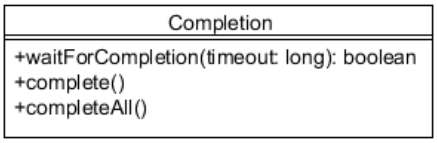
Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Programação Concorrente, Inverno de 2012/2013

Teste final (Recurso)

---

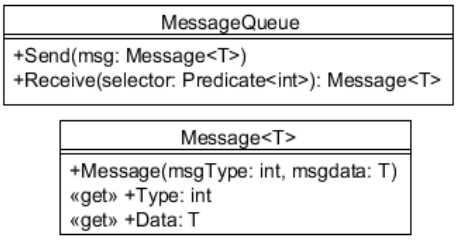
1. [5] Usando monitores intrínsecos Java, implemente o sincronizador `Completion`, que representa um gestor de unidades de conclusão de tarefas.



   A operação `complete` sinaliza a conclusão de uma tarefa e viabiliza a execução de exatamente uma chamada a `waitForCompletion`. A operação `waitForCompletion` bloqueia a *thread* invocante até que exista uma unidade de conclusão disponível, e pode terminar: com sucesso por ter sido satisfeita a condição de bloqueio, retornando `true`; produzindo `ThreadInterruptedException` caso a *thread* tenha sido interrompida, ou; retornando `false` se o tempo máximo de espera (`timeout`) foi atingido.

   O sincronizador inclui ainda a operação `completeAll` que o coloca permanentemente no estado sinalizado, ou seja, são viabilizadas todas as chamadas, anteriores ou posteriores, a `waitForCompletion`.

2. [5] Usando monitores intrínsecos CLI, implemente o sincronizador `MessageQueue` que promove a entrega de mensagens, com disciplina FIFO, às *threads* consumidoras.



   A operação `Send` promove a entrega da mensagem (`msg`) sem bloquear a *thread* invocante. A operação `Receive` promove a recolha da mensagem mais antiga que satisfaça o predicado especificado (`selector`), bloqueando a *thread* invocante caso não exista nenhuma mensagem que o satisfaça. O sincronizador garante a sinalização das *threads* há mais tempo em espera.

   Nota: Na implementação assuma que o tipo `Message<T>` é fornecido. Realize as alterações necessárias à interface pública de `MessageQueue` para que o sincronizador suporte cancelamento e desistência das *threads* em espera.

3. [4] Considere a classe `SpinMutexNonThreadSafe` apresentada:

```
public sealed class SpinMutexNonThreadSafe {
      private const Thread FREE = null;
      private Thread owner = FREE;
      private int acquisitionCount = 0;

      public void Enter() {
            if (owner == Thread.CurrentThread) acquisitionCount++;
            SpinWait sw = new SpinWait();
            while (true) {
                  if (owner == FREE) {
                        owner = Thread.CurrentThread;
                        return;
                  }
                  do { sw.SpinOnce(); } while (owner != FREE);
            }
      }

      public void Exit() {
            if (acquisitionCount == 0) owner = FREE;
            else acquisitionCount--;
      }
}
```

   a) [1,5] A classe não é *thread-safe*. Porquê?

   b) [2,5] Não recorrendo ao uso de primitivas bloqueantes, apresente as alterações necessárias (usando C#) para tornar a classe *thread-safe*. Para cada alteração realizada, justifique a opção tomada.

4. [3] Realize em C# o seguinte método estático, usando I/O assíncrono.

```
public static long CountIf(Stream source, Predicate<byte> p);
```

O método retorna o número de *bytes* de `source` que satisfazem o predicado `p` e produz no parâmetro de saída `fileSize` o número total de *bytes* de `source`. A leitura do *stream* deve ser realizada por blocos, existindo paralelização entre a leitura do bloco *N* e a verificação dos elementos do bloco *N* – 1.

Na implementação use a interface de `Stream` para I/O assíncrono (baseada no APM) composta pelos seguintes métodos:

```
IAsyncResult BeginRead(
   byte[] buffer,
   int offset,
   int count,
   AsyncCallback callback,
   object state);

int EndRead(IAsyncResult iar);
```

5. [3] Realize o seguinte método estático usando a TPL.

```
public static Task<int> ComputeAsync(
   Func<int> a,
   Func<int> b,
   Func<int,int,int> aggregate);
```

O método promove a execução assíncrona das funções `a` e `b` e produz a *task* cujo resultado é a agregação (`aggregate`) do resultado de `a` com o resultado de `b`. Apresente um troço de código que ilustre a utilização de `ComputeAsync` e que produz o resultado na consola.

Na implementação considere os seguintes tipos:

```
delegate TResult Func<in T1, in T2, out TResult>(T1 arg1, T2 arg2)
delegate TResult Func<out TResult>()
```

Duração: 2 horas e 30 minutos

Carlos Martins e *Paulo Pereira*

ISEL, 16 de Fevereiro de 2013