
concrete
PART OF ACCENTURE

Daivid Vasconcelos Leal
Mestrando na UFRPE
Sr. Android Developer na Concrete Solutions.

@daivid.v.leal
+55 81 995252247

Robot Pattern

Escrevendo testes legíveis para minha aplicação Android

Índice

Antes de tudo...

Pra que criar Testes?

- Testar **cenários** que geralmente não vemos;
- Garantir que mudanças na lógica da aplicação **não afete a Interface**;
- **Documentação** do código;
- **Incrementos** não afetem o código;

Testes Unitários

Testes Unitários

- Roda na JVM
- Utiliza o **máximo possível** de dependências mockadas, para isolar a funcionalidade
- **Muito Rápido**
- São criados na pasta **test**

Exemplo de testes unitários

Testes Unitários

```
class PasswordValidatorTest {

    @Test
    fun givenPasswordShorterThanEight_whenValidate_shouldReturnFalse(){
        //arrange
        //act
        val passwordValidator = PasswordValidator()
        var result = passwordValidator.validate("@aB12")
        //assert
        assertFalse(result)
    }

    @Test
    fun givenPasswordWithoutUpperCase_whenValidate_shouldReturnFalse(){
        //arrange
        //act
        val passwordValidator = PasswordValidator()
        var result = passwordValidator.validate("ab1#oiu3fd")
        //assert
        assertFalse(result)
    }
}
```

Problemas

- Recriar o objeto várias vezes;
- @Before não resolve problemas de estados iniciais;
- Objetos complexos tornam o teste difícil de ler;
- Quando é necessário mockar mais de um objeto o método se torna extenso demais;
- É preciso saber o framework para escrever o teste;

Testes Instrumentados

Testes Instrumentados

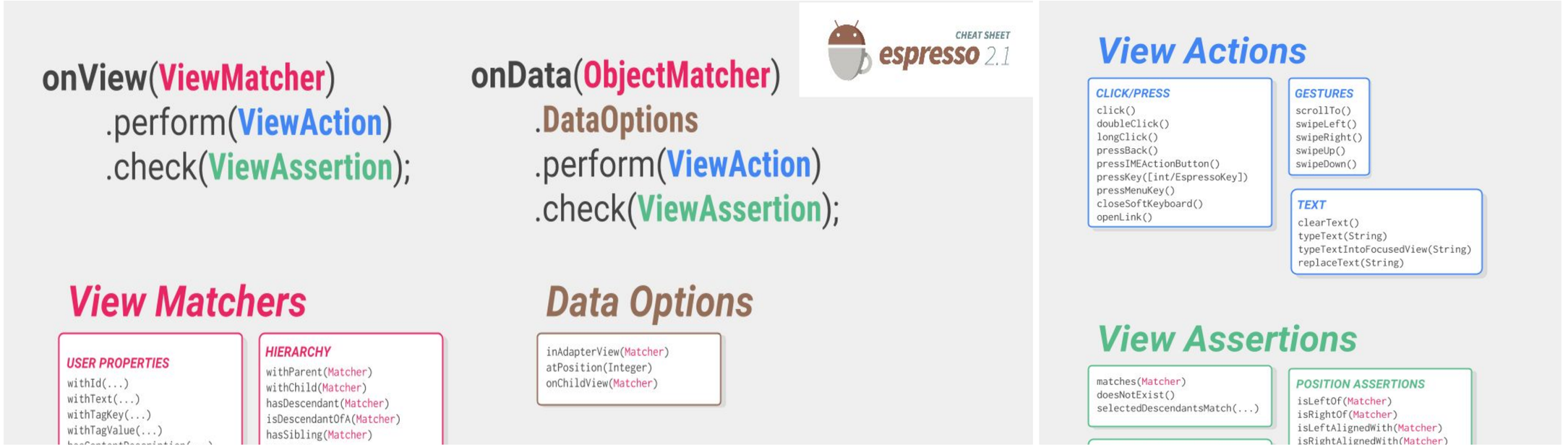
- Roda no **device** ou **emulador**
- Utiliza o **mínimo possível** de dependências mockadas, com interação próxima de um usuário do app
- **Mais lento** x **mais confiável**
- São criados na pasta **androidTest**

