Dart UP 2020

Mangirdas Kazlauskas

Devbridge

Design Patterns toolbox:
(not so) obvious patterns for Flutter

@mkobuolys
mkobuolys.medium.com
mangirdas.kazlauskas

Agenda

- About me
- OOP Design Patterns 101
- Example App
- Abstract Factory
- Composite
- Command
- Memento

Dart UP 2020

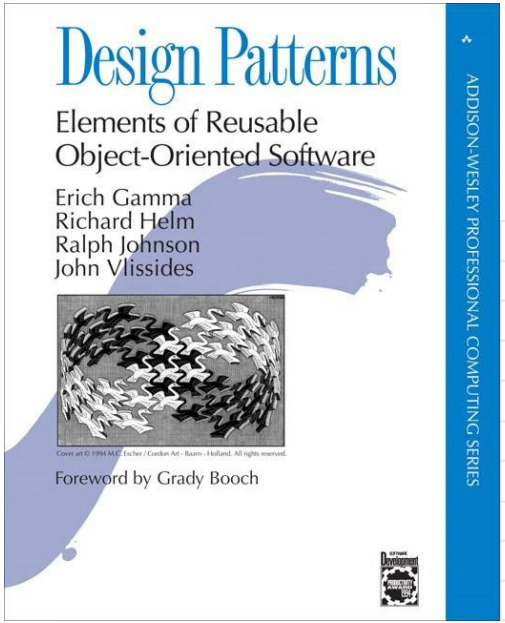
About me

- Software Engineer from Lithuania
- No prior mobile app development experience before Flutter
- Using Flutter since v0.10.2

* Finished the Snake game on Nokia 3310, twice!

OOP Design Patterns 101

- Solves common code design problems
- (Only) provides a general idea, structure/blueprint on how to deal with a particular problem
- Speeds up the development process
- Improves code flexibility and reusability



Example App: DartUPify

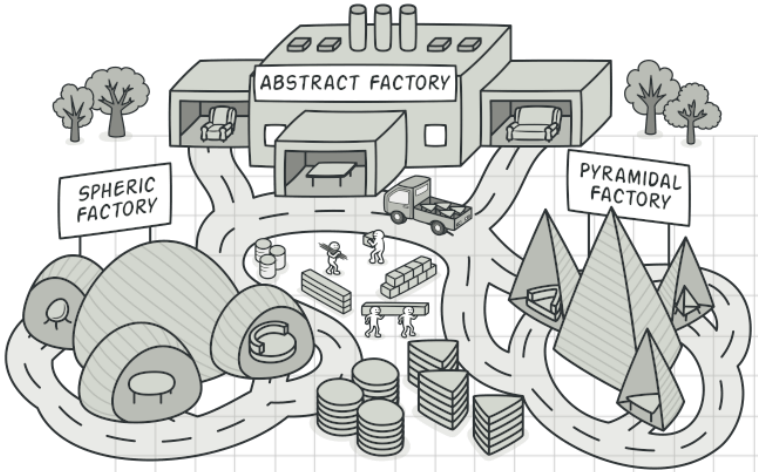
- Music playlist management app
- State management - flutter_bloc
- Dependency Injection – provider

