Mock Objects

Aula 04 — Testar sem o banco, sem o WhatsApp e sem o calendário

Aula 04 · 10/11/2026 · Indicador 1

UniSenac Centro Universitário RS
Senac
Fecomércio
IFEP
Sesc
Sindicatos Empresariais

O mapa de hoje

- **Bloco 1** — uma regra simples que nenhum teste consegue alcançar
- **Bloco 2** — o dublê: `Mock`, injeção pelo construtor e a falha simulada
- **Bloco 3** — o teste que só passa no fim de semana, e o que **não** mockar
- **Bloco 4** — mão na massa: os testes do cobrador e uma regra do grupo

> Baixem o `cobranca.py` da página da aula e abram a pasta no VS Code, com o painel **Testing** à vista. Hoje vocês vão mudar esse arquivo.

Retomando a Aula 03

- Contem o ciclo do TDD com as palavras de vocês: o que acontece no vermelho, no verde e no refatora?
- Por que vale a pena **ver o teste falhar** antes de escrever o código?
- Em que situação vocês **não** usariam TDD?

> Guardem uma pergunta para o fim do dia: quando um teste é difícil de escrever, a culpa é do teste ou do código?

Bloco 1

Uma regra simples, impossível de testar

Toda manhã, o sistema cobra os atrasados

```python
class CobradorDeAtrasos:
    def cobrar(self):
        repositorio = RepositorioDeEmprestimos()  # conecta no PostgreSQL
        avisador = AvisadorPorWhatsApp()          # manda mensagem ao cliente
        hoje = date.today()                       # pergunta a data ao computador
        if hoje.weekday() >= 5:                   # sábado ou domingo: não cobra
            return
        for emprestimo in repositorio.abertos():
            if emprestimo.atrasado_em(hoje):
                emprestimo.registrar_aviso(hoje)
                repositorio.atualiza(emprestimo)
                avisador.avisa(emprestimo)
                self.total_avisados += 1
```

Está no `cobranca.py`. Em dupla: **o que esta regra promete fazer?**

Que cenários vocês testariam?

Pensem em três antes de eu mostrar o começo do teste.

```python
def test_avisa_quem_passou_do_prazo():
    atrasado = Emprestimo(1, "Ana Souza", "51 99999-0001", date(2026, 11, 1))

    cobrador = CobradorDeAtrasos()
    cobrador.cobrar()

    assert cobrador.total_avisados == 1
```

O cenário está pronto na primeira linha — e não chega ao cobrador. Ele busca os empréstimos **no banco**, sozinho. **Como entregar a Ana a ele?**

Tentativa 1: pôr o cenário no banco de verdade

O teste faz o `INSERT` do empréstimo da Ana e depois chama `cobrar()`.

- **Palpite antes de rodar:** passa? E se rodar **de novo**, logo em seguida?

rodada	o que o teste encontra no banco	resultado
1ª	só o empréstimo que ele inseriu	verde
2ª	o da 1ª rodada, que **ficou lá** — o mesmo `INSERT` é recusado	vermelho, na `UNIQUE`

- E o WhatsApp? A Ana de verdade recebeu **duas** cobranças de um teste.

Tentativa 2: um repositório falso, escrito à mão

```python
class RepositorioFalso:
    def __init__(self, emprestimos):
        self.emprestimos = emprestimos
        self.atualizados = []

    def abertos(self):
        return [e for e in self.emprestimos if e.data_entregue is None]

    def atualiza(self, emprestimo):
        self.atualizados.append(emprestimo)
```

Nada de banco, nada de lixo entre uma rodada e outra. **Funciona.** Onde está o problema?

O falso cresce a cada pergunta nova

quero testar…	preciso escrever…
quem passou do prazo é avisado	`RepositorioFalso` e `AvisadorFalso`
o aviso saiu **uma** vez só	um contador dentro do `AvisadorFalso`
o WhatsApp caiu no meio da cobrança	um `AvisadorQueFalha`, só para isso
o banco caiu no `atualiza`	um `RepositorioQueFalha`, só para isso
o repositório ganhou um método novo	o mesmo método em **todos** os falsos

Quanto trabalho para testar uma regra de dez linhas. A ideia do falso está certa; o que falta é um jeito de não escrevê-lo à mão.

