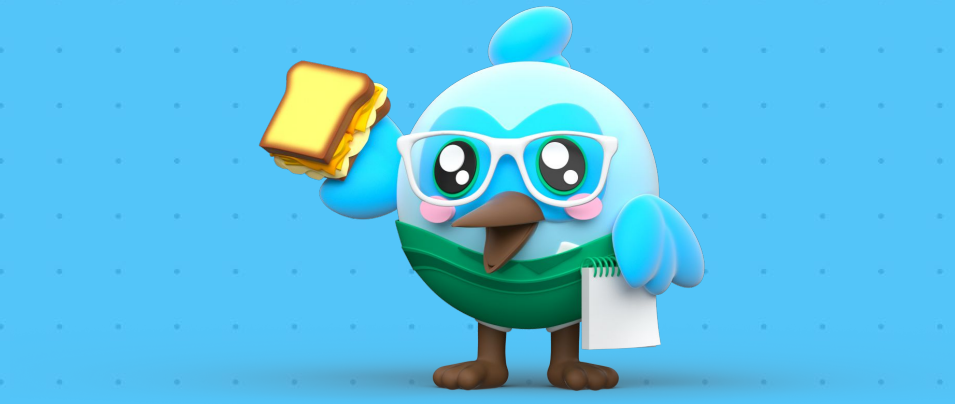
Let’s demystify Dart Patterns!

Guillaume Roux

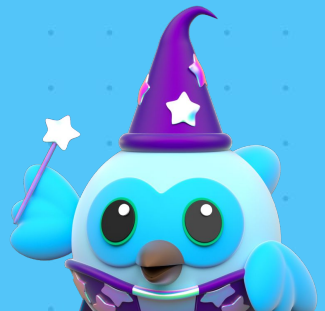
Mobile Engineer at Shipfix
Co-organizer of Flutter Lyon

@TesteurManiak



What are Patterns in Dart?

“Patterns are a syntactic category in the Dart language, like statements and expressions. A pattern represents the shape of a set of values that it may match against actual values.”



https://dart.dev/language/patterns

```
<value> case <pattern>
```

```
switch (someObject) {
  case String():
    print('a String');
  case int():
    print('an int');
  default:
    print('something else');
}
```

```
switch (someObject) {
  case String():
    print('a String');
  case int():
    print('an int');
  default:
    print('something else');
}
```

Value

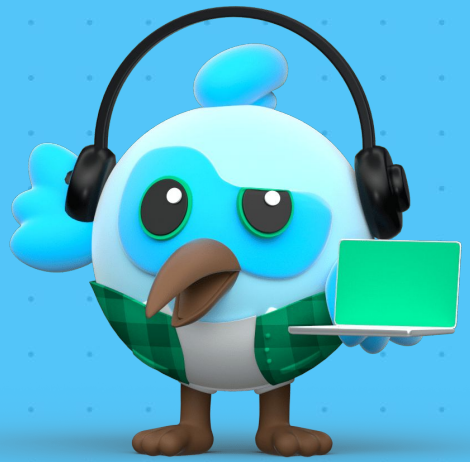
```
switch (someObject) {
  case String():
    print('a String');
  case int():
    print('an int');
  default:
    print('something else');
}
```

Patterns

*"[...] a pattern may **match** a value, **destructure** a value, or **both**"*

"[...] pattern matching allows you to check whether a given value:

- *Has a certain shape.*
- *Is a certain constant.*
- *Is equal to something else.*
- *Has a certain type."*

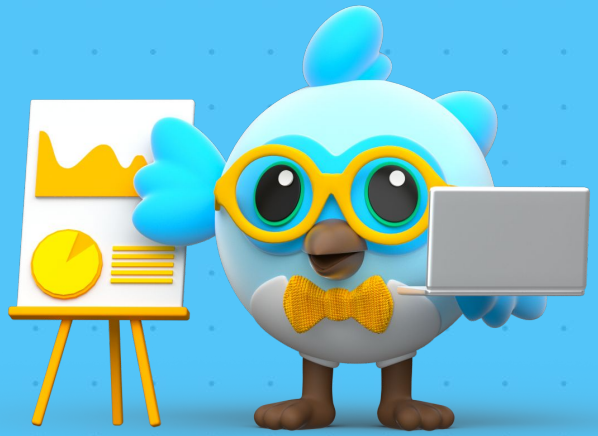


https://dart.dev/language/patterns#what-patterns-do

But what can you do with it?