https://github.com/MangirdasKazlauskas/dartupify-dartup2020

Dart UP 2020

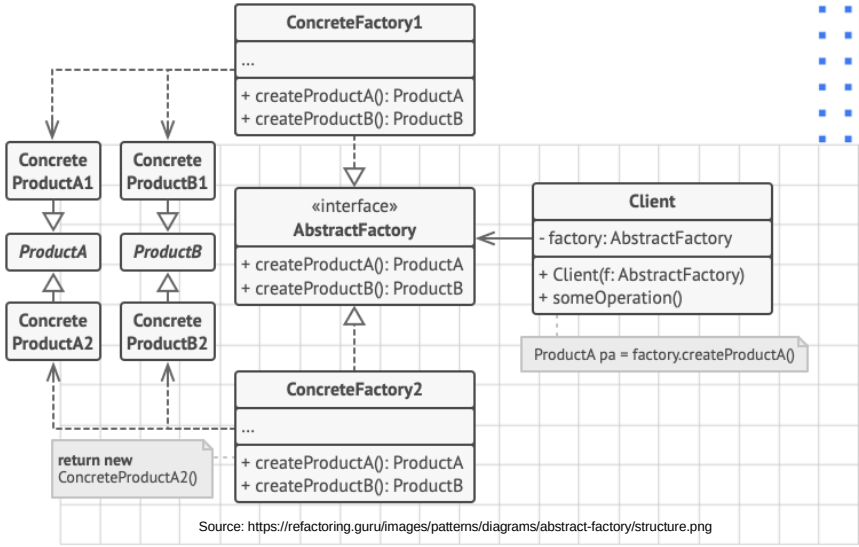
Abstract Factory

“Provide an interface for creating families of related or dependent objects without specifying their concrete classes.” - GoF



Abstract Factory

- Creational design pattern
- *Abstract Factory* - declares an interface of operations that create abstract *Product* objects
- *Concrete Factory* – implements the operations to create *Concrete Product* objects
- *Product* – interface for a type of *Product* object
- *Concrete Product* – defines a product object to be created by corresponding *Concrete Factory*



Source: https://refactoring.guru/images/patterns/diagrams/abstract-factory/structure.png

Abstract Factory: factories

```dart
abstract class IPlatformWidgetsFactory {
  Widget createAppBar(String title, [bool showSettingsButton = true]);
  Widget createBottomNavigationBar(int currentIndex, ValueSetter<int> onTap);
  Widget createLoader();
  PageRoute createPageRouter(WidgetBuilder builder);
  Widget createSwitcher(bool isActive, ValueSetter<bool> onChanged);
}
```

```dart
class CupertinoWidgetsFactory implements IPlatformWidgetsFactory {
  @override
  Widget createAppBar(String title, [bool showSettingsButton = true]) {
    return CupertinoAppBar(
      title: title,
      showSettingsButton: showSettingsButton,
    );
  }

  @override
  Widget createBottomNavigationBar(int currentIndex, ValueSetter<int> onTap) {
    return CupertinoBottomNavigationBar(
      currentIndex: currentIndex,
      onTap: onTap,
    );
  }

  @override
  Widget createLoader() {
    return CupertinoLoader();
  }

  @override
  PageRoute createPageRouter(WidgetBuilder builder) {
    return CupertinoPageRouter(
      builder: builder,
    );
  }

  @override
  Widget createSwitcher(bool isActive, ValueSetter<bool> onChanged) {
    return CupertinoSwitcher(
      isActive: isActive,
      onChanged: onChanged,
    );
  }
}
```

```dart
class MaterialWidgetsFactory implements IPlatformWidgetsFactory {
  @override
  Widget createAppBar(String title, [bool showSettingsButton = true]) {
    return MaterialAppBar(
      title: title,
      showSettingsButton: showSettingsButton,
    );
  }

  @override
  Widget createBottomNavigationBar(int currentIndex, ValueSetter<int> onTap) {
    return MaterialBottomNavigationBar(
      currentIndex: currentIndex,
      onTap: onTap,
    );
  }

  @override
  Widget createLoader() {
    return MaterialLoader();
  }

  @override
  PageRoute createPageRouter(WidgetBuilder builder) {
    return MaterialPageRouter(
      builder: builder,
    );
  }

  @override
  Widget createSwitcher(bool isActive, ValueSetter<bool> onChanged) {
    return MaterialSwitcher(
      isActive: isActive,
      onChanged: onChanged,
    );
  }
}
```