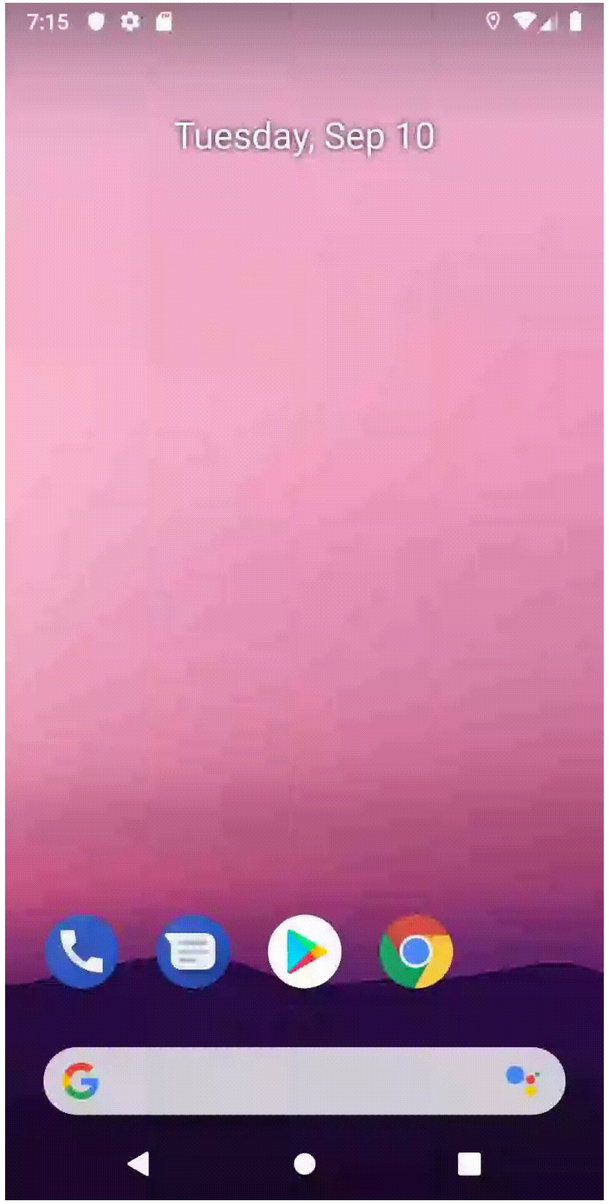
Espresso

Biblioteca criada para desenvolvedores garantir a qualidade do código através de testes instrumentados.

Espresso

Espresso foi criado para desenvolvedores, os quais acreditam que testes automatizados é uma parte importante do processo de desenvolvimento. Podendo ser utilizado para testes de caixa preta, essa poderosa biblioteca está disponível para aqueles que querem se aventurar e os que estão acostumados a criar testes instrumentados.





Um teste instrumentado que valida se o erro “Invalid password” é exibido

Exemplos de testes instrumentados

Sem Robot Pattern - Login com senha inválida

```
@RunWith(AndroidJUnit4::class)
class LoginActivityTest {

    @get:Rule
    val loginActivityTest = ActivityTestRule(LoginActivity::class.java)


    @Test
    fun givenPasswordIsInvalid_whenLogin_shouldShowPasswordIsInvalidError(){
      //arrange
      //act
      onView(withId(R.id.email)).perform(typeText("daivid.v.leal@concrete.com.br"))
      onView(withId(R.id.password)).perform(typeText("@!56Ab"))
      onView(withId(R.id.login)).perform(click())
      //assert
      onView(withText("Password is Invalid!")).check(matches(isDisplayed()))
    }
}
```

