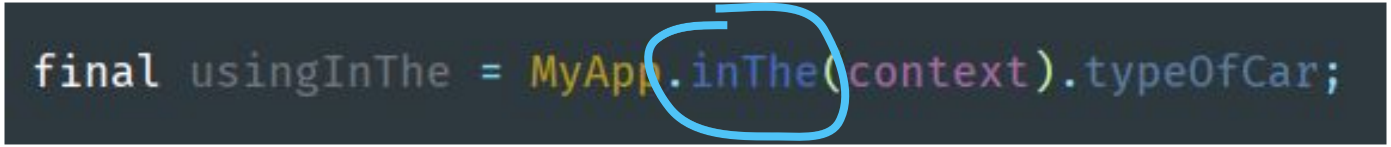
```
final usingInThe = MyApp.inThe(context).typeOfCar;
```

“Find the MyApp instance in the context and return its typeOfCar”

.of(context)



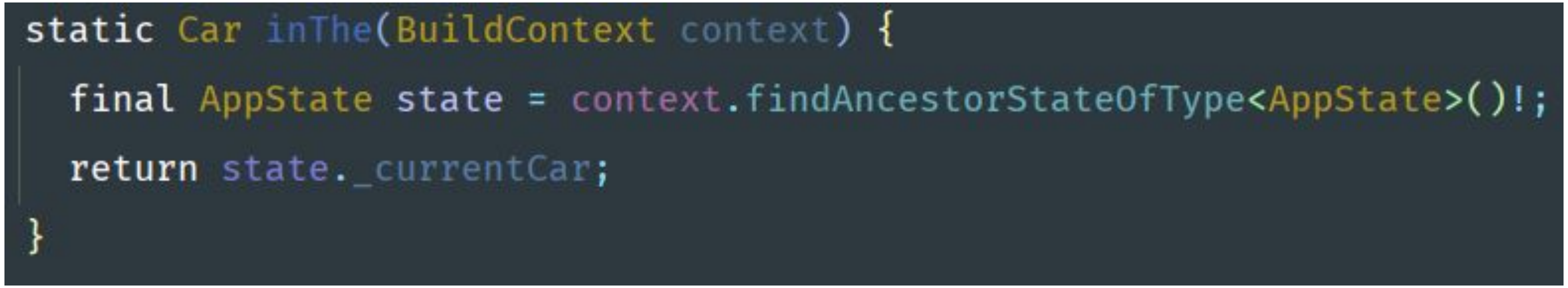
```
static Car inThe(BuildContext context) {
  final AppState state = context.findAncestorStateOfType<AppState>()!;
  return state._currentCar;
}
```



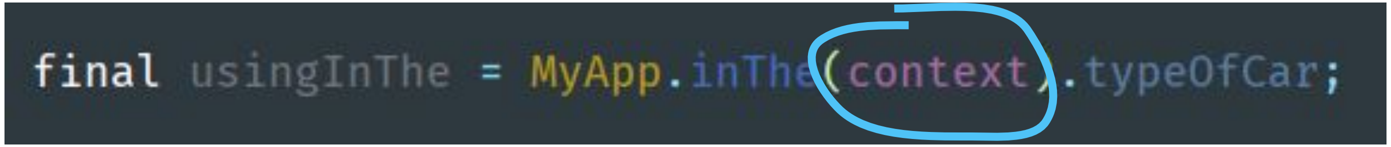
```
final usingInThe = MyApp.inThe(context).typeOfCar;
```

“Find the MyApp instance in the context and return its typeOfCar”

.of(context)



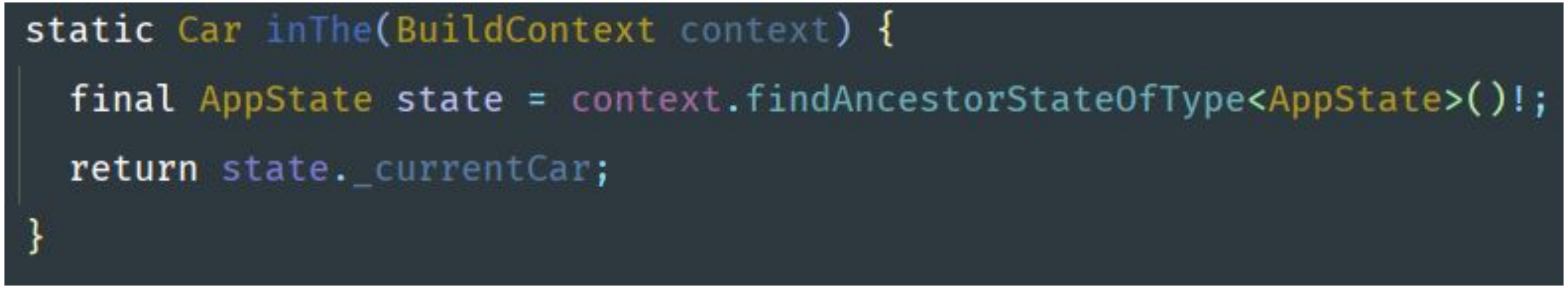
```
static Car inThe(BuildContext context) {
  final AppState state = context.findAncestorStateOfType<AppState>()!;
  return state._currentCar;
}
```