Abstract Factory: products

```dart
class CupertinoLoader extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: CupertinoActivityIndicator(),
    );
  }
}
```

```dart
class MaterialLoader extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Center(
      child: CircularProgressIndicator(
        valueColor: AlwaysStoppedAnimation(kBlackColor),
      ),
    );
  }
}
```

```dart
class CupertinoSwitcher extends StatelessWidget {
  final bool isActive;
  final ValueSetter<bool> onChanged;

  const CupertinoSwitcher({
    @required this.isActive,
    @required this.onChanged,
  });

  @override
  Widget build(BuildContext context) {
    return CupertinoSwitch(
      value: isActive,
      onChanged: onChanged,
    );
  }
}
```

```dart
class MaterialSwitcher extends StatelessWidget {
  final bool isActive;
  final ValueSetter<bool> onChanged;

  const MaterialSwitcher({
    @required this.isActive,
    @required this.onChanged,
  });

  @override
  Widget build(BuildContext context) {
    return Switch(
      value: isActive,
      onChanged: onChanged,
    );
  }
}
```

Abstract Factory: creating the factory

```dart
class App extends StatelessWidget {
  IPlatformWidgetsFactory _createPlatformWidgetsFactory() {
    switch (defaultTargetPlatform) {
      case TargetPlatform.android:
        return MaterialWidgetsFactory();
      case TargetPlatform.iOS:
        return CupertinoWidgetsFactory();
      default:
        return MaterialWidgetsFactory();
    }
  }

  @override
  Widget build(BuildContext context) {
    var widgetsFactory = _createPlatformWidgetsFactory();

    return MultiBlocProvider(
      ...
      child: Provider<IPlatformWidgetsFactory>.value(
        value: widgetsFactory,
        child: MaterialApp(
          title: 'DartUPify',
          theme: theme,
          onGenerateRoute: (settings) => AppRouter.generateRoute(
            settings,
            widgetsFactory,
          ),
          initialRoute: MainPage.route,
          debugShowCheckedModeBanner: false,
        ),
      ),
    );
  }
}
```

Abstract Factory: usage

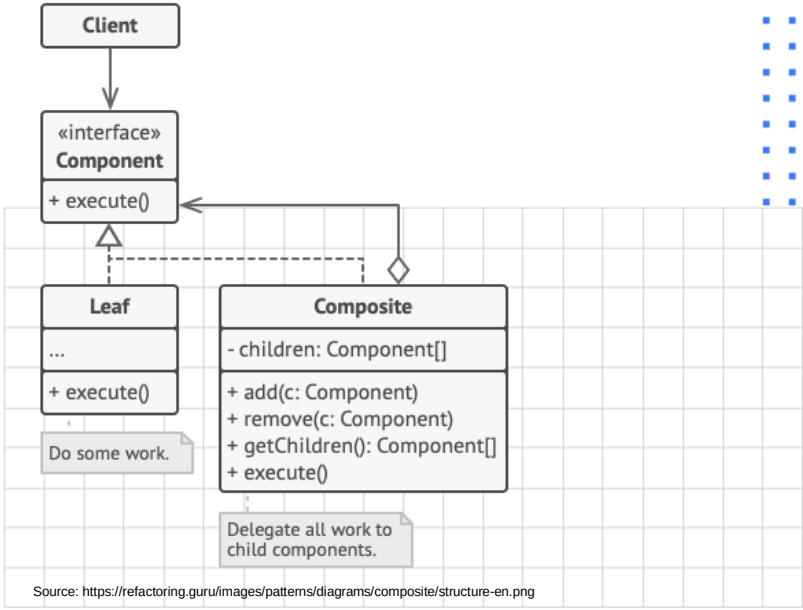
```dart
class SettingsPage extends StatelessWidget {
  static const String route = '/settings';

  void _onUseCupertinoWidgetsChanged(bool useCupertino) {
    var targetPlatform =
        useCupertino ? TargetPlatform.iOS : TargetPlatform.android;

    debugDefaultTargetPlatformOverride = targetPlatform;
    WidgetsBinding.instance.reassembleApplication();
  }

  @override
  Widget build(BuildContext context) {
    var widgetsFactory = context.watch<IPlatformWidgetsFactory>();

    return Scaffold(
      appBar: widgetsFactory.createAppBar('Settings', false),
      body: Column(
        children: <Widget>[
          ListTile(
            title: Text('Use Cupertino widgets'),
            trailing: widgetsFactory.createSwitcher(
              defaultTargetPlatform == TargetPlatform.iOS,
              _onUseCupertinoWidgetsChanged,
            ),
          ),
        ],
      ),
    );
  }
}
```