Checagem do Bloco 1

No `cobrar()` do `cobranca.py` que vocês receberam:

- Marquem **cada linha que sai do Python** — que fala com algo que o teste não controla
- Para cada uma, escrevam **o que dá errado** se o teste rodar duas vezes seguidas, num sábado, no laboratório sem internet
- Que linha do `RepositorioFalso` vocês teriam de **mudar** para testar "o banco caiu"?



`forms.gle/D2fc8bxz915QFtJq5`

Intervalo

10 minutos

Bloco 2

O dublê, e como ele entra na classe

Um falso em uma linha: `Mock`

```python
from unittest.mock import Mock

repositorio = Mock()
repositorio.abertos.return_value = [atrasado]

repositorio.abertos()      # devolve [atrasado]
repositorio.atualiza(x)    # aceita, não faz nada, e anota que foi chamado
```

GLOSSÁRIO

`mock`	dublê — objeto que finge ser outro
`unittest.mock`	módulo de dublês, **já vem com o Python**: nada a instalar
`return_value`	valor de retorno — o que o método falso devolve

Palpite: com o dublê, o teste passa?

```python
def test_avisa_quem_passou_do_prazo():
    atrasado = Emprestimo(1, "Ana Souza", "51 99999-0001", date(2026, 11, 1))
    repositorio = Mock()
    repositorio.abertos.return_value = [atrasado]

    cobrador = CobradorDeAtrasos()
    cobrador.cobrar()

    assert cobrador.total_avisados == 1
```

- Escrevam o palpite. Agora rodem pelo painel **Testing**.
- Leiam a última linha vermelha: **quem** tentou conectar no PostgreSQL?

O dublê entra pela porta da frente

Antes — a classe fabrica o que usa

```
def cobrar(self):
    repositorio = RepositorioDeEmprestimos()
    avisador = AvisadorPorWhatsApp()
```

Depois — quem cria, entrega

```
def __init__(self, repositorio, avisador):
    self.repositorio = repositorio
    self.avisador = avisador
```

No teste: `CobradorDeAtrasos(Mock(), Mock())`. No sistema de verdade: `CobradorDeAtrasos(RepositorioDeEmprestimos(), AvisadorPorWhatsApp())`.

GLOSSÁRIO

dependency injection	injeção de dependência — a classe recebe o que usa, em vez de criar

Verde. Mas o aviso saiu?

`total_avisados == 1` só prova que o contador subiu. O cliente foi avisado? O banco foi atualizado?

```
avisador.avisa.assert_called_once_with(atrasado)
repositorio.atualiza.assert_called_once_with(atrasado)
```

GLOSSÁRIO

`assert_called_once_with`	confirme que foi chamado **uma vez**, com **este** argumento
`stub`	dublê que só responde — o `return_value` de até agora
`mock`	dublê que também **confere** as chamadas — o de agora

- Apaguem a linha `avisador.avisa(...)` do `cobrar()` e rodem. O que a mensagem diz?

Contar e negar

```python
def test_nao_avisa_quem_ainda_esta_no_prazo():
    ...
    avisador.avisa.assert_not_called()

def test_avisa_cada_atrasado_uma_vez():
    ...  # três empréstimos abertos, dois atrasados
    assert avisador.avisa.call_count == 2
```

GLOSSÁRIO

`assert_not_called`	confirme que **não** foi chamado
`call_count`	contagem de chamadas — quantas vezes o método foi usado

- O empréstimo que vence **hoje** já está atrasado? Escrevam o teste antes de responder: é a fronteira da Aula 02.

⚠ Erro comum — o dublê que aceita qualquer coisa

Alguém renomeia `avisa()` para `enviar()` na classe `AvisadorPorWhatsApp`. O cobrador continua chamando `avisa()`.

- **Palpite:** os testes ficam vermelhos?

> Com `Mock()`, os testes ficam todos **verdes** — o dublê aceita qualquer nome de método. A produção quebra na primeira manhã. Com `Mock(spec=AvisadorPorWhatsApp)`, o dublê só aceita o que a classe de verdade tem, e o teste fica vermelho na hora.

