
# Programação Concorrente

Semestre Inverno

LI52D

2017/2018

Engenheiro Pedro Félix

# Série de Exercícios 1

42208 Diogo Dias

Exercício 1

Foi pedido para implementar a classe *ExpirableLazy*<T> respeitando a interface pública dada. É necessário respeitar certas regras que estão explicadas no enunciado.

No construtor do *ExpirableLazy*, colocamos o campo privado que irá conter o valor a *null*, guardamos o tempo de vida do valor numa variável, e a função *provider*.

No método *get* da propriedade *Value*, quando uma *thread* entra, observa primeiro se o campo *value* não é *null* (alguém já pode ter calculado o seu valor) e se o tempo de duração ainda não expirou. Se essas duas condições forem verdadeiras a *thread* retorna simplesmente o *value*. Se elas não forem verdadeiras é necessário ir calcular um novo valor para o *value* com a função Provider. A *thread* então irá verificar se já existe alguma *thread* a executar o *provider*, se houver bloqueia-se, se não houver altera o *boolean calculating* para *true* e vai executar o método *Provider* fora da exclusão. No fim de calcular é necessário que a *thread* entre novamente na exclusão independentemente de ser interrompida. Isso é necessário acontecer porque é necessário acordar as outras *threads* caso o valor já tenha sido calculado, ou porque o método *Provider* deu exceção e é obrigatório acordar uma *thread* para que venha calcular o valor. Se o *Provider* lança exceção, é optado por acordar só uma *thread* de modo a diminuir o número de comutações. O cuidado a ter com esta técnica é que se uma *thread* sair por interrupção tem que obrigatoriamente notificar uma outra *thread*, isto porque, aquela *thread* pode ter sido notificada e de modo a não se perder a sinalização esta tem que propagar o sinal. Se a execução do *Provider* não der erros, este atualiza o valor do *value* e renova o tempo de vida dessa variável.

Exercício 2

Neste exercício, usei um sincronizador que transfere objetos de uma *thread* para outra.

No método *Put* a *thread* irá primeiro verificar, se existe alguma *thread* bloqueada à espera de receber o objeto, se houver, esta irá fornecer-lhe o objeto, sinalizando e retirando-lhe da lista de espera. No caso de não houver ninguém à espera, a *thread* coloca o objeto numa fila, e sai. Este método não é bloqueante, logo as *threads* não esperam pela confirmação de que alguém tenha recebido o objeto.

No método *Transfer*, é também feita uma primeira observação, se existe alguma *thread* bloqueada à espera de receber. Se houver fornece o objeto à *thread* que tenha chegado primeiro, respeitando uma lógica *FIFO (First In First Out).* Se não houver, esta verifica se tem *timeout*, se não tem, esta retorna false. Mas se tiver, a *thread* bloquear-se-á numa fila, com lógica FIFO, à espera que alguém retire o seu objeto. Durante este processo a *thread* poderá sofrer uma interrupção, mas como valorizamos a delegação de trabalho, se esta já deu o objeto, não irá sair com exceção, armando novamente assim a interrupção e retornando *true*. Caso ainda ninguém tenha obtido o seu objeto lança a exceção. Também poderá acontecer que o tempo de espera tenha acabado e ainda ninguém tenha obtido o seu objeto, assim a *thread* sai retornando false.

No método Take é também feita uma primeira observação, se existe alguma *thread* bloqueada que esteja a fornecer algum objeto. Se houver, esta tira-o e sinaliza-a. Todo isto respeitando a lógica FIFO. Se não houver, esta verifica se tem *timeout*, se não tem, retorna false. Se tem tempo de espera, a *thread* bloqueia-se à espera que venha uma *thread* para lhe poder dar o objeto. Nesta situação, poderão acontecer três caso, vem um *thread* pelo o método *transfer*, e dá-lhe o objeto e sai assim a *thread* com *true*; há uma interrupção e ainda não obteve o objeto e assim lança exceção; ou acabou o tempo e assim sai sem objeto retornando *false*.