Composite

“Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.” - GoF

Composite

- Structural design pattern
- *Component* – declares the interface for objects in the composition
- *Leaf* – represents leaf objects in the composition
- *Composite* — stores sub-elements (children) and implements child-related operations in the Composite interface.



Source: https://refactoring.guru/images/patterns/diagrams/composite/structure-en.png

Composite: components

```dart
abstract class IMusicLibraryItem {
  int getItemsCount();
  int getDuration();
  Widget build(BuildContext context);
}
```

Leaf node

```dart
class MusicLibrarySong extends StatelessWidget implements IMusicLibraryItem {
  final Song data;

  const MusicLibrarySong({
    @required this.data,
    ValueKey key,
  }) : super(key: key);

  @override
  int getItemsCount() => 1;

  @override
  int getDuration() => data.duration;

  @override
  Widget build(BuildContext context) {
    ...
  }
}
```

Composite node

```dart
class MusicLibraryCollection extends StatelessWidget
    implements IMusicLibraryItem {
  final MusicCollection data;

  final List<IMusicLibraryItem> _items = <IMusicLibraryItem>[];

  MusicLibraryCollection({
    @required this.data,
  });

  void addItems(List<IMusicLibraryItem> items) {
    _items.addAll(items);
  }

  @override
  int getItemsCount() {
    return _items.fold<int>(
      0,
      (prev, item) => prev + item.getItemsCount(),
    );
  }

  @override
  int getDuration() {
    return _items.fold<int>(
      0,
      (prev, item) => prev + item.getDuration(),
    );
  }

  @override
  Widget build(BuildContext context) {
    ...
  }
}
```

Composite: building the composition

```dart
class MusicLibraryService {
  final MusicLibraryRepository repository;

  const MusicLibraryService({
    @required this.repository,
  });

  Future<List<IMusicLibraryItem>> getMusicLibraryItems() async {
    var collections = await repository.getCollections();
    var songs = await repository.getSongs();

    return _buildMusicLibraryItems(collections, songs);
  }
}
```

```dart
List<MusicLibraryCollection> _buildMusicLibraryItems(
    List<MusicCollection> collections,
    List<Song> songs,
  ) {
    var musicLibraryCollectionsMap = <int, MusicLibraryCollection>{
      for (var collection in collections)
        collection.id: MusicLibraryCollection(
          data: collection,
        )
    };

    for (var musicLibraryCollection in musicLibraryCollectionsMap.values) {
      var musicCollection = musicLibraryCollection.data;
      var parentId = musicCollection.parentId;

      if (musicLibraryCollectionsMap.containsKey(parentId)) {
        musicLibraryCollectionsMap[parentId].addItems([musicLibraryCollection]);
      }
    }

    _addSongsToCollections(musicLibraryCollectionsMap, songs);

    return musicLibraryCollectionsMap.values
        .where((musicLibraryCollection) =>
            musicLibraryCollection.data.parentId == null)
        .toList();
  }
```