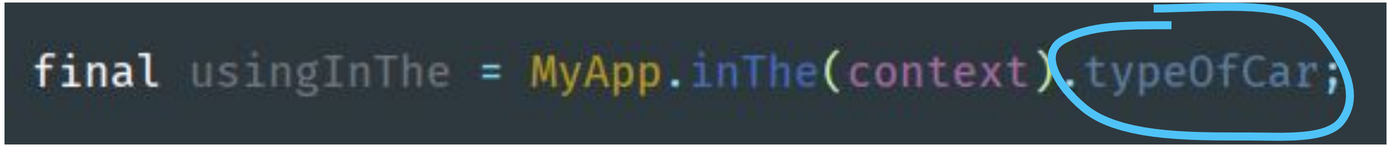
```
final usingInThe = MyApp.inThe(context).typeOfCar;
```

“Find the MyApp instance in the context and return its typeOfCar”

.of(context)



```
static Car inThe(BuildContext context) {
  final AppState state = context.findAncestorStateOfType<AppState>()!;
  return state._currentCar;
}
```



```
final usingInThe = MyApp.inThe(context).typeOfCar;
```

“Find the MyApp instance in the context and return its typeOfCar”

.of(context)

Important!

.of(context)

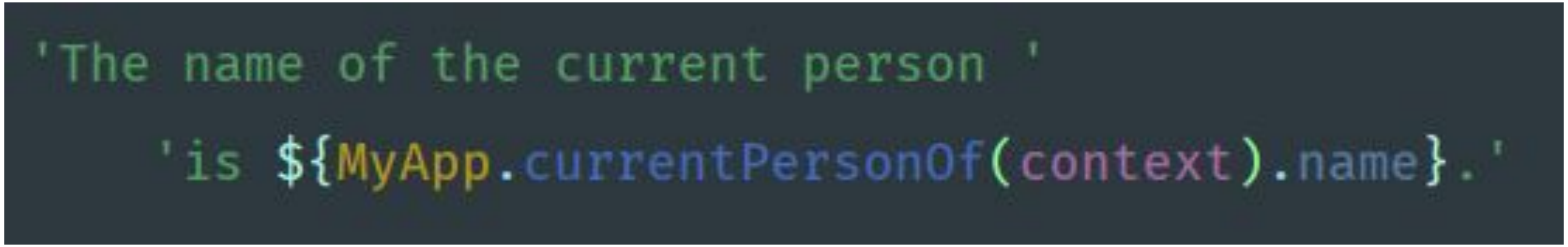
Important!

- The class before the .of (or .whatever) is where you find the method, NOT the class that owns the method!

.of(context)

Important!

- The class before the .of (or .whatever) is where you find the method, NOT the class that owns the method!



