



Testes

Imagem gerada por AI

Testes em Flutter

- Testes unitários

Testam uma “unidade” de forma isolada (função, classe)

- Testes de widget

Testam um widget de forma isolada

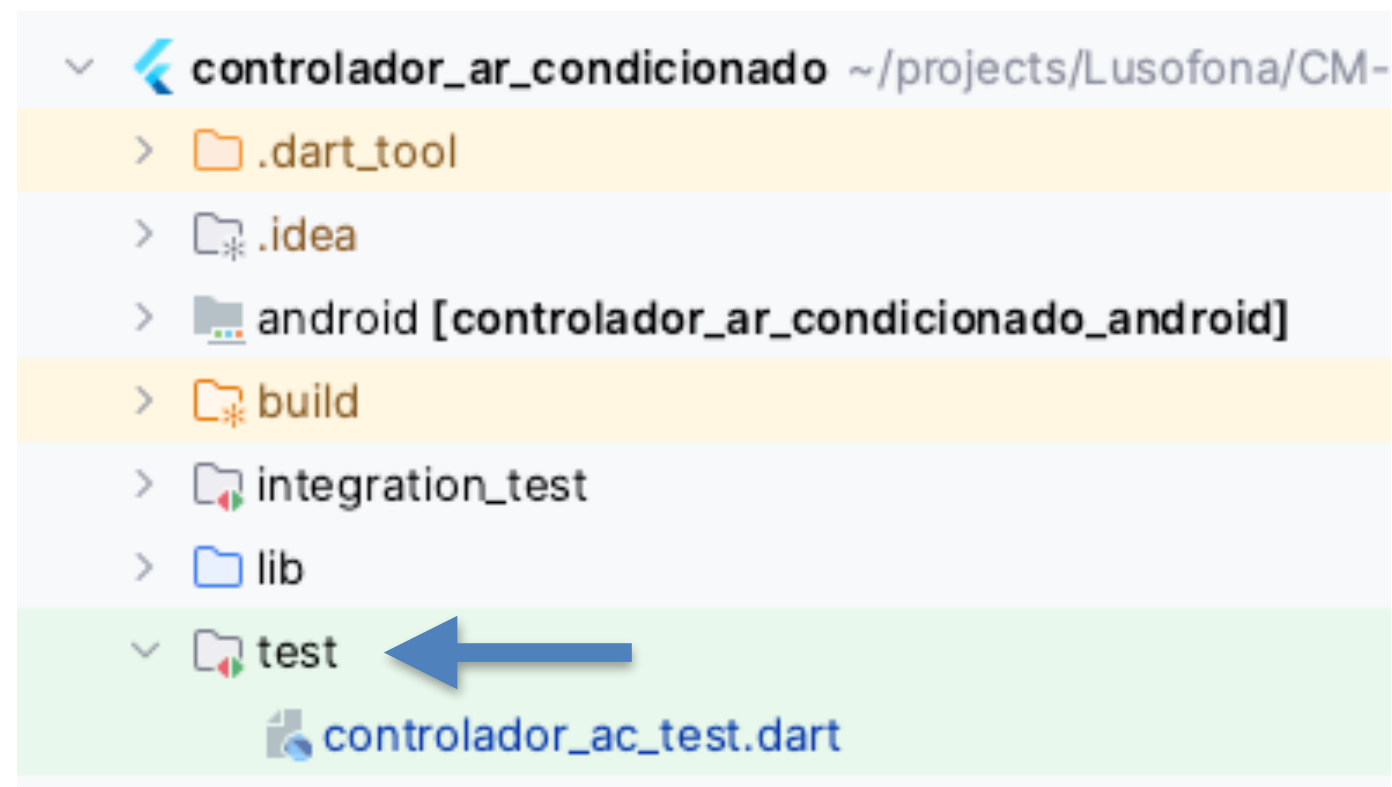
- Testes de integração

Testam a aplicação toda

Testes em Flutter

| | Dependências | Precisa emulador? | Execução |
|----------------------|-----------------------------------|-------------------|----------|
| Testes unitários | nenhuma | não | rápida |
| Testes de widget | flutter_test | não | rápida |
| Testes de integração | flutter_test, integration_test | sim | lenta |

Testes unitários em Flutter



Testes unitários em Flutter

```
void main() {  
  
  test("construtor inicializa bem o valor", () {  
    final controladorAC = ControladorAC(20);  
    expect(controladorAC.temperatura, 20);  
  });  
  
  test("aumenta", () {  
    final controladorAC = ControladorAC(20);  
  
    controladorAC.aumenta();  
    expect(controladorAC.temperatura, 21);  
  });  
  
}
```

Testes unitários em Flutter

```
test(String description, Function body)
```

| Junit | Dart Unit Tests |
|--|---|
| <code>assertEquals(number, 1)</code> <code>assertEquals(text, "hello")</code> | <code>expect(number, 1)</code> <code>expect(text, 'hello')</code> |
| <code>assertNull(obj)</code> <code>assertNotNull(obj)</code> | <code>expect(obj, isNull)</code> <code>expect(obj, isNotNull)</code> |
| <code>assertTrue(expr)</code> <code>assertFalse(expr)</code> | <code>expect(expr, isTrue)</code> <code>expect(expr, isFalse)</code> |
| <code>fail(message)</code> | <code>fail(message)</code> |
| ... | ... |

<https://cms.invertase.io/wp-content/uploads/2023/03/cheat-sheet.png>

Agrupar testes

```
void main() {  
  
    group("grupo de testes", () {  
        test("teste 1", () {  
            ...  
        });  
  
        test("teste 2", () {  
            ...  
        });  
    });  
  
}
```

Testes de widgets/integração

- Cria/executa widgets

```
pumpWidget(MyWidget())
```

- Encontra widgets

```
find.byType(ElevatedButton), ...
```

- Testar o “conteúdo” dos widgets

```
tester.widget(...)
```

- Interagir com widgets

```
tester.tap(...), ...
```


Testes de widgets

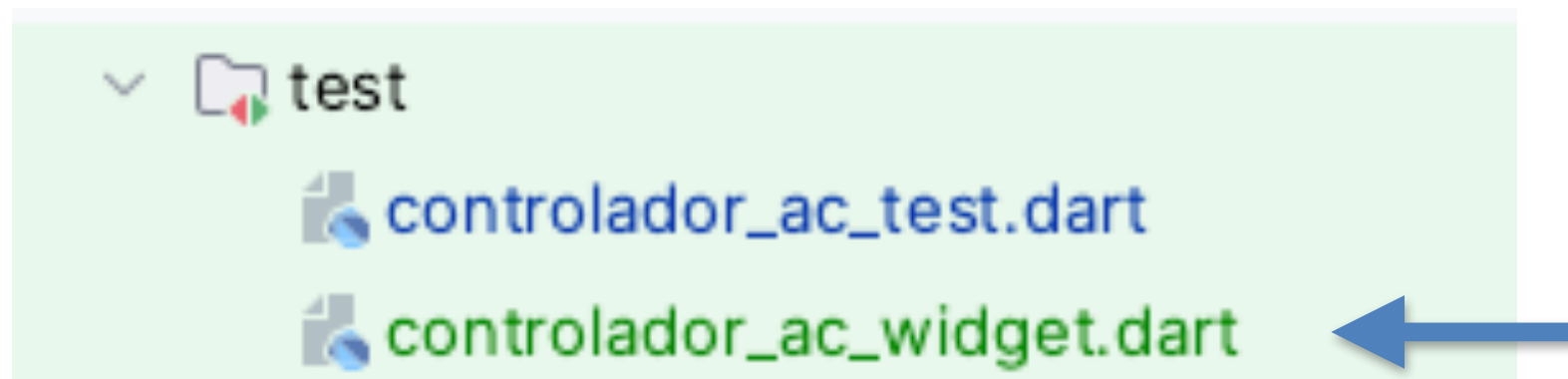
Os testes de Widgets devem testar Widgets de forma isolada