Composite: rendering

As composition

```dart
class MusicLibraryPage extends StatelessWidget {
  static const route = '/music-library';

  final String title;
  final List<IMusicLibraryItem> musicLibraryItems;

  const MusicLibraryPage({
    @required this.title,
    @required this.musicLibraryItems,
  });

  @override
  Widget build(BuildContext context) {
    var widgetsFactory = context.watch<IPlatformWidgetsFactory>();

    return Scaffold(
      appBar: widgetsFactory.createAppBar(title),
      body: Column(
        children: [
          Expanded(
            child: ListView(
              children: musicLibraryItems
                  .map<Widget>((item) => item.build(context))
                  .toList(),
            ),
          ),
          ...
        ],
      ),
    );
  }
}
```

As stand-alone widget

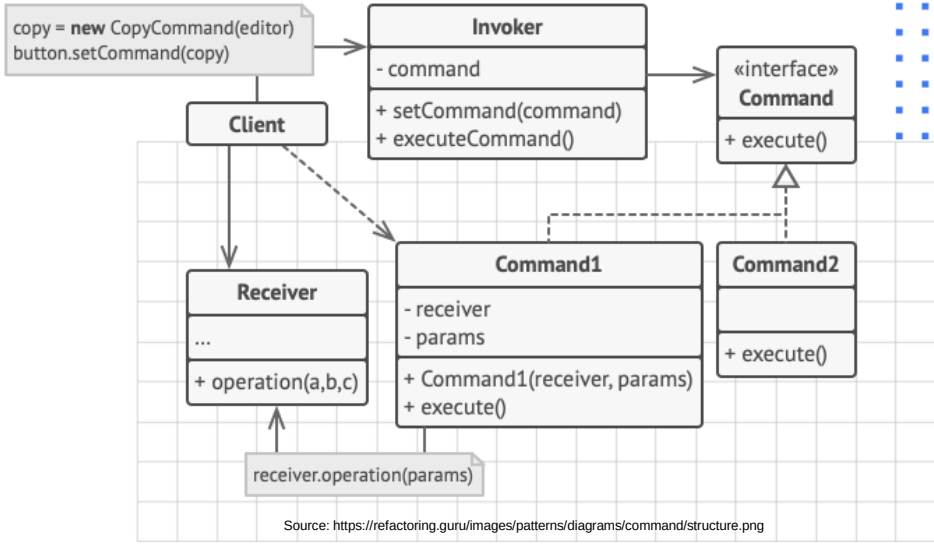
```dart
class PlaylistPage extends StatelessWidget {
  final Playlist playlist;

  const PlaylistPage({
    @required this.playlist,
  });

  @override
  Widget build(BuildContext context) {
    var widgetsFactory = context.watch<IPlatformWidgetsFactory>();

    return Scaffold(
      appBar: widgetsFactory.createAppBar('Playlist'),
      body: Column(
        children: [
          Expanded(
            child: ReorderableListView(
              children: [
                for (var song in playlist.songs)
                  MusicLibrarySong(
                    key: ValueKey(song),
                    data: song,
                  ),
              ],
              ...
            ),
          ),
          ...
        ],
      ),
    );
  }
}
```

Command

“Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.” - GoF

Command

- Behavioural design pattern
- *Command* – declares an interface for executing an operation
- *Command1, Command1* – implements the specific execution logic
- *Invoker* – triggers the command instead of sending the request directly to the *Receiver*
- *Receiver* – knows how to perform the operation



Source: https://refactoring.guru/images/patterns/diagrams/command/structure.png

Command: base

State (receiver)

```dart
class Playlist {
  final List<Song> songs;

  const Playlist({
    @required this.songs,
  });

  Playlist copyWith({List<Song> songs}) => Playlist(songs: songs ?? this.songs);

  @override
  String toString() {
    return 'Playlist { songs: $songs }';
  }
}
```

Base class

```dart
abstract class PlaylistCommand implements IPlaylistCommand {
  @protected
  final Playlist playlist;
  final List<Song> _songsBackup;

  PlaylistCommand(Playlist playlist)
      : this.playlist = playlist.copyWith(),
        _songsBackup = [...playlist.songs];

  @override
  Playlist undo() => playlist.copyWith(songs: _songsBackup);
}
```