Exercício 3

Neste exercício, é criado um sincronizador para emparelhar dois objetos de *threads* diferentes. Para isso é usado um monitor, para alterar o estado da fila sincronizadamente, e uma fila. A fila é composta por *booleans*, para saber o estado, qual o objeto que falta, e por um *tuplo*, que irá conter os dois objetos. Quando o *tuplo* obter os dois objetos, o mesmo é retornado às duas *threads*. Para minimizar o número de comutações entre *threads*, é usado notificação especifica, em que o monitor de cada *thread* irá ser o nó de cada fila. O nó representa a sua posição de espera, e também o objeto que ele forneceu.

No método *Provide,* em que recebe um objeto do tipo T, a *thread* entra com um*value*, e vai verificar primeiro se a lista está vazia, se não estiver vai ver se aquela lista é uma lista de *Ts* ou de Us. Para fazer essa verificação vai observar o primeiro nó, e com um *boolean* observa se o U está presente. Se estiver, coloca no *tuplo* o seu objeto, aciona a condição, e mete o *boolean tIsPresent* a *true*. A *thread* que estava à espera de um objeto T, acorda, e verifica que o T está presente, saindo assim com *true*. A *thread* que entra com um objeto do tipo T, se observar que a lista não é de tipos U, esta então vai-se bloquear no fim da lista. Nesta situação de bloqueio só poderá sair se for interrompida, retornando *true* se tem o *tuplo* composto ou lança a exceção, o *tuplo* já tem o outro valor que estava à espera ou foi por *timeout*, retornando false.

O método *Provide* que recebe por parâmetro o U, tem a mesma lógica, a única diferença é que este irá sempre verificar é se o objeto do tipo T está presente.

Quando uma *thread* sair por interrupção, ou por *timeout*, o objeto que essa *thread* colocou na fila será removido para que outra *thread* não possa fazer par com esse valor.

Exercício 4

Neste exercício, é pedido que, implemente um sincronizador que executa comandos (*Runnables*) nas suas *worker threads*. Existem uma regras estabelecidas que são: não pode haver um número de *threads* maior que o inteiro *maxPoolSize*, e o tempo máximo que uma *thread* continua viva sem executar nada é de *keepAliveTime*.

Para a implementação deste sincronizador são usados 4 métodos onde será explicado cada um.

O método execute é o método onde as *threads* dão trabalho ao sincronizador. Se o sincronizador estiver em modo *shutdown*, este não aceita mais comandos, e todas as *threads* que o chamarem saem com uma exceção. Quando o uma *thread* chama o execute esta vai primeiro ver se existem *threads* bloqueadas para dar o seu trabalho, se não houverem verifica de seguida se pode criar mais *threads*. É verificado primeiro se existem *threads* bloqueadas de modo a reduzir o custo de criação e destruição de uma *thread*. Se não houver *threads* bloqueadas, e o número de *threads* a executar já está no máximo, a *threads* vai se bloquear numa fila de *tasks*, com lógica FIFO, há espera que uma *workerthread* possa pegar no seu comando. Se não acabar o tempo de espera, e nenhuma *worker thread* ainda não executou o seu trabalho, este retira o seu pedido da fila e sai retornando false. Se ocorrer uma interrupção enquanto esta está à espera, ela retorna *true* se alguém já obteve o seu comando, ou caso contrário lança a exceção.

No método *shutdown* é alterado o *boolean shutdown* para *true*, de modo a sinalizar, e se houver alguma *thread* bloqueada, acordo-a de modo a concluir essa *thread*.

No método *awaitTermination* a *thread* verifica se o sincronizador está em modo *shutdown* e se não existe *threads* ativas, se isso se verificar esta sai retornando *true*. Caso contrário bloqueia-se até isso verificar-se ou então sai por *timeout*, retornado false.

O método *threadWork* é o método que a *worker thread* vai executar. Nesse método ele verifica se existe trabalho, se houver executa-o, mas sem ser em exclusão, de modo a melhorar a eficiência de execução diminuindo a janela de exclusão. Se não houver trabalho este verifica se o sinalizador está em modo *shutdown*. Se não estiver a *worker thread* bloqueia-se. Está pode ser interrompida no bloqueio, mas se houve uma *thread* que lhe delegou trabalho, esta não sai com exceção e começa a trabalhar. No fim do tempo de espera e se ninguém lhe atribuiu trabalho está finaliza-se, diminuindo o número de *threads* vivas e removendo-se da lista.