Atividade Prática - Locks

A rede de mercados Pepperoni tem algumas lojas próximas de Universidades e é frequentada por muitos estudantes. Possuindo orçamento limitado, os estudantes precisam verificar se possuem dinheiro suficiente para comprar os itens escolhidos. Infelizmente, a inflação está tão alta que às vezes os preços mudam enquanto os estudantes compram. Isso causa constrangimento e filas nos caixas, além de gerar publicidade negativa para a rede Pepperoni.

No código dado, a classe `Market` possui as seguintes operações:

```
public class Market {
    // Atribui um preço a um produto específico
    public void setPrice(@Nonnull Product product, double value);

    // Pega um produto da gôndola e coloca na cesta.
    // O retorno é o valor do produto
    public double take(@Nonnull Product product);

    // Tira um produto da cesta e coloca de volta na gôndola
    public void putBack(@Nonnull Product product);

    // Espera até que o preço do produto baixe para um valor
    // menor que maximumValue. Quando isso acontecer, coloca
    // o produto na cesta. O método retorna o valor do produto
    // colocado na cesta
    public double waitForOffer(@Nonnull Product product,
                               double maximumValue) throws InterruptedException;

    // Paga por um produto. O retorno é o valor pago, que deve
    // ser o mesmo retornado por waitForOffer() ou take()
    public double pay(@Nonnull Product product);
}
```

Garanta as seguintes regras:

- Vários clientes podem possuir o mesmo produto na cesta ao mesmo tempo.
- Um cliente pode possuir o mesmo produto na cesta múltiplas vezes (vários itens).
- Se algum cliente possui um produto "x" na cesta, uma chamada a `setPrice()`deve bloquear até que nenhum cliente tenha o produto "x" na cesta.
- Se um cliente colocou na cesta um produto "x" que possuía um preço "y", ele deve pagar o mesmo preço ao sair do mercado (ou seja, `pay()` deve retornar "y" mesmo que outra thread tenha chamado `setPrice`() com valor diferente de preço após o cliente colocar o produto "x" na cesta).
- Um cliente pode pagar por um produto em sua cesta ou devolvê-lo à gôndola.
- Permita concorrência de quaisquer operações que não violem as regras acima. Exemplo: alterar o preço do café enquanto um cliente tem coca-cola na cesta.

Dicas:

A interface `ReadWriteLock` junta dois `Locks`:

- o `readLock()` pode ser compartilhado por várias threads simultaneamente, e simboliza que todas elas tem permissão para ler valores compartilhados;
- o `writeLock()` funciona como um `Lock` normal e só pode ser obtido por uma única thread (exclusão mútua). Ele é mutuamente exclusivo inclusive com o `readLock()`, ou seja:
  - enquanto alguma thread possui o `readLock()`, nenhuma outra consegue adquirir o `writeLock()`, e ...
  - enquanto alguma thread possui o `writeLock()`, nenhuma outra consegue adquirir o `readLock()`.

A interface `Lock` permite criar uma instância da classe `Condition`através de `newCondition()`.
Uma `Condition` possui métodos `await()`, `signal()` e `signalAll()` que funcionam de forma **similar**ao monitor. A

diferença é que é **necessário obter o `Lock`** antes de usar a `Condition`.

No caso de `ReentrantReadWriteLock` (que implementa `ReadWriteLock`), apenas o `writeLock()` permite criar uma `Condition`.

O enunciado acima inclui cenários nos quais cada uma dessas classes e métodos precisam ser usados**.** Um método precisa usar a condition, outro precisa usar um read lock, outro um write lock ... **Relacione a especificação de cada método da classe `Market` com as características de cada mecanismo de controle de concorrência**.

Correção automática

O script de correção (**grade-locks.sh**) está dentro do esqueleto e deve ser executado na própria pasta onde está. Você deve implementar as classes já criadas. Não crie suas soluções em novas classes ou o script corretor não as irá encontrar. Os testes estão visíveis na classe `MarketTest`, para os curiosos. O arquivo de testes será substituído durante a avaliação.

Caso queira usar features do Java 8+ altere o pom.xml.

Entrega do Exercício

Submeta um arquivo **.tar.gz (ou zip)** na **mesma estrutura do esqueleto**dado como ponto inicial nessa tarefa. O uso do esqueleto fornecido é **obrigatório**. Utilize **make submission** para garantir que será gerado um arquivo conforme solicitado.

O prazo para entrega é **11 de Novembro às 23h55**.

Esqueleto

(Atualizado em 27/10/2018 - 13:08)

- atividade_11.tar.gz