Interface

```dart
abstract class IPlaylistCommand {
  Playlist execute();
  Playlist undo();
}
```

Command: commands

```dart
class AddToPlaylistCommand extends PlaylistCommand {
  final Song song;

  AddToPlaylistCommand({
    @required Playlist playlist,
    @required this.song,
  }) : super(playlist);

  @override
  Playlist execute() {
    playlist.songs.add(song);

    return playlist;
  }
}
```

```dart
class RemoveFromPlaylistCommand extends PlaylistCommand {
  final Song song;

  RemoveFromPlaylistCommand({
    @required Playlist playlist,
    @required this.song,
  }) : super(playlist);

  @override
  Playlist execute() {
    playlist.songs.remove(song);

    return playlist;
  }
}
```

```dart
class ReorderPlaylistCommand extends PlaylistCommand {
  final Song song;
  final int oldIndex;
  final int newIndex;

  ReorderPlaylistCommand({
    @required Playlist playlist,
    @required this.song,
    @required this.oldIndex,
    @required this.newIndex,
  }) : super(playlist);

  @override
  Playlist execute() {
    var insertAtIndex = newIndex > oldIndex ? newIndex - 1 : newIndex;

    playlist.songs.removeAt(oldIndex);
    playlist.songs.insert(insertAtIndex, song);

    return playlist;
  }
}
```

Command: execute

```dart
class PlaylistPage extends StatelessWidget {
  final Playlist playlist;

  const PlaylistPage({
    @required this.playlist,
  });

  @override
  Widget build(BuildContext context) {
    var widgetsFactory = context.watch<IPlatformWidgetsFactory>();

    return Scaffold(
      appBar: widgetsFactory.createAppBar('Playlist'),
      body: Column(
        children: [
          Expanded(
            child: ReorderableListView(
              children: [
                ...
              ],
              onReorder: (oldIndex, newIndex) {
                var command = ReorderPlaylistCommand(
                  playlist: playlist,
                  song: playlist.songs[oldIndex],
                  oldIndex: oldIndex,
                  newIndex: newIndex,
                );

                context.read<PlaylistCubit>().executeCommand(command);
              },
            ),
          ),
          ...
        ],
      ),
    );
  }
}
```

```dart
class PlaylistCubit extends Cubit<PlaylistState> {
  final Stack<IPlaylistCommand> commandHistory = Stack();

  PlaylistCubit() : super(PlaylistState.init());

  void executeCommand(IPlaylistCommand command) {
    commandHistory.push(command);

    var playlist = command.execute();
    var updatedState = state.copyWith(
      playlist: playlist,
      isCommandHistoryEmpty: false,
    );

    emit(updatedState);
  }

  ...
}
```