GLOSSÁRIO

`spec`	especificação — o molde que o dublê é obrigado a seguir

O WhatsApp caiu no primeiro cliente

```
avisador.avisa.side_effect = [ConnectionError("WhatsApp fora do ar"), None]
repositorio.abertos.return_value = [primeiro, segundo]
```

A primeira chamada estoura; a segunda funciona.

- **Palpite:** o `segundo` é avisado?
- Rodem. Agora imaginem a locadora com 40 atrasados e o WhatsApp instável numa segunda-feira.

GLOSSÁRIO

`side_effect`	efeito colateral — erro a lançar, ou uma resposta diferente por chamada
`ConnectionError`	erro de conexão

O conserto: a falha é de um cliente, não da cobrança

```
for emprestimo in self.repositorio.abertos():
    if not emprestimo.atrasado_em(hoje):
        continue
    try:
        self.avisador.avisa(emprestimo)
    except Exception:
        self.falhas.append(emprestimo)   # o balcão liga para este depois
        continue
    emprestimo.registrar_aviso(hoje)
    self.repositorio.atualiza(emprestimo)
    self.total_avisados += 1
```

Comparem com o slide 5: além do `try`, **duas linhas mudaram de lugar**. Quais, e por quê?

Checagem do Bloco 2

Com o WhatsApp caindo no `primeiro` e funcionando no `segundo`:

- Escrevam a linha que confere que **só o** `segundo` foi gravado no banco
- Escrevam a linha que confere que o `primeiro` ficou em `cobrador.falhas`
- Na versão original do `cobranca.py`, esse teste passaria? O que o banco diria da Ana?
- Escrevam o `side_effect` para "o banco cai no `atualiza` do primeiro"



`forms.gle/f2LLRNJiDgfgx6mw8`

Intervalo

10 minutos

Bloco 3

O calendário, e o que não se mocka

Sábado de manhã, em casa: quatro vermelhos

Alguém do grupo roda no sábado, 14/11, os testes que ficaram verdes hoje. **Nada no código mudou.**

```
FAILED test_avisa_quem_passou_do_prazo
FAILED test_avisa_cada_atrasado_uma_vez
FAILED test_whatsapp_fora_do_ar_nao_para_a_cobranca
FAILED test_nao_registra_aviso_que_nao_saiu
```

- Procurem no `cobranca.py` a linha que explica isso. E os dois que ficaram verdes: verdes **por quê**?
- Agora escrevam `test_nao_cobra_no_fim_de_semana`. **Palpite:** dá para deixá-lo verde hoje, terça?

Nenhum dia deixa os dois verdes

hoje é...	`avisa_quem_passou_do_prazo`	`nao_cobra_no_fim_de_semana`
segunda a sexta	verde	**vermelho**
sábado ou domingo	**vermelho**	verde

O teste não mede o código: mede o dia em que alguém apertou o play. Teste que muda de resultado sem o código mudar é pior do que teste nenhum — a turma para de acreditar no vermelho.

A data também é uma dependência

```
class RelogioDoSistema:
    def hoje(self):
        return date.today()
```

O cobrador recebe o relógio no construtor e troca `date.today()` por `self.relogio.hoje()`.

```
relogio = Mock(spec=RelogioDoSistema)
relogio.hoje.return_value = date(2012, 4, 7)  # um sábado

CobradorDeAtrasos(repositorio, avisador, relogio).cobrar()
avisador.avisa.assert_not_called()
```

Verde na terça, no sábado e daqui a dez anos.

O que mockar, e o que não

dependência	mockar?	por quê
repositório (PostgreSQL)	**sim**	deixa sujeira, precisa do servidor ligado
avisador (WhatsApp, e-mail)	**sim**	a mensagem sai de verdade e não volta
relógio	**sim**	muda sozinho, todo dia
`Emprestimo`	**não**	é a regra que está sendo testada
conta simples (dias de atraso)	**não**	escrever o dublê dá mais trabalho que usar a conta

Mocka-se o que é **difícil de controlar**, não tudo o que aparece no código.