Imaginando que quero testar o widget **MyWidget**

```
void main() {  
  testWidgets('MyWidget has a title and message', (tester) async {  
    await tester.pumpWidget(const MyWidget(title: 'Title', message: 'Msg'));  
    final titleFinder = find.text('Title');  
    final messageFinder = find.text('Msg');  
  
    expect(titleFinder, findsOneWidget);  
    expect(messageFinder, findsOneWidget);  
  });  
}
```

Testes de widgets

Devem ser colocados na pasta test, tal como os testes unitários



Testes de widget

Necessário acrescentar esta dependência no pubspec.yaml

```
dev_dependencies:  
  flutter_test:  
    sdk: flutter
```

Testes de integração

Testam a aplicação inteira ou partes da aplicação, executando-a num emulador e simulando interações

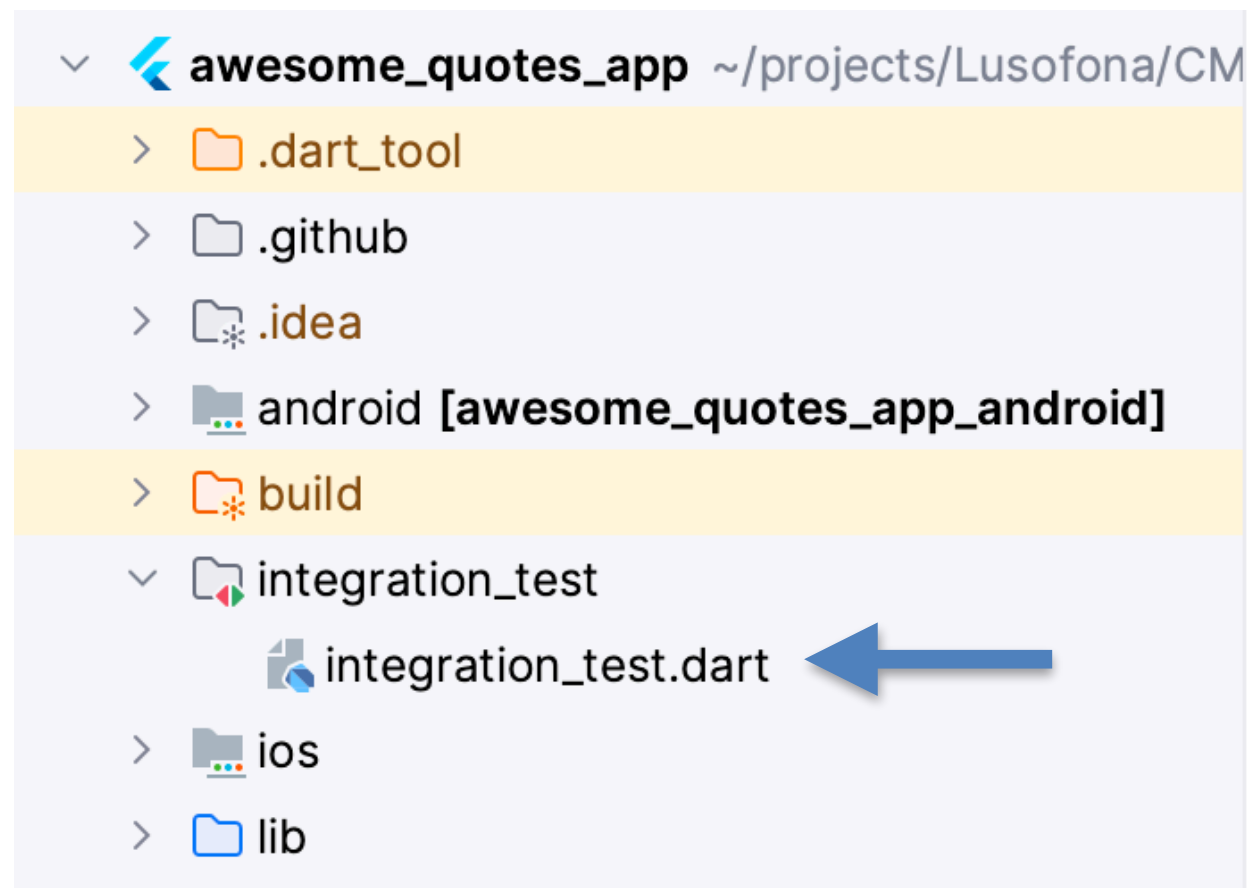
A programação é similar aos testes de widget

Necessário acrescentar estas dependências no pubspec.yaml

```
dev_dependencies:  
  flutter_test:  
    sdk: flutter  
  
  integration_test:  
    sdk: flutter
```

Testes de integração

Devem estar numa pasta própria, chamada integration_test



Testes de widgets/integração

Como a programação é similar, os slides seguintes são válidos quer para testes de widget quer para testes de integração

Testes de widgets/integração

```
testWidgets('descrição', (tester) async {  
  // Test code goes here.  
});
```

Testes de widgets/integração

Encontrar widgets

Usa-se o objeto `find` (global), fornecido pela biblioteca de testes

```
Finder finder1 = find.text('Ola');
```

```
Finder finder2 = find.byType(ElevatedButton);
```

```
Finder finder3 = find.byKey(Key('widget-key'));
```

Testes de widgets/integração

Encontrar widgets

Usa-se o objeto `find` (global), fornecido pela biblioteca de testes

```
Finder finder1 = find.text('Ola');
```

```
Finder finder2 = find.byType(ElevatedButton);
```

```
Finder finder3 = find.byKey(Key('widget-key'));
```

Usar este método, sempre que possível

Testes de widgets/integração

Encontrar widgets

código dos testes

```
Finder finderGravarBtn = find.byKey(Key('gravar-btn'));
```

```
@override
Widget build(BuildContext context) {
  return ElevatedButton(
    key: Key('gravar-btn'),
    onPressed: () {},
    child: Text('Gravar'),
  );
}
```

código do widget

Nota: chave deve ser única!

© Pedro Alves 2025

Testes de widgets/integração

Encontrar widgets

```
Finder finder2 = find.byType(ElevatedButton);
```

O finder pode encontrar 0, 1 ou mais widgets que satisfaçam a condição.

Deve-se começar por verificar quantos widgets encontrou

```
expect(finder2, findsNothing);  
expect(finder2, findsOneWidget);  
expect(finder2, findsNWidgets(3));
```

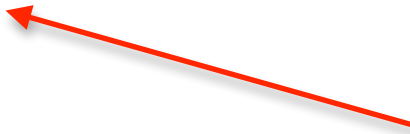
Testes de widgets/integração

Testar o “conteúdo” dos widgets

```
Finder finderQuoteText = find.byKey('quote-text');
```

```
Text quoteText = tester.widget(finderQuoteText);
```

```
String? quote = quoteText.data;  
expect(quote, isNotNull);  
expect(quote, 'Texto da quote');
```



Permite obter o widget encontrado por este finder (assumindo que é único)

Testes de widgets/integração

Testar o “conteúdo” dos widgets (avanzado)

```
ElevatedButton(  
  key: Key('like-btn'),  
  onPressed: () {},  
  child: Text('Like'),  
)
```

Como testar que o botão com a chave 'like-btn' tem um filho com um certo texto?