Command: undo

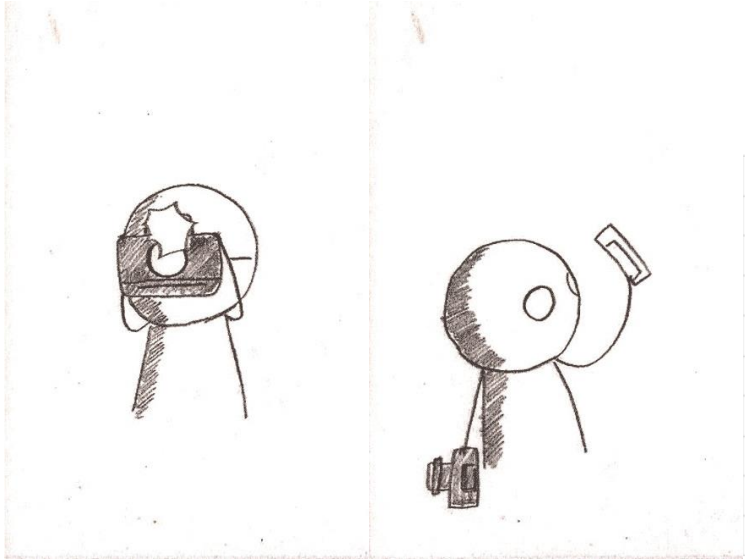
```dart
class MaterialAppBar extends StatelessWidget with PreferredSizeWidget {
  final String title;
  final bool showSettingsButton;

  const MaterialAppBar({
    @required this.title,
    this.showSettingsButton = true,
  });

  @override
  Widget build(BuildContext context) {
    return AppBar(
      backgroundColor: Colors.black,
      title: Text(title),
      actions: <Widget>[
        BlocBuilder<PlaylistCubit, PlaylistState>(
          builder: (context, state) {
            return Visibility(
              visible: !state.isCommandHistoryEmpty,
              child: IconButton(
                icon: Icon(Icons.replay),
                onPressed: () =>
                    context.read<PlaylistCubit>().undoLastCommand(),
              ),
            );
          },
        ),
        ...
      ],
    );
  }

  ...
}
```

```dart
class PlaylistCubit extends Cubit<PlaylistState> {
  final Stack<IPlaylistCommand> commandHistory = Stack();

  PlaylistCubit() : super(PlaylistState.init());

  ...

  void undoLastCommand() {
    if (commandHistory.isNotEmpty) {
      var playlistCommand = commandHistory.pop();
      var playlist = playlistCommand.undo();

      emit(state.copyWith(
        playlist: playlist,
        isCommandHistoryEmpty: commandHistory.isEmpty,
      ));
    }
  }
}
```

Dart UP 2020

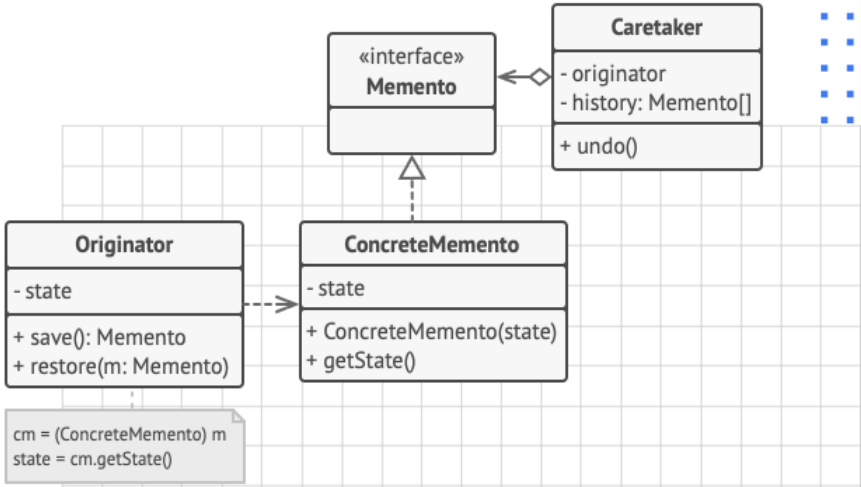
Memento

“Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.” - GoF



Memento

- Behavioural design pattern
- *Memento* – restricts access to the *ConcreteMemento's* fields
- *ConcreteMemento* – stores Originator's internal state
- *Caretaker* – only keeps the *Memento*, but never operates or examines its data
- *Originator* - creates a *ConcreteMememnto* containing a snapshot of its current internal state.