⚠ Erro comum — mockar a entidade

```
emprestimo = Mock(spec=Emprestimo)
emprestimo.atrasado_em.return_value = True
```

Alguém troca `hoje > self.data_prevista` por `hoje >= self.data_prevista` no `Emprestimo`: agora quem devolve **no dia do prazo** é cobrado.

- **Palpite:** este teste pega o defeito?

> Não pega: o dublê responde `True` de qualquer jeito, e o teste continua verde. Com o `Emprestimo` de verdade, o teste da fronteira fica vermelho na hora. Quanto mais se mocka, menos o teste testa.

⚠ Erro comum — testar o caminho em vez do resultado

```
relogio.hoje.assert_called_once()
repositorio.abertos.assert_called_once()
```

Alguém refatora o `cobrar()` e passa a perguntar a data duas vezes. **A cobrança continua idêntica** — e esses testes ficam vermelhos.

> Confiram o que o mundo vê: **quem foi avisado** e **o que foi gravado**. Como a classe chegou lá é problema dela. Teste que conhece o caminho quebra a cada refatoração, e o refatora é um terço do ciclo da Aula 03.

Checagem do Bloco 3

Uma regra da locadora: "no aniversário do cliente, o aluguel sai com 20% de desconto, e ele recebe um e-mail de parabéns".

- Listem as dependências: **quais vocês mockariam e quais não**, e por quê
- Escrevam o teste do aniversário com o relógio dublê — escolham a data
- Este teste confere resultado ou caminho? `relogio.hoje.assert_called_once()`



forms.gle/BnbxP66x53NPP3EW9

Intervalo

10 minutos

MÃO NA MASSA

O cobrador, e uma regra de vocês

Parte A — no `cobranca.py` da locadora

1. Passem repositório, avisador e relógio pelo construtor
2. Testem: atrasado avisado, no prazo não, dia do prazo, fim de semana
3. Testem o WhatsApp caído, e consertem o `cobrar()` até ficar verde

Parte B — no minimundo do grupo

1. Escolham uma regra que dependa de algo de fora: banco, data, e-mail
2. A regra recebe essa dependência pelo construtor; escrevam dois testes com dublê

MÃO NA MASSA

O combinado desta entrega

GLOSSÁRIO

`test_grupoNN_aulaNN.py` `test_grupo03_aula04.py`

Um arquivo só: o `cobranca.py` alterado e, embaixo, os testes das Partes A e B.

O que eu confiro: Ninguém cria a própria dependência dentro do método · Todo dublê de classe do projeto usa `spec` · O `Emprestimo` e as entidades de vocês **não** são dublês · Há pelo menos um `side_effect` e um `assert_not_called` · Nenhum teste depende do dia em que roda · Os testes conferem resultado, não caminho

Devolutiva — passo de mesa em mesa

Eu passo em cada grupo. Tenham prontos na tela:

- o painel **Testing** todo verde — e um teste que vocês viram ficar **vermelho** antes
- a regra da Parte B, e o que nela vocês **decidiram não mockar**
- a linha do código de vocês que dependia do relógio, do banco ou da internet

> Não há apresentação para a turma. Quem quiser mostrar o que achou de curioso mostra; ninguém é obrigado.

PARA CASA · OPCIONAL

Antes da próxima aula

Testar em casa: rodem os testes de vocês num sábado ou domingo. Algum ficou vermelho? Então ainda tem relógio escondido.

Tarefa opcional: procurem no CRUD do grupo, feito na UC2, uma função que não dá para testar sem o PostgreSQL ligado. Anotem qual é e por quê.

Para conferir: o `cobranca.py` está na página da aula. Os testes de vocês são o exercício.

Para ver acontecer: demonstrações da Aula 04 — montar o laço do cobrador com o WhatsApp caindo, e procurar o dia da semana em que os dois testes ficam verdes — e oito perguntas com resposta na hora.

Para a próxima aula

Mockamos o banco. E se
o erro estiver no SQL?

UniSenac Centro Universitário RS
Senac
Fecomércio
IFEP
Sesc
Sindicatos Empresariais