Testes de widgets/integração

Testar o “conteúdo” dos widgets (avançado)

```
ElevatedButton(  
  key: Key('like-btn'),  
  onPressed: () {},  
  label: Text('Like'),  
)
```

Como testar que o botão com a chave 'like-btn' tem um filho com um certo texto?

```
final Finder buttonFinder = find.byKey(Key('like-btn'));  
final Finder textInsideButtonFinder =  
  find.descendant(of: buttonFinder, matching: find.byType(Text));  
Text textInsideButton = tester.widget(textInsideButtonFinder);  
expect(textInsideButton.data, "Like");
```

Testes de widgets/integração

Interagir com os widgets

```
Finder finderSaveBtn = find.byKey('save-btn');
```

```
await tester.tap(finderSaveBtn);
```

```
await tester.pump();
```



Temos que explicitamente dizer para “refrescar” o ecrã

Testes de widgets/integração

Interagir com os widgets


```
Finder finderSaveBtn = find.byKey('save-btn');
```

```
await tester.tap(finderSaveBtn);
```

```
await tester.pump();
```

```
await tester.pumpAndSettle();
```

Se um evento gerar
vários “refresh”,
podemos usar esta
função



Testes de widgets/integração

Interagir com os widgets

```
// enter username and password  
await tester.enterText(find.byKey(Key('emailTextField')), email);  
await tester.enterText(find.byKey(Key('passwordTextField')), password);  
await tester.tap(find.byKey(Key('signInButton')));
```

Testes de integração

Diferença importante entre testes de widget e testes de integração

```
void main() {
```

```
  IntegrationTestWidgetsFlutterBinding.ensureInitialized();
```

```
  testWidgets('Login and show list', (tester) async {  
    // Test code goes here.  
  });  
}
```


Na prática...

- Uma vez que os testes de widget conseguem testar qualquer widget, também podem testar a aplicação completa, pois ela está encapsulada num Widget (o famoso “MyApp”)
- Como executam muito mais rápido, mais vale fazer os testes usando testes de widget
- Porque se usam então testes de integração?

Testes de integração

(mesmo assim são úteis)

- Pode ser difícil perceber o que está a correr mal num teste de Widget (ex: diz que não encontra o Widget no ecrã mas achamos que ele está lá). Nos testes de integração, conseguimos visualizar o que está a acontecer.
- Há coisas que não conseguimos testar com testes de widget (ex: acesso a uma base de dados local, geo-localização, etc...)
- Temos mais garantias que a aplicação vai funcionar bem se a testarmos num emulador (ou até em vários...)