Source: https://refactoring.guru/images/patterns/diagrams/memento/structure2.png

Memento: components

```dart
abstract class Memento {
  Playlist getState();
}
```

```dart
class Originator {
  Playlist _state;
  Playlist get state => _state;

  Originator(Playlist playlist) : _state = playlist;

  Memento createMemento() => PlaylistMemento(_state);

  void restore(Memento memento) {
    _state = memento.getState();
  }
}
```

```dart
class PlaylistMemento implements Memento {
  final Playlist _state;

  PlaylistMemento(Playlist playlist) : _state = Playlist.copy(playlist);

  @override
  Playlist getState() => _state;
}
```

Memento: usage

Caretaker

```dart
abstract class PlaylistCommand implements IPlaylistCommand {
  @protected
  final Originator originator;
  @protected
  final Memento backup;

  PlaylistCommand(this.originator) : backup = originator.createMemento();

  @override
  Playlist undo() {
    originator.restore(backup);

    return originator.state;
  }
}
```

```dart
class PlaylistPage extends StatelessWidget {
  final Playlist playlist;

  const PlaylistPage({
    @required this.playlist,
  });

  @override
  Widget build(BuildContext context) {
    var widgetsFactory = context.watch<IPlatformWidgetsFactory>();

    return Scaffold(
      appBar: widgetsFactory.createAppBar('Playlist'),
      body: Column(
        children: [
          Expanded(
            child: ReorderableListView(
              ...
              onReorder: (oldIndex, newIndex) {
                var command = ReorderPlaylistCommand(
                  originator: Originator(playlist),
                  song: playlist.songs[oldIndex],
                  oldIndex: oldIndex,
                  newIndex: newIndex,
                );

                context.read<PlaylistCubit>().executeCommand(command);
              },
            ),
          ),
          ...
        ],
      ),
    );
  }
}
```

Memento: command without vs with memento

```dart
abstract class PlaylistCommand implements IPlaylistCommand {
  @protected
  final Playlist playlist;
  final List<Song> _songsBackup;

  PlaylistCommand(Playlist playlist)
      : this.playlist = playlist.copyWith(),
        _songsBackup = [...playlist.songs];

  @override
  Playlist undo() => playlist.copyWith(songs: _songsBackup);
}
```

```dart
abstract class PlaylistCommand implements IPlaylistCommand {
  @protected
  final Originator originator;
  @protected
  final Memento backup;

  PlaylistCommand(this.originator) : backup = originator.createMemento();

  @override
  Playlist undo() {
    originator.restore(backup);

    return originator.state;
  }
}
```

```dart
class AddToPlaylistCommand extends PlaylistCommand {
  final Song song;

  AddToPlaylistCommand({
    @required Playlist playlist,
    @required this.song,
  }) : super(playlist);

  @override
  Playlist execute() {
    playlist.songs.add(song);

    return playlist;
  }
}
```

```dart
class AddToPlaylistCommand extends PlaylistCommand {
  final Song song;

  AddToPlaylistCommand({
    @required Originator originator,
    @required this.song,
  }) : super(originator);

  @override
  Playlist execute() {
    var playlist = backup.getState();

    return playlist.copyWith(songs: [...playlist.songs]..add(song));
  }
}
```

Dart UP 2020

mkobuolys.medium.com

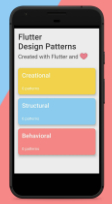
Flutter Design Patterns
Template Method

Flutter Design
Memento

Flutter Design Patterns
Strategy

Flutter D
Iterator

Flutter Design Patterns
Introduction



Flutter Design Patterns
Adapter

Flutter Design Patterns
Singleton

Flutter Design Patterns
Proxy

Flutter Design Patterns
Chain of Responsibility

Flutter Design Patterns
Prototype

Flutter Design Patterns
Composite

Flutter Design Patterns
Flyweight

Flutter Design Patterns
Facade

Flutter Desig
Decorator

Flutter Design Patterns
Factory Method

Flutter Design Patterns
State

Flutter Design P
Command

Flutter Design Patterns
Interpreter

Flutter Design
Bridge

Flutter Design Patte
Builder

Flutter Design Patterns
Abstract Factory

Flutter Design Patterns
Visitor

Dart UP 2020

Thank you!

@mkobuolys

mkobuolys.medium.com

mangirdas.kazlauskas