A pattern can be used for:

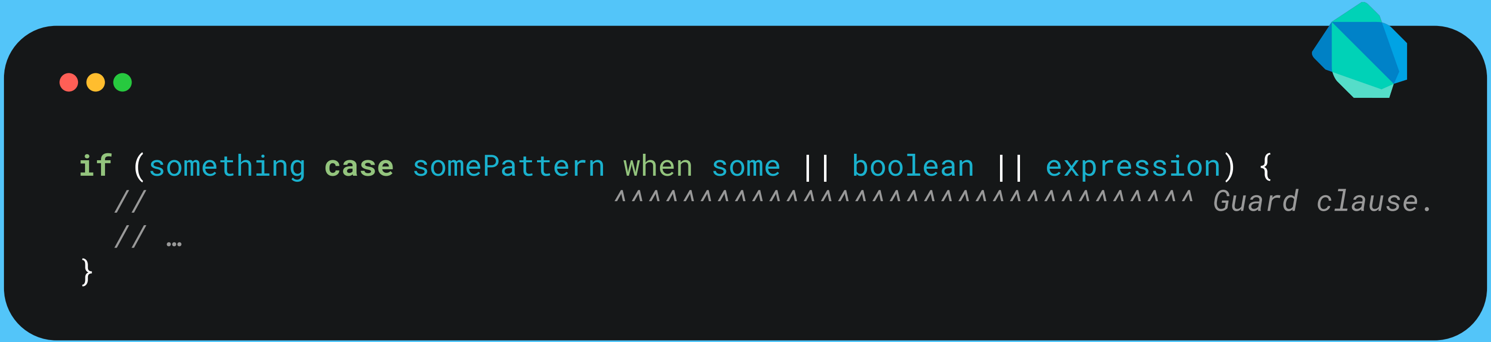
- Variable declarations & assignments
- **for** and **for-in** loops
- **if-case** and **switch-case**
- Control flow in **collection literals**



https://dart.dev/language/patterns#places-patterns-can-appear

Guard clause

“To set an optional guard clause after a `case` *clause, use the keyword* `when`*. A guard clause can follow* `if case`*, and both* `switch` *statements and expressions.”*



```
if (something case somePattern when some || boolean || expression) {
  //                           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ Guard clause.
  // …
}
```

https://dart.dev/language/branches#guard-clause

Destructuration

```
List<int> list = [1, 2, 3];

final result = switch (list) {
    [..., final item, _] => 'Item before last: $item',
    [] || [_] => 'List is too small',
};
```

List matching & destructuration

```
List<int> list = [1, 2, 3];

final result = switch (list) {
    [..., final item, _] => 'Item before last: $item',
    [] || [_] => 'List is too small',
};
```

List matching & destructuration

```
List<int> list = [1, 2, 3];

final result = switch (list) {
    [..., final item, _] => 'Item before last: $item',
    [] || [_] => 'List is too small',
};
```

List matching & destructuration

```
List<int> list = [1, 2, 3];

final result = switch (list) {
    [..., final item, _] => 'Item before last: $item',
    [] || [_] => 'List is too small',
};
```

List matching & destructuration

```
if (json case {'user': [String name, int age]}) {
  print('User $name is $age years old.');
}
```

Map matching & destructuration

```
if (json case {'user': [String name, int age]}) {
  print('User $name is $age years old.');
}
```

Map matching & destructuration

```
class Data {
  final NestedData data;
}

class NestedData {
  final String content;
}

final String content = switch (someObj) {
  Data(data: NestedData(:final content)) => content,
  _ => 'Empty',
};
```

Nested destructuration

```
class Data {
  final NestedData data;
}

class NestedData {
  final String content;
}

final String content = switch (someObj) {
  Data(data: NestedData(:final content)) => content,
  _ => 'Empty',
};
```