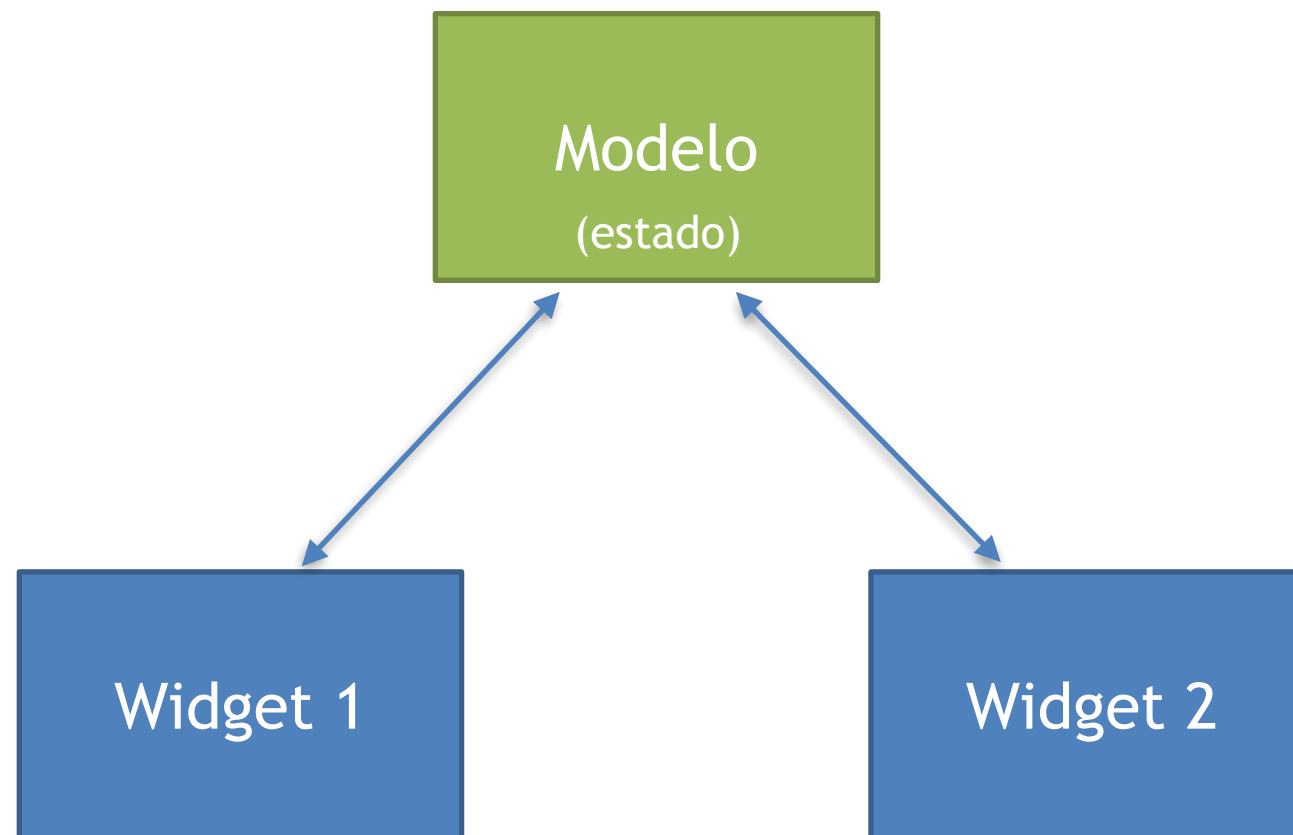
Injeção de dependências



Imagem gerada por AI 🤖

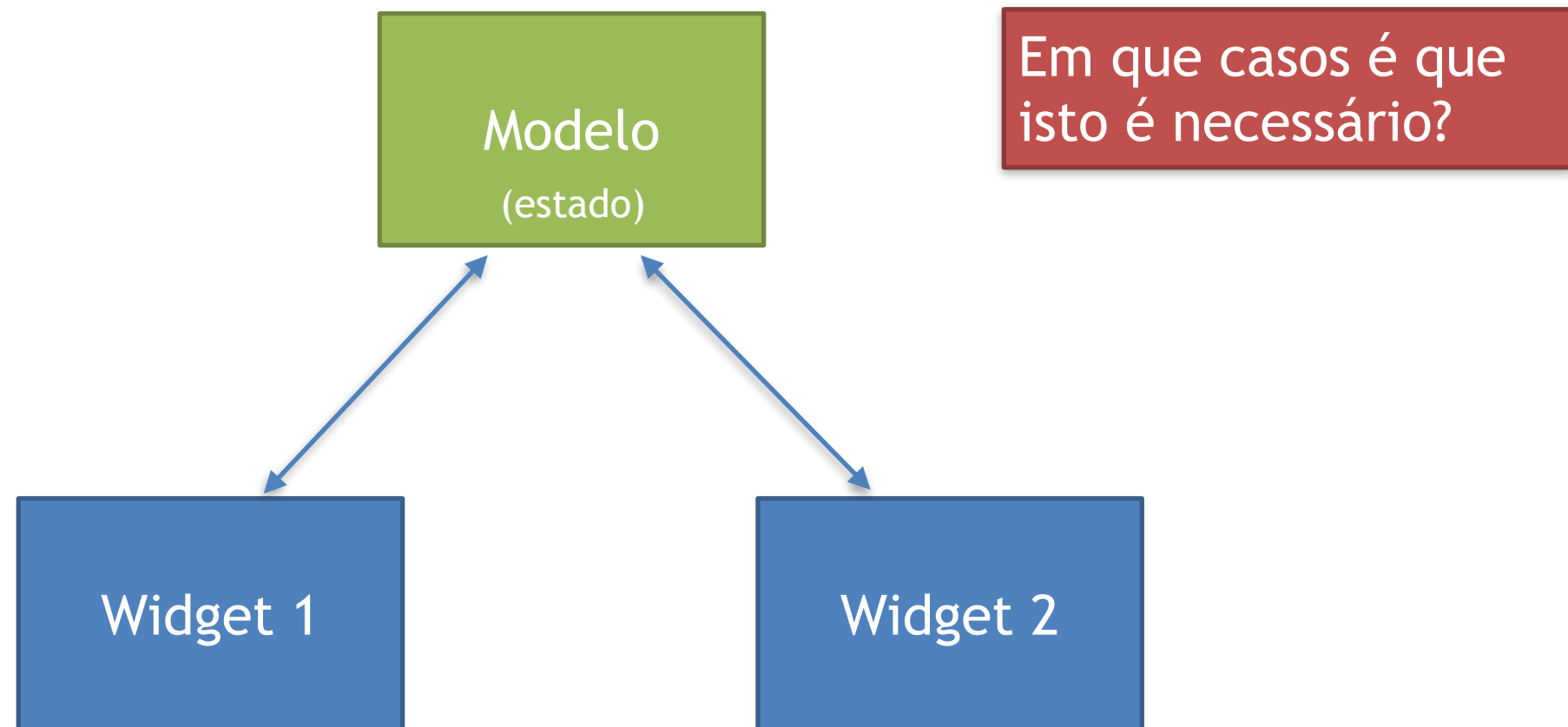
Problema 1

Vários Widgets precisam de aceder ao mesmo modelo
(objeto com dados que contém estado da aplicação)



Problema 1

Vários Widgets precisam de aceder ao mesmo modelo
(objeto com dados que contém estado da aplicação)



Problema 1

Vários Widgets precisam de aceder ao mesmo modelo
(objeto com dados que contém estado da aplicação)

- Lista de items e respetivo detalhe
- Formulário de registo e lista com registos
- Dashboard e lista de items
- ...

Lista de items e respectivo detalhe



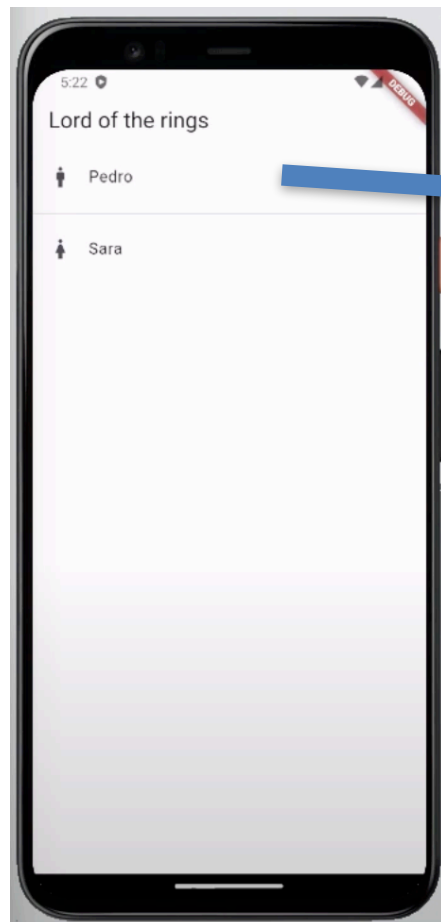
Passar o modelo através do construtor

```
onTap: push(PersonagemDetailPage(personagens[0]))
```

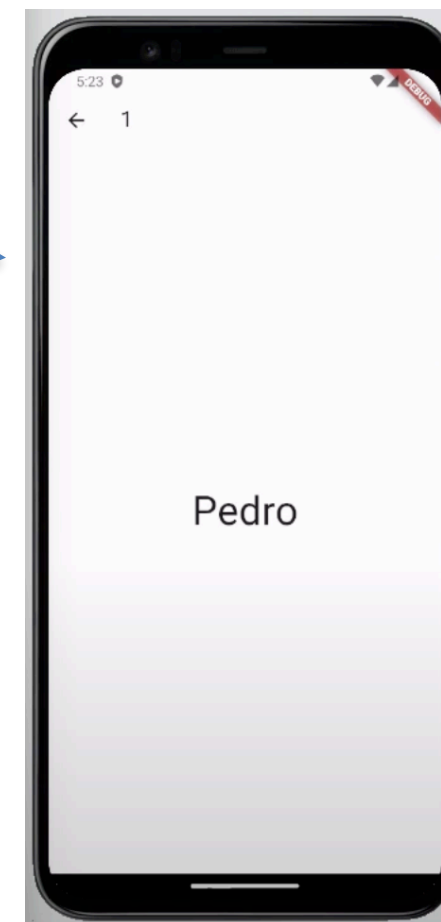
(código simplificado)

List personagens

Personagem personagem



personagens[0]



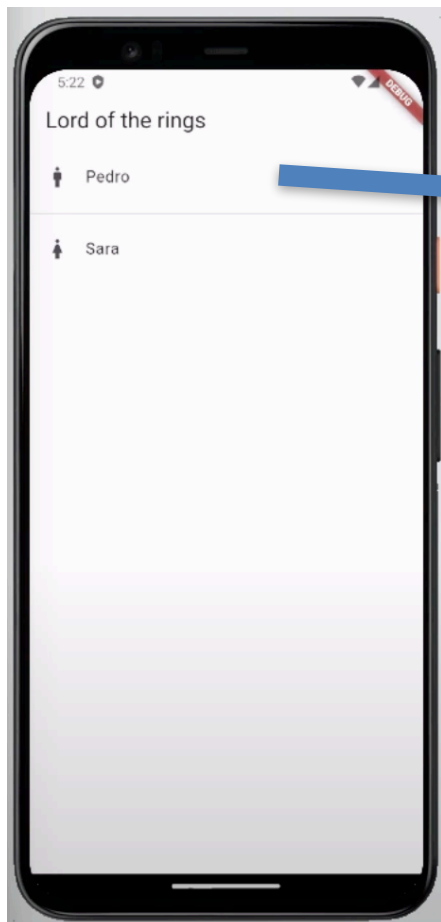
Passar o modelo através do construtor

(alternativa)

```
onTap: push(PersonagemDetailPage(personagens[0].id))
```

(código simplificado)