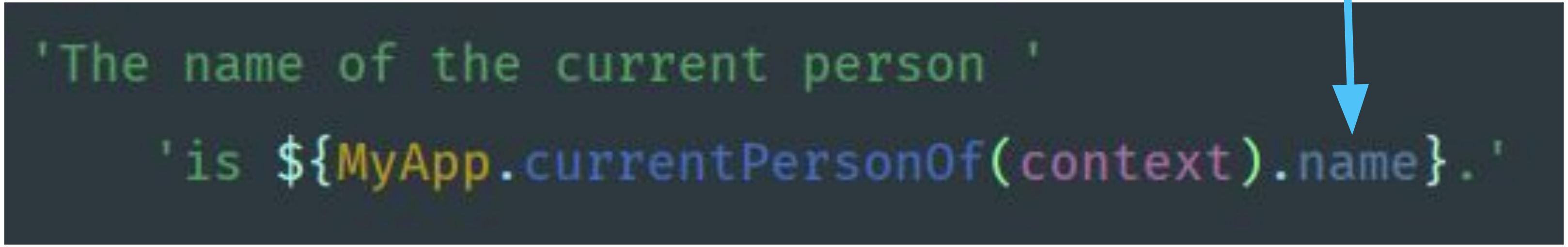
```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

.of(context)

Important!

- The class before the .of (or .whatever) is where you find the method, NOT the class that owns the method!

We get a name from a Person

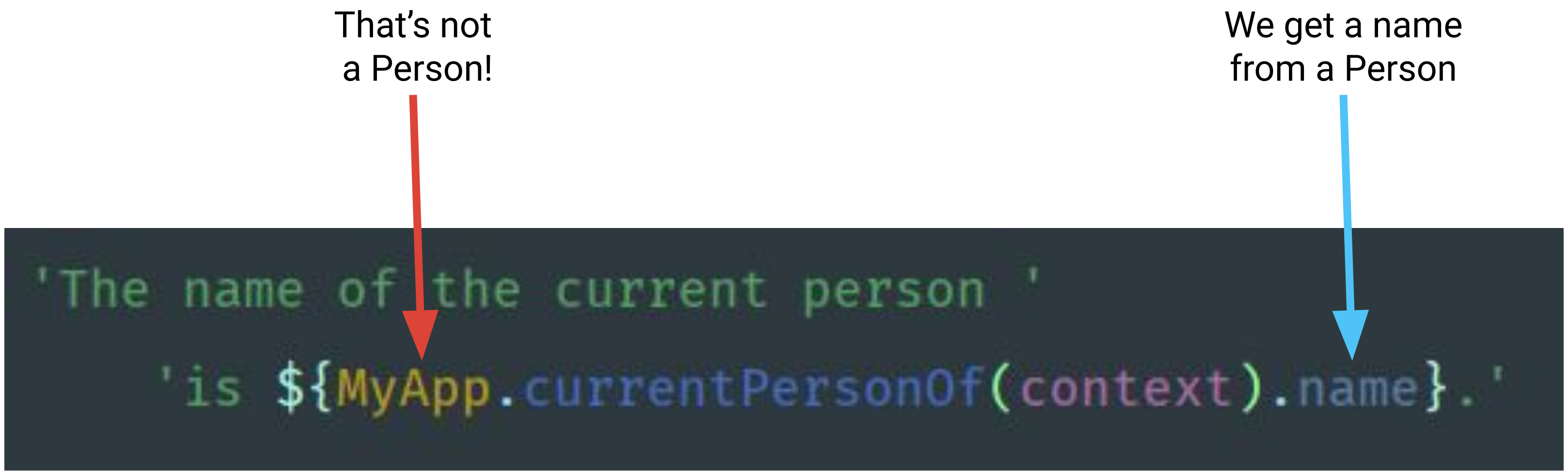


```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

.of(context)

Important!

- The class before the .of (or .whatever) is where you find the method, NOT the class that owns the method!

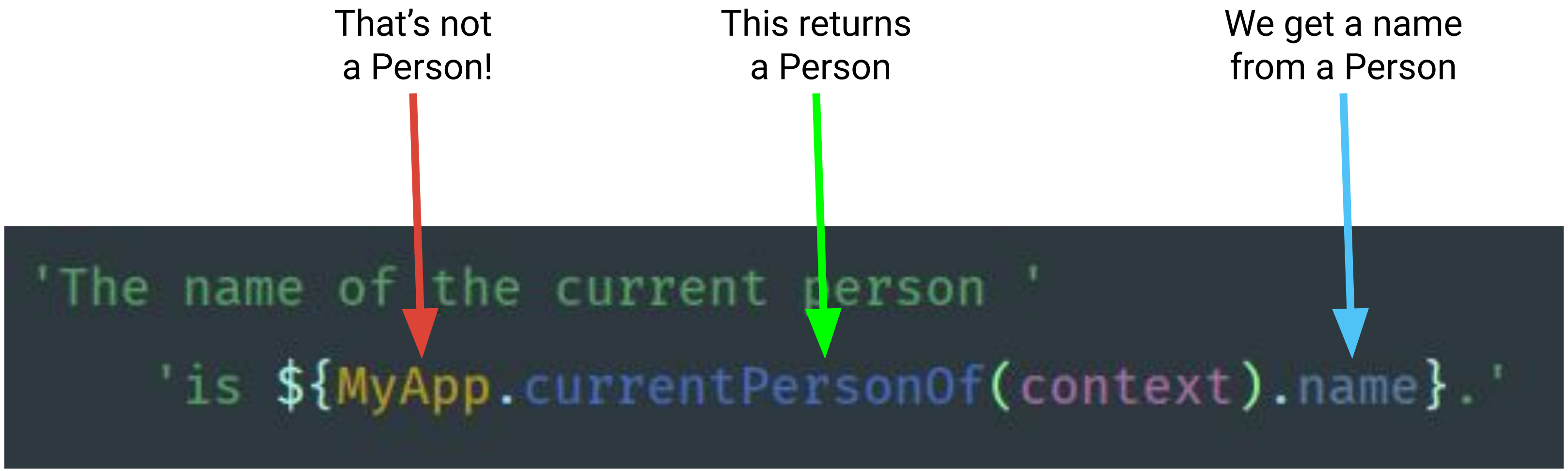


```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

.of(context)

Important!

- The class before the .of (or .whatever) is where you find the method, NOT the class that owns the method!



