Coordenação II

Fila Bloqueante

O objetivo deste exercício é desenvolver uma estrutura de dados genérica para Filas Bloqueantes . Esta estrutura consiste numa Fila (*First In First Out*) com as operações **offer** (adiciona um elemento no fim da fila) e **poll** (retira e devolve o primeiro elemento da fila). Porém, as chamadas a estas operações são bloqueantes (*wait*) nas seguintes situações :

- offer (...) : no caso da fila estar no limite da sua capacidade.
- poll ( ) : no caso da fila estar vazia.

Desta forma, a fila pode ser acedida concorrentemente, e é útil para problemas cuja solução encaixa no padrão produtor-consumidor.
Implemente a estrutura numa classe chamada BlockingQueue , tendo em conta o seguinte:

- Os objetos são parameterizados com um parâmetro genérico <T> para o tipo de elementos da fila. Internamente, os elementos devem ser guardados numa estrutura apropriada para filas (convencionais), tal como LinkedList (lista ligada) ou ArrayDeque (vetor circular).
- Deverão ser disponibilizados dois construtores:
  - Sem parâmetros, e neste caso, a estrutura não terá uma capacidade máxima, e, logo, a operação offer nunca bloqueará e a operação poll não necessitará de notificar outras threads (pois não haverão chamadas bloqueadas em poll)
  - Com um parâmetro para definir a capacidade da fila (pré-condição: número positivo).
- Para além das operações offer e poll, deverá existir também a operação **size** (para obter o número de elementos da fila) e **clear** (para esvaziar a fila). As operações bloqueantes devem propagar as exceções de interrupção de thread (InterruptedException) para os clientes.

```
public class BlockingQueue <T> {
     private Queue <T> queue = ...
```

Cozinheiros e Glutões com Fila Bloqueante

Desenvolva uma nova solução para o problema dos Cozinheiros/Glutões utilizando a fila bloqueante desenvolvida (a qual substituirá o objeto da Mesa).

Barreiras

O objetivo deste exercício é implementar uma classe Barrier para concretizar barreiras. Neste tipo de sincronização, um grupo de threads bloqueia na barreira até que o número de threads bloqueadas na mesma atinja determinado valor (configurável).

- Deverá ser possível instanciar a barreira indicando o número de threads necessário para desbloquear a barreira.
- A operação bloqueante deverá obedecer à seguinte assinatura: `public synchronized void await() throws InterruptedException`
- Deverá existir uma operação para saber quantas threads estão bloqueadas na barreira
- A barreira deverá ser observável, dando origem a um evento por cada thread que se regista na barreira

Escreva uma pequeno programa para testar o comportamento da barreira. Por exemplo, criando 5 threads, as quais imprimem uma mensagem inicial, esperam um tempo aleatório entre 1 e 5 segundos, depois registam-se na barreira (instanciada para 5), e finalmente imprimem uma mensagem final. O efeito esperado será observar as mensagens finais a aparecerem simultaneamente.

Corredores com Barreiras

Arquivos anexados: Corredor.java (473 B)

Neste exercício o objetivo é simular uma corrida, à semelhança dos exercícios dos Cavalos e dos Carros. Porém, dados alguns requisitos mais específicos, deverá ser utilizada a barreira desenvolvida anteriormente para sincronizar o final da corrida. Utilizando a classe Corredor em anexo, desenvolva a classe Corrida de forma a que:

- A mensagem impressa por cada corredor apenas é mostrada quando todos os corredores terminaram. Desta forma, a chamada cheguei() é bloqueante, sendo desbloqueada apenas quando termina a corrida. O valor devolvido deverá ser o lugar em que o corredor ficou.
- À medida que os corredores terminam, o objeto Corrida imprime uma mensagem a informar qual dos corredores chegou.

O output do programa para 5 corredores poderá ser algo do género do que se apresenta em baixo, sendo que as primeiras 5 linhas aparecem à medida que cada corredor termina, ao passo que as últimas cinco linhas aparecem em simultâneo.

```
chegou Thread-4
chegou Thread-0
chegou Thread-1
chegou Thread-2
chegou Thread-3
Thread-3 : 5° lugar
Thread-0 : 2° lugar
Thread-1 : 3° lugar
Thread-4 : 1° lugar
Thread-2 : 4° lugar
```