List personagens



`personagens[0].id`

Personagem personagem



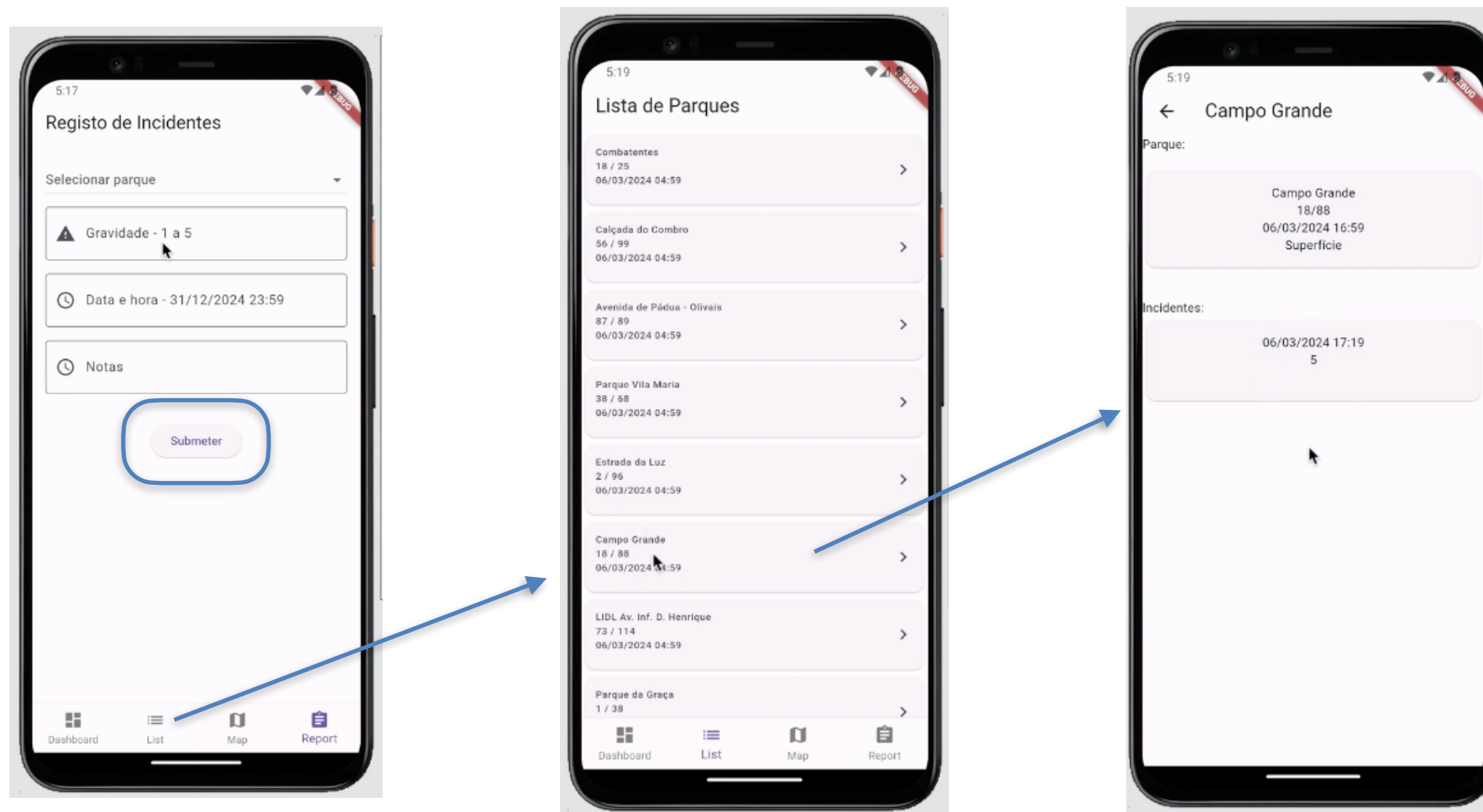
vai ao servidor/BD
buscar os dados do
personagem com
esse id

Lista de items e respetivo detalhe

Duas opções:

- Passar ao construtor da página de detalhe o objeto completo
- Passar ao construtor da página de detalhe o id do objeto e obter os dados a partir desse id

Formulário e lista



Formulário e lista

Não dá para passar o modelo através do construtor
(o widget da lista já foi construído e “vive” na bottom bar)

Incidente incidente

5:17

Registo de Incidentes

Selecionar parque

Gravidade - 1 a 5

Data e hora - 31/12/2024 23:59

Notas

Submeter

Dashboard List Map Report

List incidentes

5:19

Lista de Parques

| | |
|---|---|
| Combatentes 18 / 25 06/03/2024 04:59 | > |
| Calçada do Combro 56 / 99 06/03/2024 04:59 | > |
| Avenida de Pádua - Olivais 87 / 89 06/03/2024 04:59 | > |
| Parque Vila Maria 38 / 68 06/03/2024 04:59 | > |
| Estroada da Luz 2 / 96 06/03/2024 04:59 | > |
| Campo Grande 18 / 88 06/03/2024 11:59 | > |
| LIDL Av. Inf. D. Henrique 73 / 114 06/03/2024 04:59 | > |
| Parque da Graça 1 / 38 | > |

Dashboard List Map Report

5:19

← Campo Grande

Parque:

Campo Grande
18/88
06/03/2024 16:59
Superfície

Incidentes:

06/03/2024 17:19
5

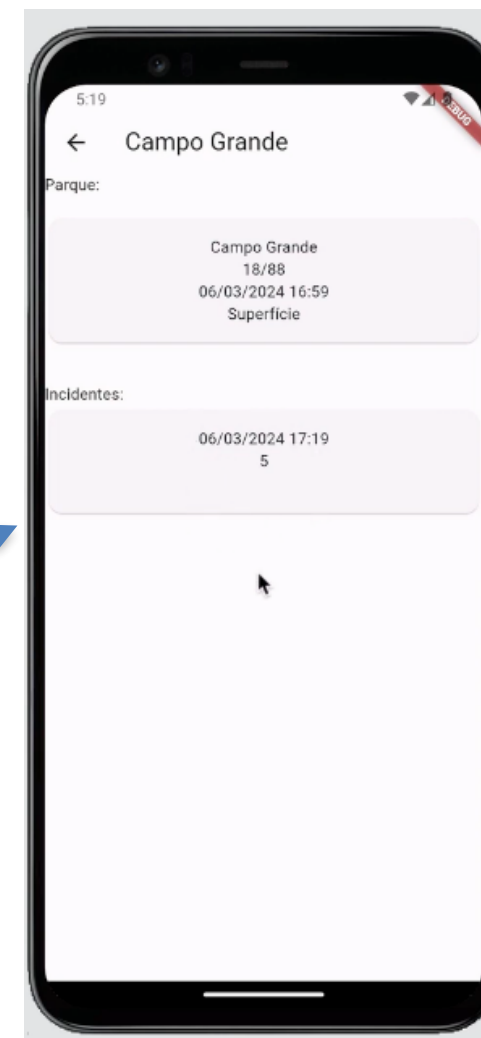
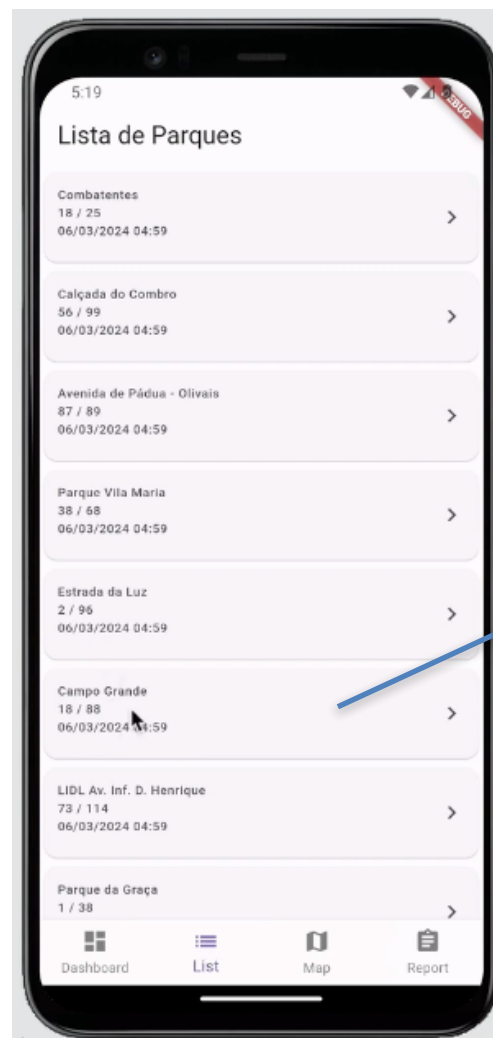
Singleton