```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

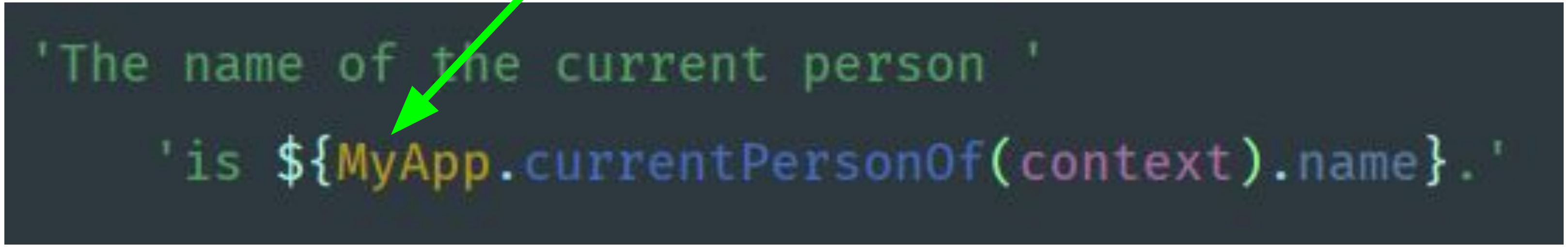
.of(context)

And where is that method?

.of(context)

And where is that method?

It's in here.

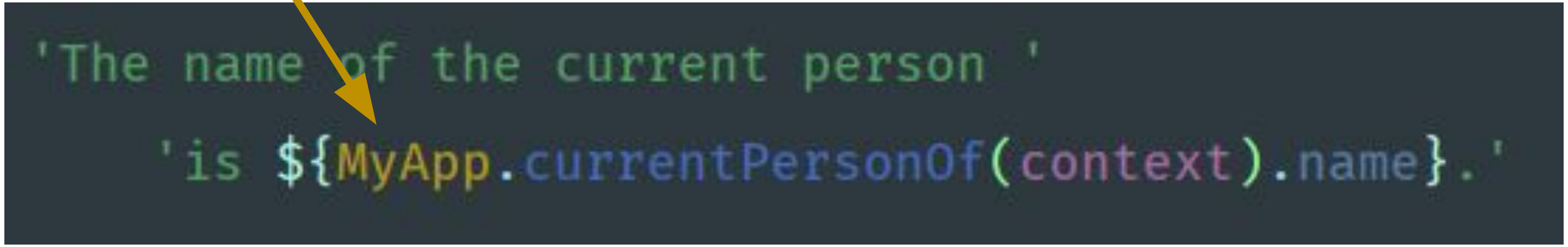


```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

.of(context)

And where is that method?

Here is where you find this method, which takes a BuildContext, and returns this.

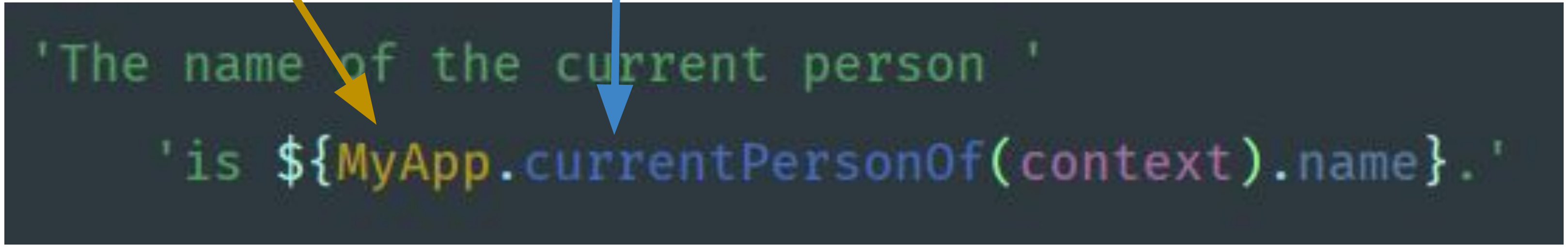


```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

.of(context)

And where is that method?