Sem Robot Pattern - Login com sucesso

```
@RunWith(AndroidJUnit4::class)
class LoginActivityTest {

    @get:Rule
    val loginActivityTest = IntentsTestRule(LoginActivity::class.java)

    @Test
    fun givenValidEmailAndPassword_whenLogin_shouldGoToHomeActivity(){
        //arrange
        intending(hasComponent(HomeActivity::class.java.name))
            .respondWith(Instrumentation.ActivityResult(Activity.RESULT_CANCELED, null))
        //act
        onView(withId(R.id.email)).perform(typeText("daivid.v.leal@concrete.com.br"))
        onView(withId(R.id.password)).perform(typeText("@!56Ab654"))
        onView(withId(R.id.login)).perform(click())
        //assert
        intended(hasComponent(HomeActivity::class.java.name))
    }

}
```

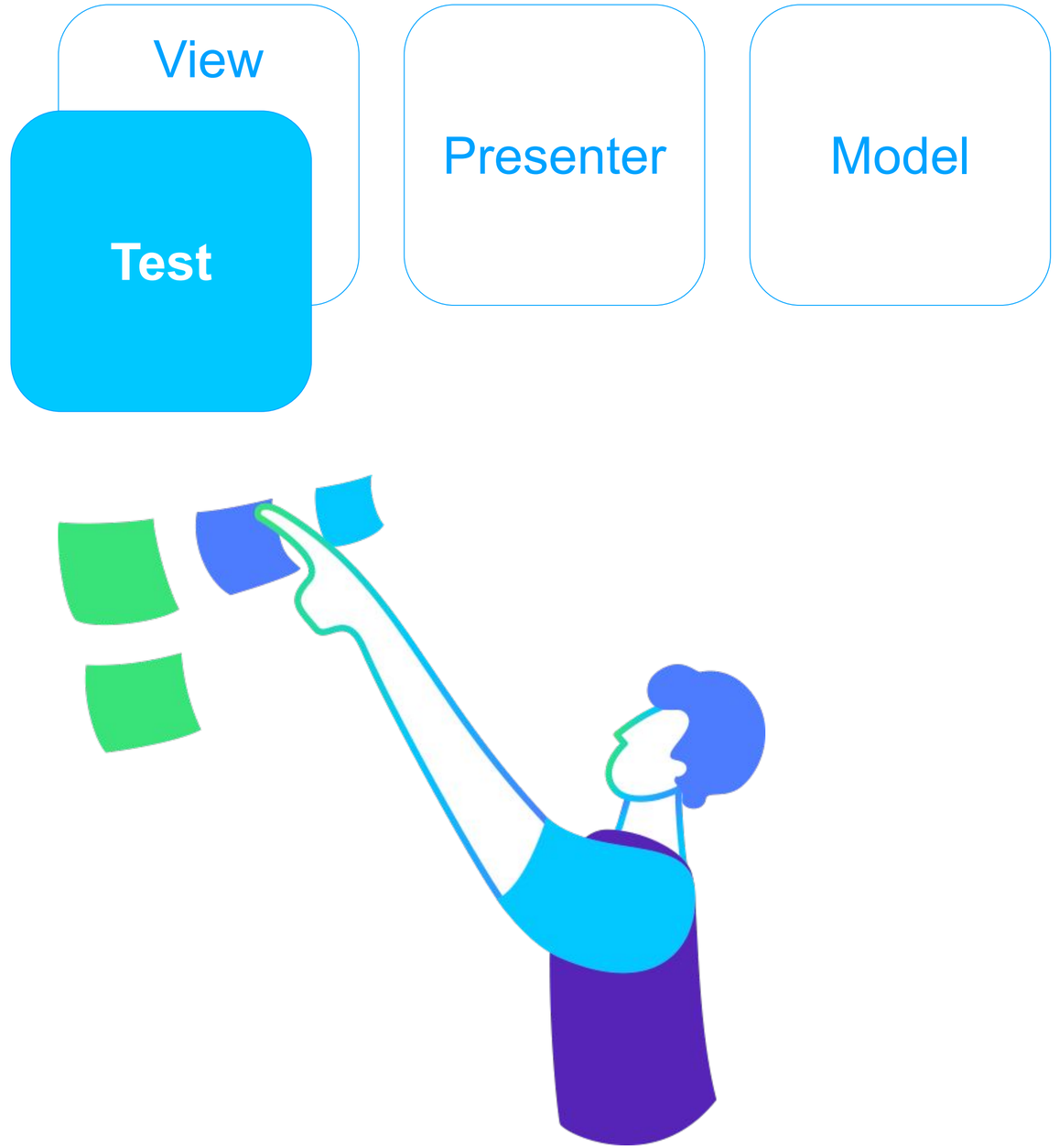
Problemas

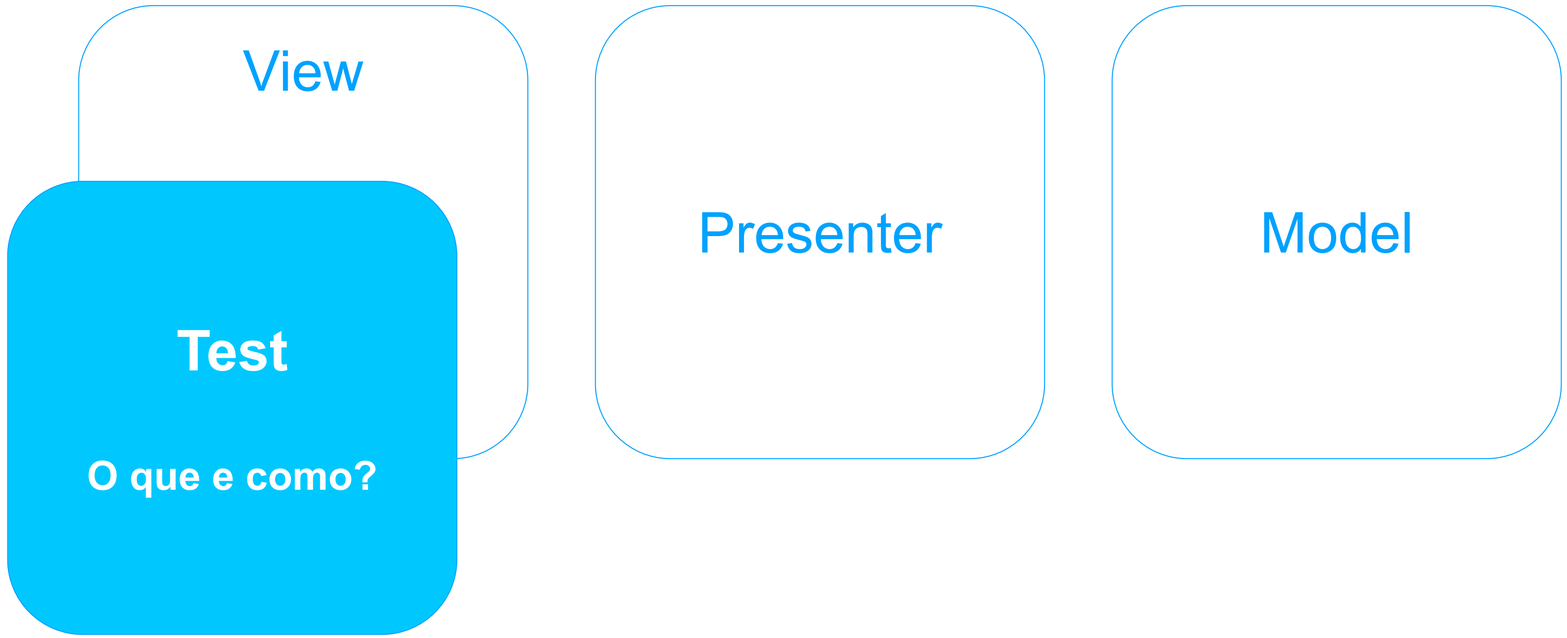
- Por ter uma API verbosa, o Espresso, dificulta o entendimento **do quê** está sendo feito no teste;
- **Repetição de código**
  - Quantas vezes digitamos `onView(withId(R.id.email)`?;
- É necessário saber da API do Espresso para escrever um cenário novo;
- Ler o corpo do teste exige um **esforço para interpretar** os detalhes de implementação;