List incidentes
(singleton)

Nota: A lista de incidentes
tem que estar associada a
um parque

Incidente incidente

List incidentes



Singleton

Um objeto que representa uma instância única em toda a aplicação

Exercício

```
// representa o utilizador autenticado  
class UtilizadorSessao {  
  
    String _nome;  
  
    UtilizadorSessao(this._nome);  
  
    String get nome => _nome;  
    set nome(String value) => _nome = value;  
}
```

É possível criar um singleton a partir desta classe?

Resolução

```
// representa o utilizador autenticado  
class UtilizadorSessao {  
  
    String _nome;  
  
    UtilizadorSessao(this._nome);  
  
    String get nome => _nome;  
    set nome(String value) => _nome = value;  
}
```

É possível criar um singleton a partir desta classe?

Não! Consegui criar várias instâncias da classe UtilizadorSessao

```
void main() {  
    final utilizador1 = UtilizadorSessao('Pedro');  
    final utilizador2 = UtilizadorSessao('Cris');  
}
```

Tornar o construtor privado

```
// representa o utilizador autenticado
class UtilizadorSessao {

    String _nome;

    UtilizadorSessao.__(this._nome);

    String get nome => _nome;
    set nome(String value) => _nome = value;
}
```

Erro de compilação a
instanciar

```
final utilizador1 = UtilizadorSessao('Pedro');
final utilizador2 = UtilizadorSessao('Cris');
```

Tornar o objeto acessível a todos

// representa o utilizador autenticado

```
class UtilizadorSessao {
```

```
    String? _nome;
```

```
    UtilizadorSessao.__( );
```

← Objeto passa a ser criado “vazio”

```
    String? get nome => _nome;
```

```
    set nome(String? value) => _nome = value;
```

// única instância

```
    static final UtilizadorSessao _instance = UtilizadorSessao.__( );
```

guardar a única instância
numa variável static (global)

// Getter para aceder à instância

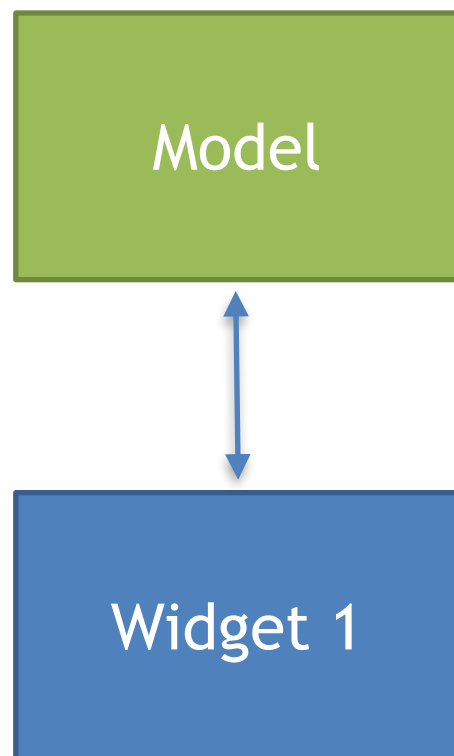
```
    static UtilizadorSessao get instance => _instance;
```

```
}
```

```
UtilizadorSessao.instance.nome = 'Pedro';
```

Widget instancia Model

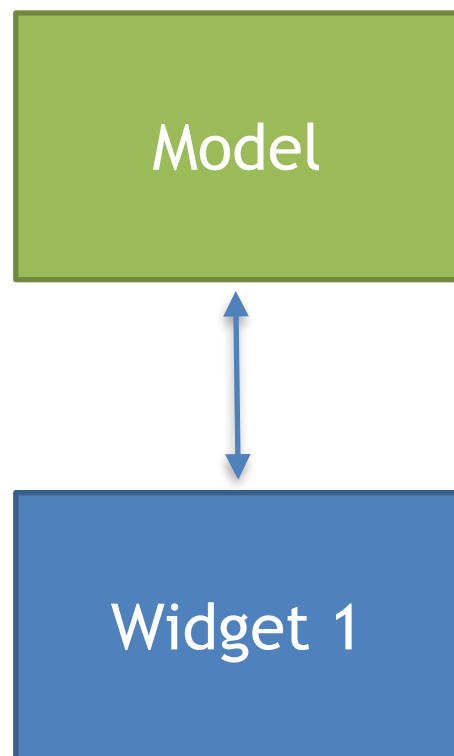
Se o model é apenas usado por um widget então ele pode instanciá-lo



```
class Widget1 {  
    Model _model = Model(...);  
    ...  
}
```

Widget instancia Model

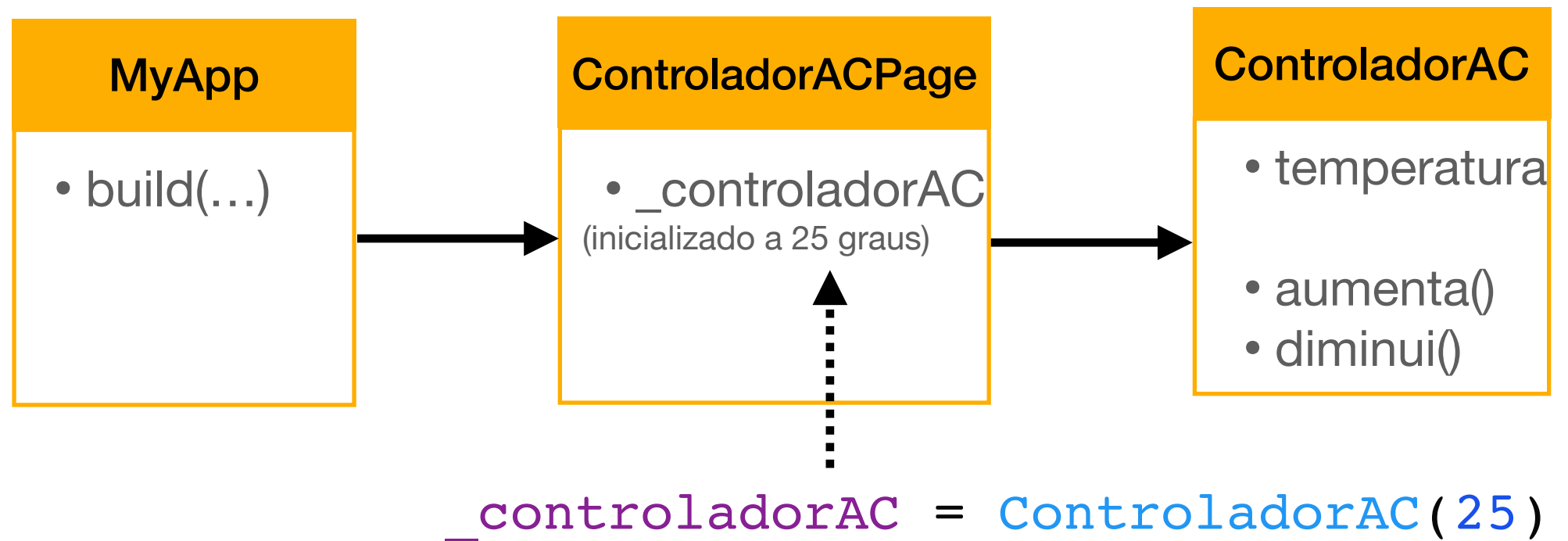
Se o model é apenas usado por um widget então ele pode instanciá-lo