Nested destructuration

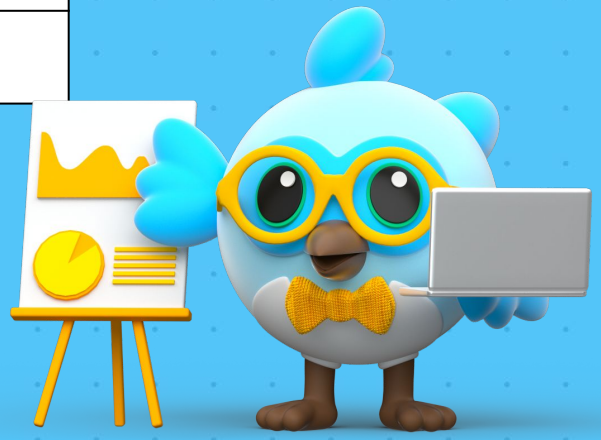
```
class Data {
  final NestedData data;
}

class NestedData {
  final String content;
}

final String content = switch (someObj) {
  Data(data: NestedData(:final content)) => content,
  _ => 'Empty',
};
```

Nested destructuration

?	Match if the value is not null
_	Wildcard, match any value
...	Represent elements from an iterable
:	Access inner object properties



Concrete use-cases

```dart
final int? maybeInt = 42;
if (maybeInt case final x?) {
  debugPrint('$x');
}
```

Null guard

```dart
final int? maybeInt = 42;
if (maybeInt case final x?) {
  debugPrint('$x');
}
```

Null guard

```dart
final int? maybeInt = 42;
if (maybeInt case final x?) {
  debugPrint('$x');
}
```

Null guard

```
void process(int? a, int? b) {
  if ((a, b) case (final a?, final b?)) {
    debugPrint('${a + b}');
  }
}
```

Null guard (with record)

```
void process(int? a, int? b) {
  if ((a, b) case (final a?, final b?)) {
    debugPrint('${a + b}');
  }
}
```

Null guard (with record)

```
void process(int? a, int? b) {
  if ((a, b) case (final a?, final b?)) {
    debugPrint('${a + b}');
  }
}
```

Null guard (with record)

```
void process(int? a, int? b) {
  if ((a, b) case (final a?, final b?)) {
    debugPrint('${a + b}');
  }
}
```

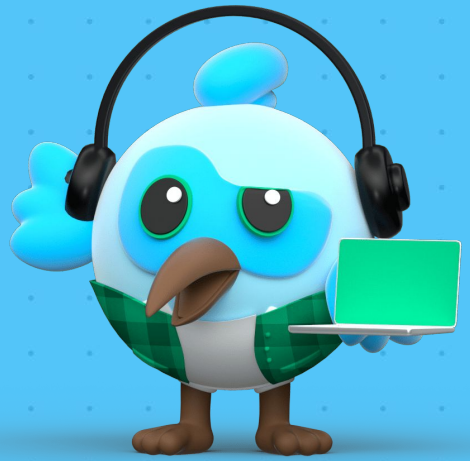
Null guard (with record)

```dart
switch (json) {
  {'response': final Map<String, dynamic> json} => parse(json),
  // ...
}
```

JSON validation

Why use pattern matching?

- Do more with less
- Makes your code more readable
- Helps to improve type-safety
- *Might* make your code more performant



```
final int? maybeInt = 42;
if (maybeInt case final x?) {
  print(x);
}
```

```
let maybeInt: Int? = 42
if case let x? = maybeInt {
  print(x)
}
```

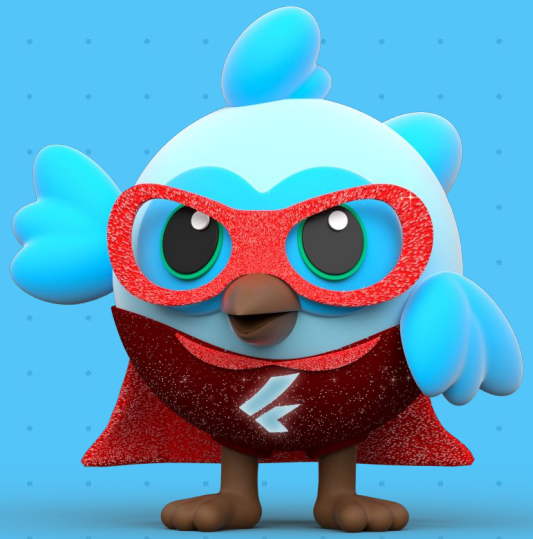
Still some limitations

```dart
if (json case {'optionalKey'?: final String? value}) {
  // ...
}
```

No optional key check

Learn more!

- Patterns | Dart
- Pattern types | Dart
- Collections | Dart
- Branches | Dart
- Dive into Dart's patterns and records



Thank You!

Questions?

Guillaume Roux

Mobile Engineer at Shipfix
Co-organizer of Flutter Lyon

@TesteurManiak