Here is where you find this method, which takes a BuildContext, and returns this.

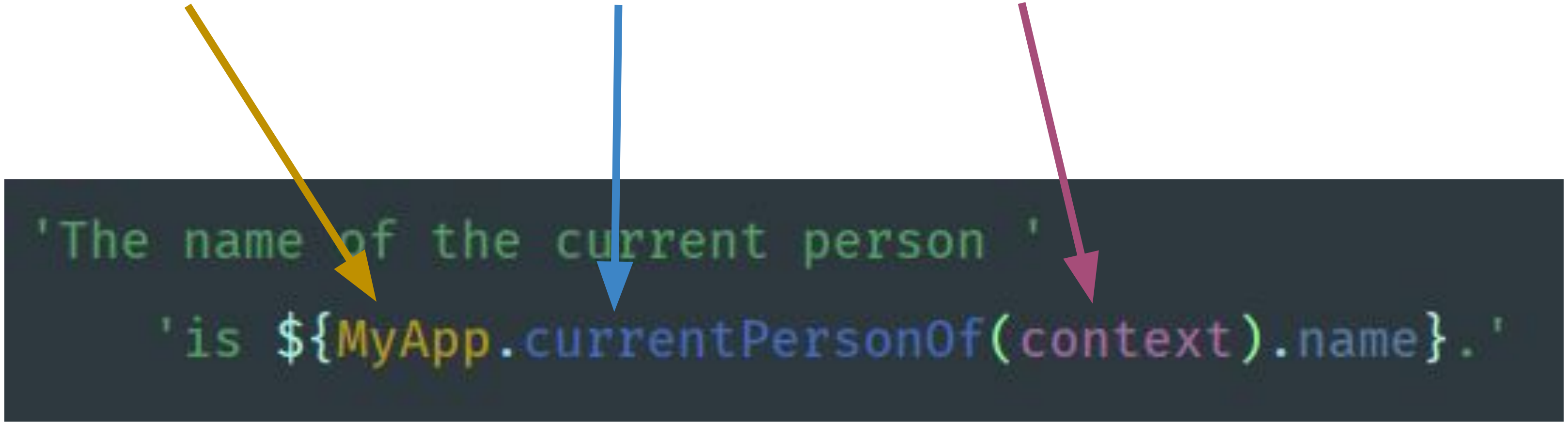


```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

.of(context)

And where is that method?

Here is where you find this method, which takes a BuildContext, and returns this.

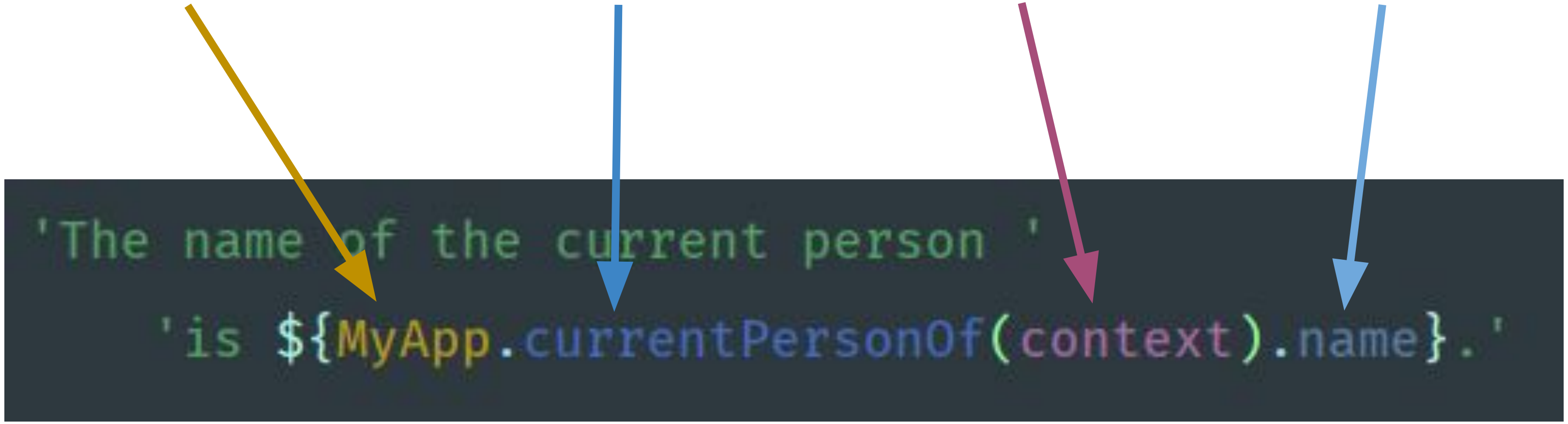


```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

.of(context)

And where is that method?