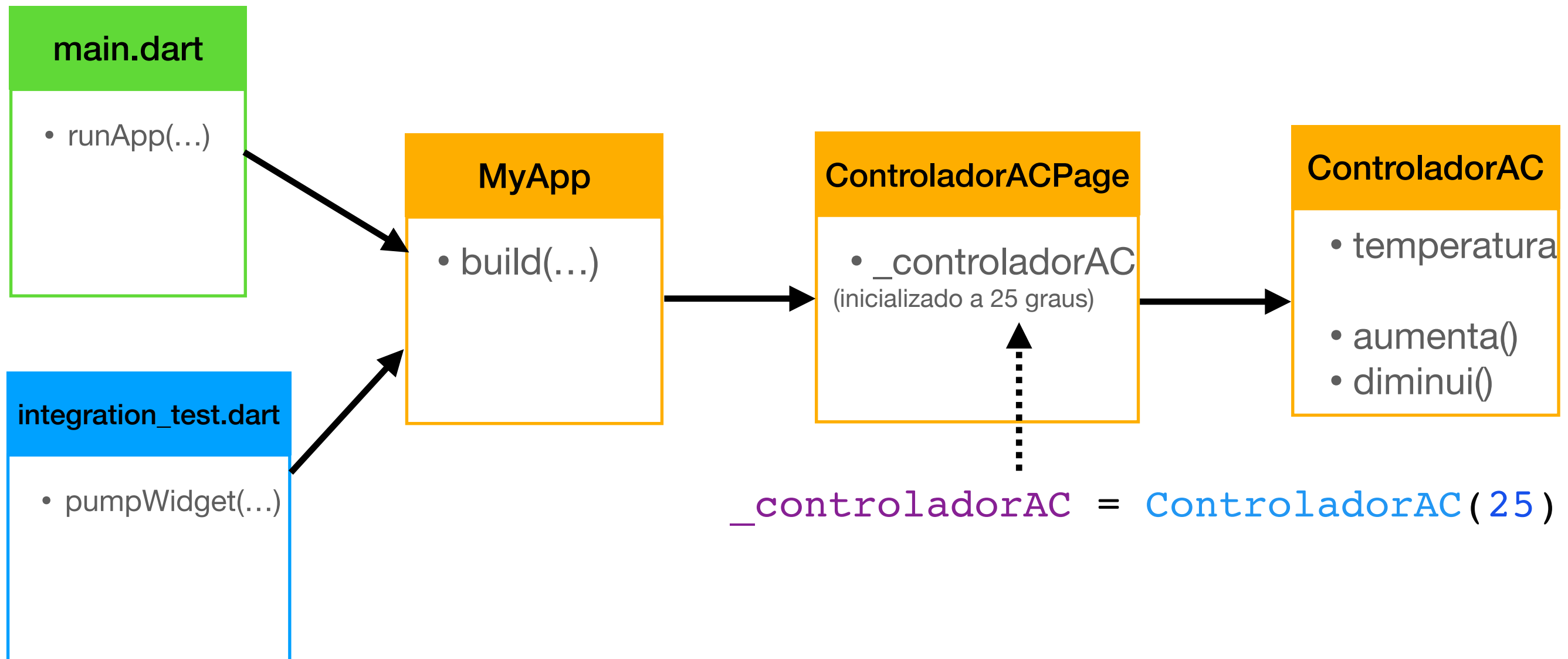
Mas esta solução acarreta potenciais problemas

```
class Widget1 {  
    Model _model = Model(...);  
    ...  
}
```

Controlador AC

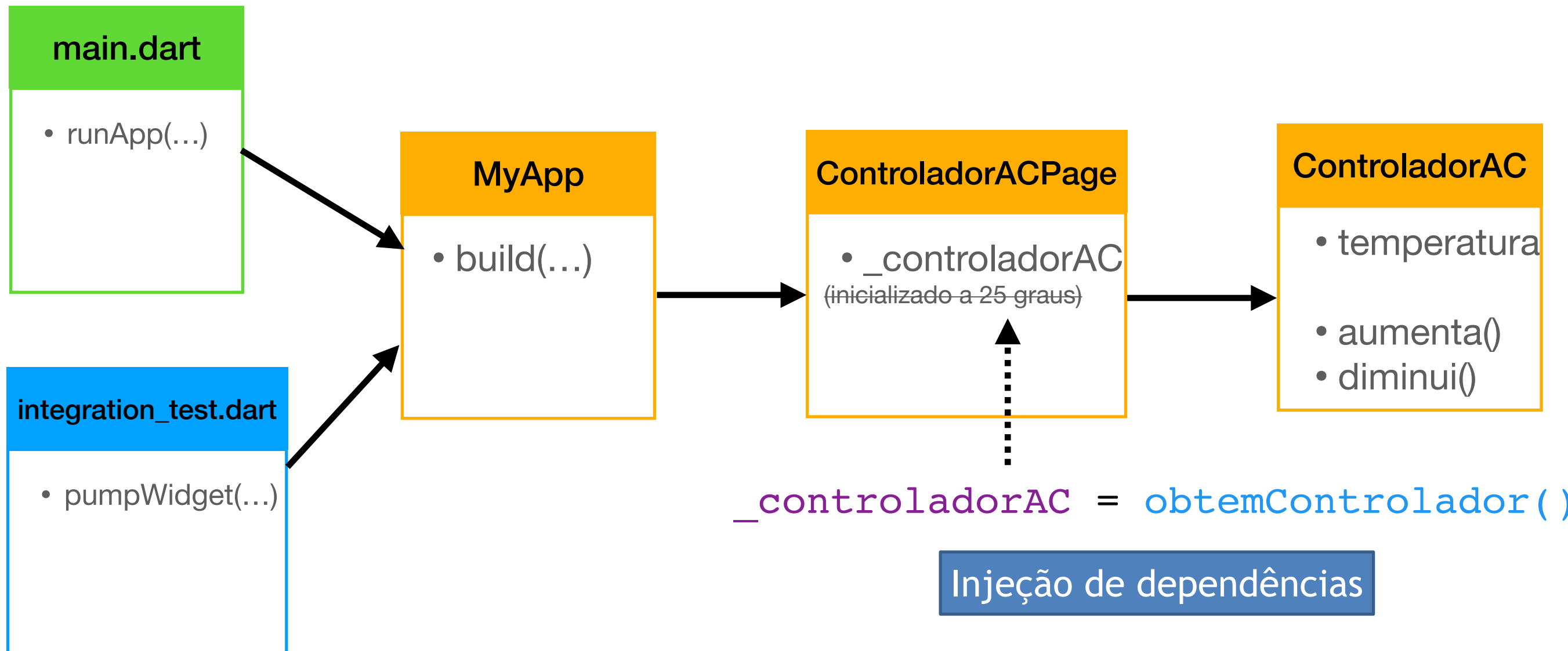


Controlador AC



Os meus testes vão assumir que o Controlador é inicializado a 25° (e se mudar?)