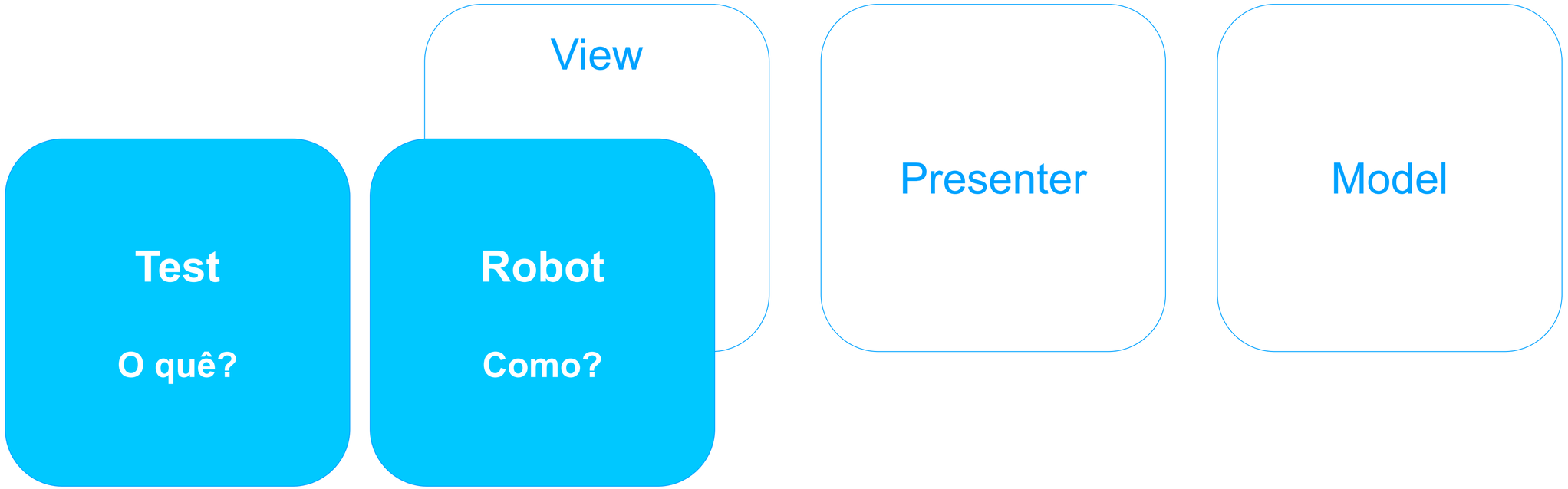
O que é Robot Pattern?

Chega de introdução e vamos ao que nos interessa...

- O Robot Pattern é um padrão criado por Jake Wharton para que você possa separar o “O que?” do “Como?”.
- Possibilita a minimização de Boilerplate.
- Torna os teste mais legíveis.




View
Test
O que e como?
Presenter
Model


View
Test
O quê?
Robot
Como?
Presenter
Model

Robot File

Extension Function

```
fun Test.mock(func: Arrange.() -> Unit)
    : Act {
      Arrange().apply {
            func()
      }
      return RobotAct()
}
```

Classe para Mocks

```
class RobotArrange{
    fun mockOne( … ){ ... }
    fun mockTwo( … ){ ... }
}
```

Classe para Ações

```
class RobotAct {
  infix fun act(func: Act.() -> Unit): Assert{
      this.apply{
          func()
      }
     return Assert()
  }
  fun actionOne( … ) { ... }
  fun actionTwo( … ) { ... }
}
```

Classe para Asserções

```
class RobotAssert {
  infix fun assert(func: Assert.() -> Unit){
      this.apply{
          func()
      }
  }
  fun assertationOne( … ) { ... }
  fun assertationTwo( … ) { ... }
}
```

Na Classe de Test

```
mock {
    ...
} act {
    ...
} assert {
    ...
}
```


Let’s Code

Test

O quê?