Here is where you find this method, which takes a BuildContext, and returns this.

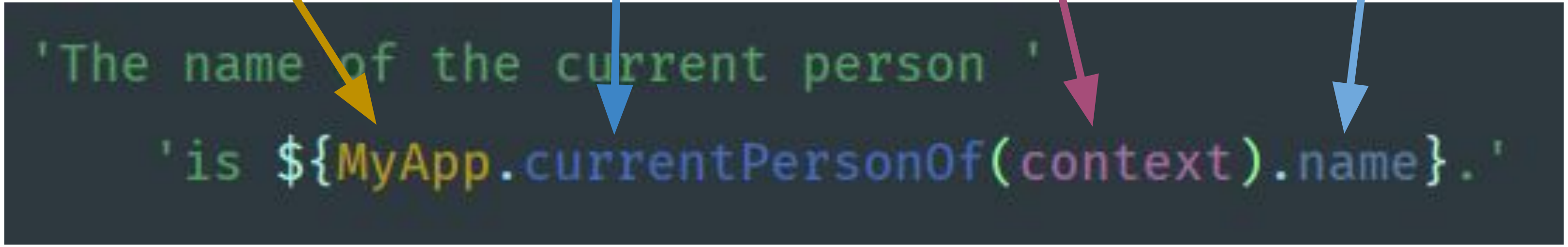


```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

.of(context)

Memorize This

Here is where you find this method, which takes a BuildContext, and returns this.



```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

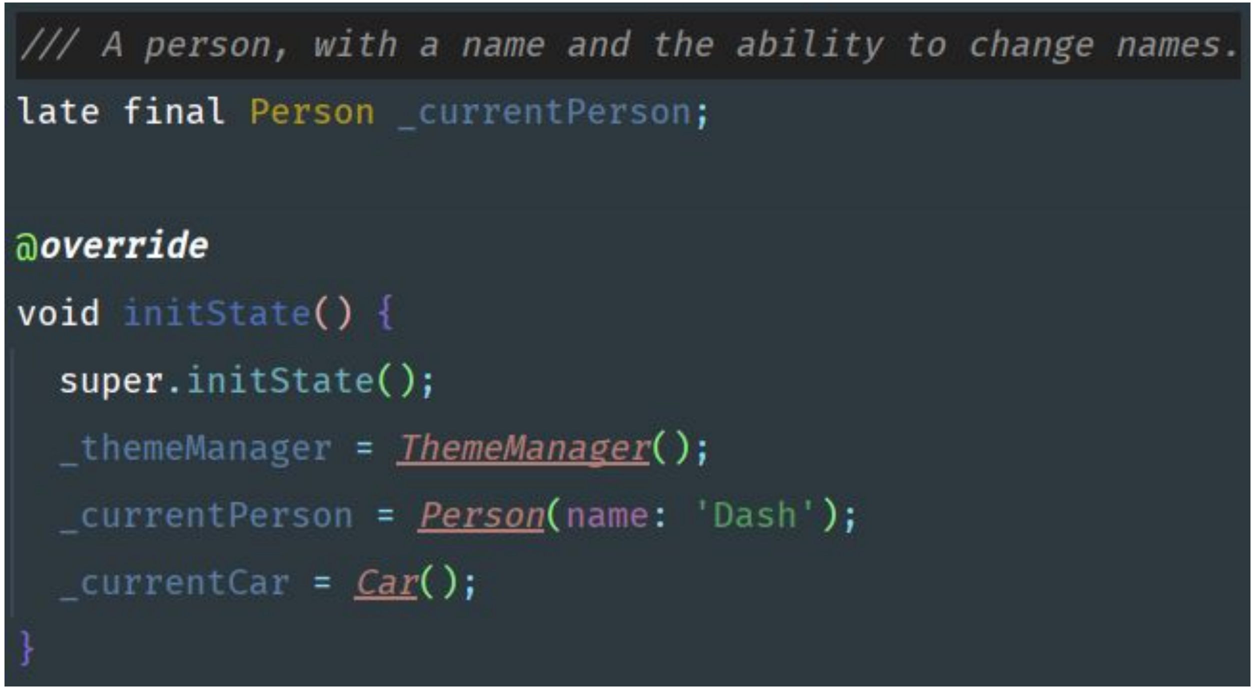
.of(context)