Controlador AC



Controlador AC

```
_controladorAC = ControladorAC(20)
```

main.dart

- runApp(...)

integration_test.dart

- pumpWidget(...)

MyApp

- build(...)

ControladorACPage

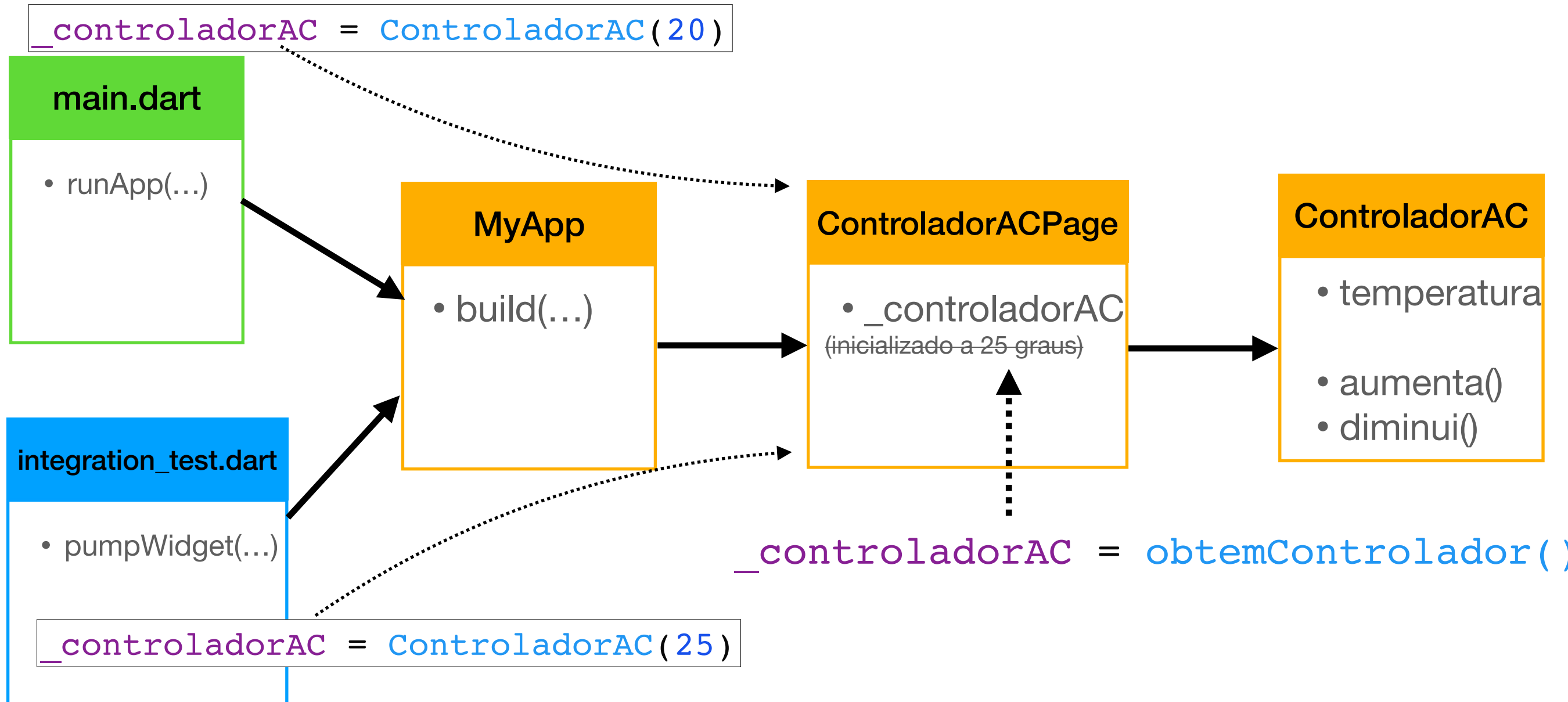
- _controladorAC
(inicializado a 25 graus)

ControladorAC

- temperatura
- aumenta()
- diminui()

```
_controladorAC = obterControlador()
```

Controlador AC



Injeção de dependências

objetoA depende de **objetoB** para fazer alguma coisa
(ex: ControladorACPage depende do ControladorAC)

Tradicional:

objetoA instancia **objetoB**



Injeção de dependências:

objetoA recebe **objetoB**



Injeção de dependências

Com injeção de dependências, deixamos de precisar de singletons

```
// representa o utilizador autenticado
class UtilizadorSessao {

    String? _nome;

    UtilizadorSessao.__( );

    String? get nome => _nome;
    set nome(String? value) => _nome = value;

    // única instância
    static final UtilizadorSessao _instance = UtilizadorSessao.__( );

    // Getter para aceder à instância
    static UtilizadorSessao get instance => _instance;
}
```

Injeção de dependências

Basta uma classe simples

```
// representa o utilizador autenticado  
class UtilizadorSessao {  
  
    String? _nome;  
  
    UtilizadorSessao();  
  
    String? get nome => _nome;  
    set nome(String? value) => _nome = value;  
  
}
```

Instanciá-la no main e injetá-la

```
void main() {  
    UtilizadorSessao utilizadorSessao = UtilizadorSessao();  
    runApp(MultiProvider(  
        providers: [ Provider.value(value: utilizadorSessao) ],  
        child: MyApp(),  
    ));  
}
```

Material de apoio

Foram publicados vídeos sobre isto:

- Desenvolvidimentos com widgets em flutter (parte 5)
Testes de integração + injeção de dependências
- Desenvolvidimentos com widgets em flutter (parte 6)
Padrão Observer/Observable
- Navegação em flutter (parte 2)
Melhorar a navegação com Observer/Observable

Nota: O projeto vai precisar disto

Exercícios

“If you can’t explain it simply, you don’t understand it well enough.”

— *Einstein (*)*

Exercício 1

Explica a diferença entre testes unitários, testes de widget e testes de integração de forma que uma criança de 5 anos entenda

(Dica: Usa analogias. Ex: “Imagina que tens várias gomas de diferentes cores...”)

Coloca a resposta no slido

Exercício 1 - possível resposta



Imagina que queres montar uma casa de legos.

Os **testes unitários** são para testar cada peça do lego separadamente (a porta abre e fecha?)

Os **testes de widget** são para testar divisões da casa (a sala tem um sofá e uma televisão?)

Os **testes de integração** são para testar a casa toda (dá para entrar pela porta, ir até ao sofá e sentar-se nele a ver televisão?)

Exercício 2

Explica o conceito de injeção de dependências de forma que uma criança de 5 anos entenda

Coloca a resposta no slido

Exercício 2 - possível resposta



Imagina que tens um carro telecomandado que precisa de pilhas.

És tu que colocas as pilhas no carro, não é ele que fabrica as pilhas.

Porque é que isso é bom? Porque podes facilmente trocar as pilhas, usar diferentes tipos de pilha, etc..