10

Promise mais elegante

Wittgenstein implementou as classes `ConnectionFactory` e `NegociacaoDao` como ensinado neste capítulo. Inclusive, ele teve certeza que seu código funcionava a partir do seguinte teste:

```
ConnectionFactory
    .getConnection()
    .then(conexao => {

        let dao = new NegociacaoDao(conexao);
        dao.adiciona(new Negociacao(new Date(), 1, 100))
            .then(() => {
                alert('Negociação adicionada com sucesso');
            });

    })
    .catch(erro => console.log(erro));
```

Apesar de funcional, veja que se tivéssemos mais chamadas à `then`, cairíamos em algo parecido com o *Callback Hell*, assunto que tocamos no primeiro módulo de uma sequência de treinamentos avançados.

Qual das opções abaixo reescreve elegantemente o código de Wittgenstein, evitando assim o aninhamento de chamadas à `then`?

Selecione uma alternativa

A

```
ConnectionFactory
    .getConnection()
    .then(conexao => new NegociacaoDao(conexao))
    .then(dao => dao(new Negociacao(new Date(), 1, 100)))
    .then(() => alert('Negociação adicionada com sucesso'))
    .catch(erro => console.log(erro));
```

B

```
ConnectionFactory
    .getConnection()
    .then(new NegociacaoDao())
    .then(dao => dao.adiciona(new Negociacao(new Date(), 1, 100)))
    .then(() => alert('Negociação adicionada com sucesso'))
    .catch(erro => console.log(erro));
```

C

```
ConnectionFactory
    .getConnection()
    .then(conexao => new NegociacaoDao(conexao))
    .then(dao => dao.adiciona(new Negociacao(new Date(), 1, 100)))
    .then(() => alert('Negociação adicionada com sucesso'))
    .catch(erro => console.log(erro));
```