Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação Concorrente, Verão de 2016/2017

Segunda Série de Exercícios

Resolva os seguintes exercícios usando como ponto de partida o código distribuído em anexo.

1. Implemente em *Java* e C# a classe **`ConcurrentQueue`**<T> que define um contentor com disciplina FIFO (*First In First Out*) suportado numa lista simplesmente ligada. A classe disponibiliza as operações **`put`**, **`tryTake`** e **`isEmpty`**. A operação **`put`** coloca no fim da fila o elemento passado como argumento; a operação **`tryTake`** retorna o elemento presente no início da fila ou **`null`**, no caso da fila se encontrar vazia; a operação **`isEmtpy`** indica se a fila está vazia. O contentor suporta acessos concorrentes e nenhuma das operações disponibilizadas bloqueia as threads invocantes.
   Nota: Na implementação tenha em consideração as explicações sobre *lock-free queue*, proposta por *Michael e Scott*, que consta no Capítulo 15 do livro J*ava Concurrency in Practice*.

2. Tirando partido dos mecanismos e técnicas *non-blocking*, discutidos nas aulas teóricas, implemente em *Java* e C# uma versão optimizada do sincronizador **`ThrottledRegion`** (cuja especificação consta na primeira série de exercícios). As optimizações devem incidir sobre as situações onde a entrada na região não bloqueia a *thread* invocante e a saída da região não precisa de libertar uma *thread* em espera. Caso seja necessário bloquear uma *thread*, use os monitores disponíveis nas plataformas *Java* ou .NET.

3. [**Opcional**] No artigo *Nonblocking Concurrent Data Structures with Condition Synchronization*, *William N. Scherer III* e *Michael L. Scott* propõem duas estruturas de dados *lock free*, para utilizar na comunicação de dados entre *threads*, designadas pelos autores por *dual stack* e *dual queue*. Os algoritmos propostos no artigo encontram-se aqui descritos em pseudocódigo.
   Tendo em consideração o artigo citado acima, o pseudocódigo associado ao artigo e o código distribuído no anexo, complete a implementação, em *Java*, da classe **`LockFreeDualQueue<T>`**. Esta classe define uma *dual data queue*, que se destina a suportar comunicação entre *threads*, em cenários produtor/consumidor, onde a espera em ciclo *busy-wait* seja adequada. A classe a implementar deve disponibilizar as operações **`enqueue`**, **`dequeue`** e **`isEmpty`**. A operação **`enqueue`** coloca no fim da fila o elemento passado como argumento, satisfazendo uma operação **`dequeue`** pendente, se existir; a operação **`dequeue`** retorna o item de dados mais antigo que se encontra na fila, forçando a *thread* invocante a esperar enquanto a fila estiver vazia; a operação **`isEmpty`** indica se a fila se encontra vazia ou se apenas contém nós inseridos pela operação **`dequeue`** (nós do tipo *request*).

Data limite de entrega: 4 de Junho de 2017

ISEL, 16 de Maio de 2017