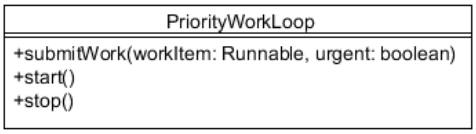
Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Programação Concorrente, Inverno de 2012/2013

Teste final

---

1. [4] Usando monitores intrínsecos Java ou CLI, implemente o sincronizador `PriorityWorkLoop` que promove a execução sequencial dos itens de trabalho submetidos na *thread* criada para esse efeito. Esta *thread* é iniciada na primeira chamada à operação `start`.

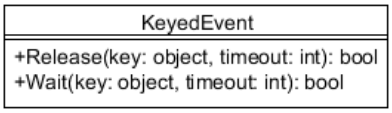


   A operação `submitWork` submete para execução unidades de trabalho (`workItem`) com a urgência indicada no parâmetro `urgent` e nunca promove o bloqueio da *thread* invocante. As unidades de trabalho são executadas pela ordem de submissão sendo dada prioridade às marcadas como urgentes.

   A operação `stop` inicia a sequência de terminação controlada, colocando o sincronizador no modo *shutdown*. Uma vez neste modo, o sincronizador garante a execução dos itens de trabalho já submetidos mas rejeita novas submissões, produzindo a excepção `InvalidOperationException` nas chamadas a `submitWork`. Por simplificação admita que, uma vez no modo *shutdown*, as chamadas às operações `start` e `stop` produzem a excepção `InvalidOperationException`.

   Nota: Na eventualidade de implementar o sincronizador em C# não se esqueça de realizar as adaptações que considerar necessárias à interface pública apresentada na figura.

2. [5] Usando monitores intrínsecos Java ou CLI, implemente o sincronizador `KeyedEvent`. O sincronizador fornece duas operações, ambas bloqueantes. A operação `Release` bloqueia a *thread* invocante até que seja chamada a operação `Wait` especificando a mesma `key`. A operação `Wait` bloqueia a *thread* invocante até que seja chamada a operação `Release` com a mesma `key`.



   Ambas as operações podem 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 foi atingido.

   Nota: Na eventualidade de implementar o sincronizador em Java, não se esqueça de realizar as adaptações que considerar necessárias à interface pública apresentada na figura.

3. [4] Considere a classe `TypeWithLazyIdNonThreadSafe` apresentada a seguir:

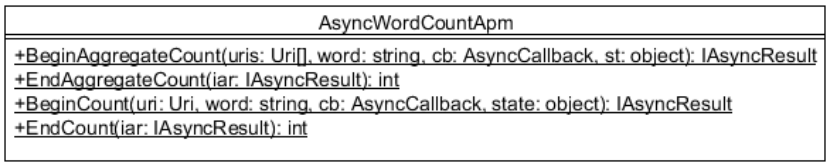
```
public class TypeWithLazyIdNonThreadSafe {
      private static int idSeed;
      private int id;

      public int getId() {
            if (id == 0) {
                  nid = 0;
                  do { nid = ++idSeed;} while (nid == 0);
                  if (id == 0) id = nid,
            }
            return id;
      }
      //...
}
```

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

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

   Note que o valor `0` no campo `id` simboliza que ainda não foi atribuído nenhum `valor` à instância (não foi concluída nenhuma chamada a `getId`) e, por isso, não pode ser usado como valor válido.

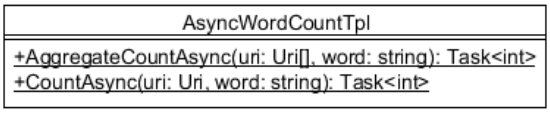
4. [5] Considere a classe `AsyncWordCountApm` que fornece operações assíncronas (baseadas no APM) para contagem de ocorrências de palavras em documentos HTML obtidos da web.



   Os métodos `BeginCount` e `EndCount`, cuja implementação é fornecida, correspondem à operação assíncrona que produz a contagem de ocorrências da palavra `word` no documento com o endereço `uri`.

   Implemente os métodos `BeginAggregateCount` e `EndAggregateCount`, que correspondem à operação assíncrona que produz a contagem de ocorrências da palavra `word` nos documentos com os endereços contidos no *array* `uris`. Na implementação faça uso dos métodos fornecidos e tenha em conta a necessidade de realizar uma implementação de `IAsyncResult` para acumulação dos resultados parciais. Por simplificação, assume-se que as operações assíncronas não produzem erros.

5. [2] Considere a classe `AsyncWordCountTpl` que fornece operações assíncronas (baseadas na TPL) para contagem de ocorrências de palavras em documentos HTML obtidos da web.



   O método `CountAsync`, cuja implementação é fornecida, produz a *task* cujo resultado é a contagem de ocorrências da palavra `word` no documento com o endereço `uri`.

   Implemente o método `AggregateCountAsync` que produz a *task* cujo resultado é a contagem de ocorrências da palavra `word` nos documentos com os endereços contidos no *array* `uris`. Na implementação faça uso do método fornecido.

Duração: 2 horas e 30 minutos

Carlos Martins e *Paulo Pereira*

ISEL, 18 de Janeiro de 2013