Summary

.of(context)

Summary

- Declare your instance in AppState and initialize in initState

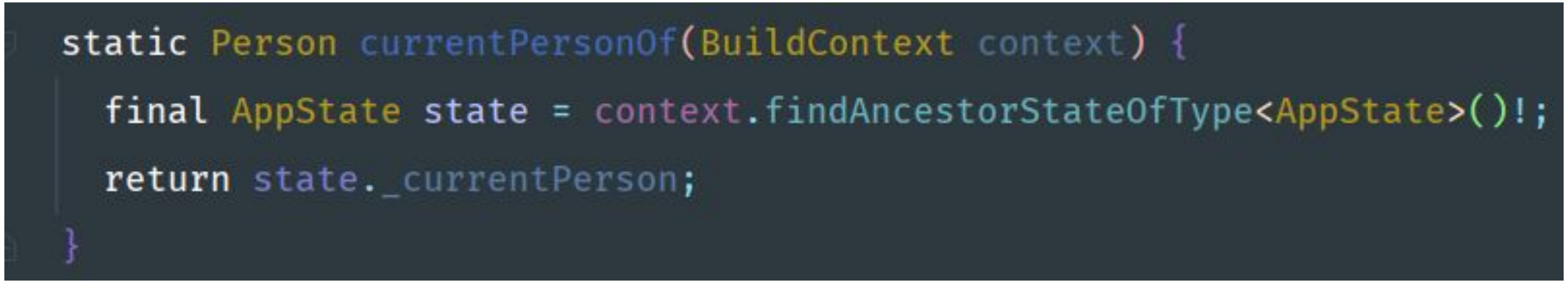


```
/// A person, with a name and the ability to change names.
late final Person _currentPerson;

@override
void initState() {
  super.initState();
  _themeManager = ThemeManager();
  _currentPerson = Person(name: 'Dash');
  _currentCar = Car();
}
```

.of(context)

Summary

- Create a static method to access it, often called of( )
  - It needs to take a BuildContext



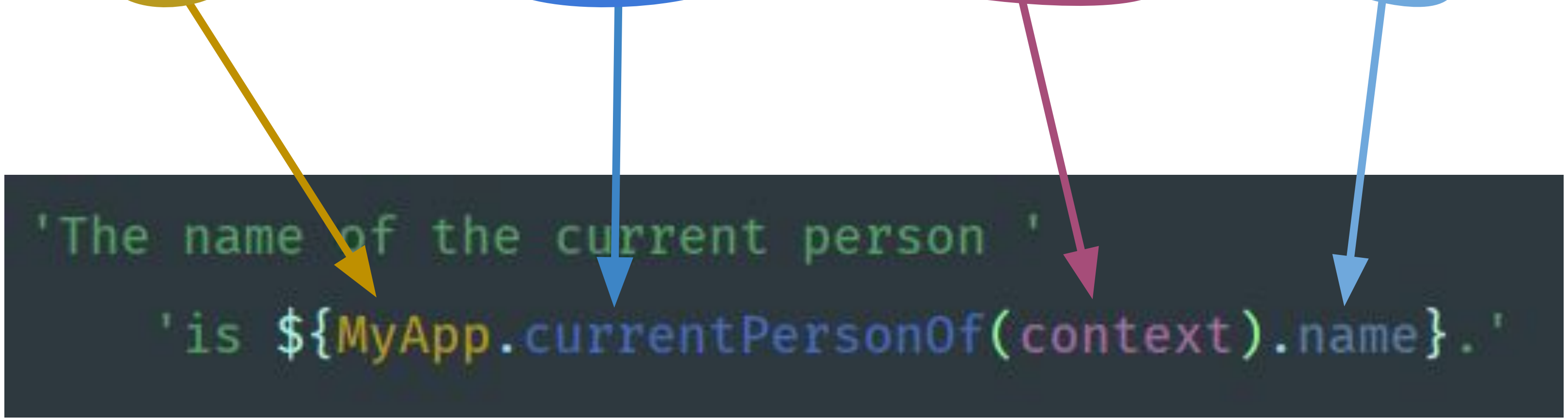
```
static Person currentPersonOf(BuildContext context) {
  final AppState state = context.findAncestorStateOfType<AppState>()!;
  return state._currentPerson;
}
```

.of(context)

Summary

- When you need to use it, this is how you access it.