Robot

Como?

View

Presenter

Model

Comparando as duas escritas

```
@Test
fun
givenValidEmailAndPassword_whenLogin_shouldGoToHome
Activity(){
    //arrange
    intending(hasComponent(HomeActivity::class.java.name)).
    respondWith(Instrumentation.ActivityResult(Activity.RESU
    LT_CANCELED, null))
    //act
    onView(withId(R.id.email)).perform(typeText("daivid.v.leal
    @concrete.com.br"))

    onView(withId(R.id.password)).perform(typeText("@!56Ab
    654"))
    onView(withId(R.id.login)).perform(click())

    //assert
    intended(hasComponent(HomeActivity::class.java.name))
}
```

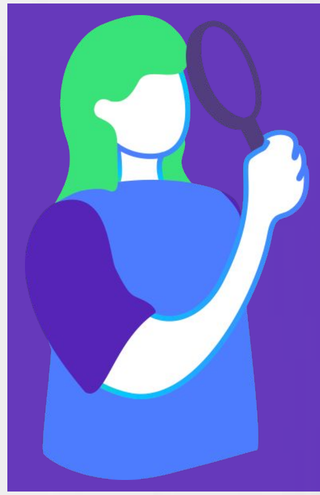
```
@Test
fun
givenValidEmailAndPassword_whenLogin_shouldGoToHome
Activity(){

    mockHomeActivity {
        mockGoToHomeActivity()
    } act {
        typeText("daivid.v.leal@concrete.com.br", R.id.email)
        typeText("@!56Ab654", R.id.password)
        click(R.id.login)
    } assert {
        checkGoTo(HomeActivity::class.java.name)
    }

}
```

Ganhos utilizando Robot Pattern

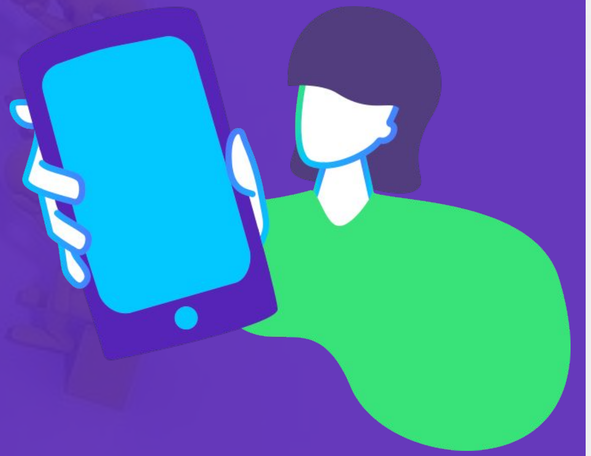
- Reutilização de código;
- Resiliência a mudanças;
- Possibilidade de escrever códigos com mais clareza;
- Não é necessidade saber android ou uma lib de testes para escrever novos testes;

Atenção ao utilizar Robot Pattern

- Decisão do time;
- Não garante a estrutura dos testes;
- Não há necessidade quando temos testes muito simples;



Dúvidas?



Quer fazer parte do nosso time de times?

Temos vagas!

concrete
PART OF ACCENTURE

NÓS MOVEMOS O MUNDO.

RIO

Centro
Av. Presidente Wilson, 231
29º andar
(21) 2240-2030

SÃO PAULO

Cidade Monções
Av. Nações Unidas, 11.541
3º andar
(11) 4119-0449

BH

Savassi
Av. Getúlio Vargas, 671
Sala 800 - 8º andar
(31) 3360-8900

RECIFE

Ilha do Leite
Rua Sen. José Henrique, 199
2º andar
(81) 3018-6299

WWW.CONCRETE.COM.BR