Here is where you find this method, which takes a BuildContext, and returns this.



```
'The name of the current person '
    'is ${MyApp.currentPersonOf(context).name}.'
```

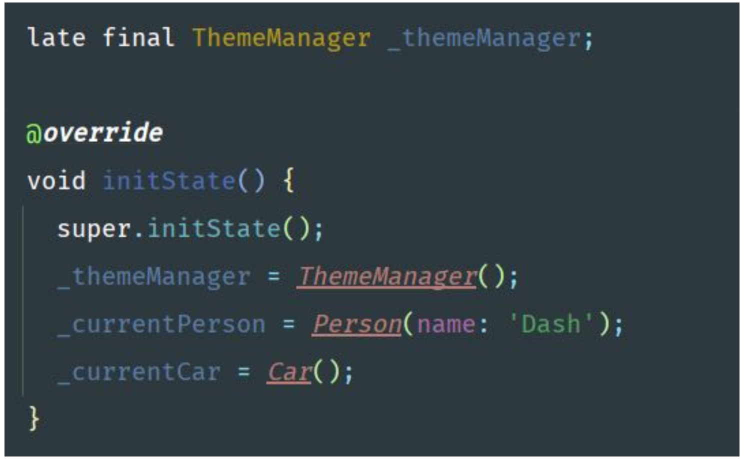
.of(context)

Recommended State Management

.of(context)

Recommended State Management

- Have your class extend ChangeNotifier and wrap your affected Widgets in an AnimatedBuilder. Your instance is the Listenable. (E.G.: _themeManger is the listenable)

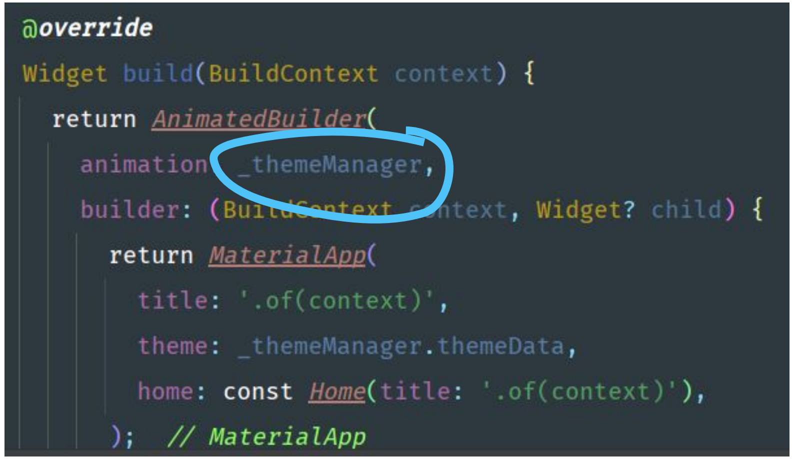


```
late final ThemeManager _themeManager;

@override
void initState() {
  super.initState();
  _themeManager = ThemeManager();
  _currentPerson = Person(name: 'Dash');
  _currentCar = Car();
}
```

.of(context)

Recommended State Management

- Have your class extend ChangeNotifier and wrap your affected Widgets in an AnimatedBuilder. Your instance is the Listenable. (E.G.: _themeManger is the listenable)



```
@override
Widget build(BuildContext context) {
  return AnimatedBuilder(
    animation: _themeManager,
    builder: (BuildContext context, Widget? child) {
      return MaterialApp(
        title: '.of(context)',
        theme: _themeManager.themeData,
        home: const Home(title: '.of(context)'),
      ); // MaterialApp
```

.of(context)

Recommended State Management

Use provider / riverpod / flutter_riverpod

OR

Use bloc / flutter_bloc

.of(context)

Recommended State Management

AVOID BIG COMPLICATED PACKAGES

THAT TRY TO DO EVERYTHING AND BE

EVERYTHING TO EVERYBODY!!!

- Gitter: The official chat channel of Flutter. Accessible via flutter.dev
- The Flutter Dev Google Group: https://groups.google.com/forum/#!forum/flutter-dev
- Reddit:
  https://www.reddit.com/r/FlutterDev/
- Flutter Community on Medium:
  https://medium.com/flutter-community
- Flutter Community Page on Facebook
  https://www.facebook.com/FlutterCommunity/
- Twitter: #Flutter
  Official: @flutterdev
  Follow who I follow: @scottstoll2017

Live open Q&A every Wednesday
on Flutter Community YouTube!