Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Programação Concorrente

Verão de 2022/2023, Segunda Série de Exercícios

Resolva os seguintes exercícios e apresente os testes com os quais validou a correção da implementação de cada exercício. A entrega deve ser feita através da criação da tag **`0.2.0`** no repositório individual de cada aluno.

1. Implemente uma classe com a mesma funcionalidade da classe **`java.util.concurrent.CyclicBarrier`**.

2. Implemente, sem utilizar *locks*, uma versão *thread-safe* da classe **`UnsafeContainer`** que armazena um conjunto de valores e o número de vezes que esses valores podem ser consumidos.

```
class UnsafeValue<T>(val value: T, var initialLives: Int)
class UnsafeContainer<T>(private val values: Array<UnsafeValue<T>>){
    private var index = 0
    fun consume(): T? {
       while(index < values.size) {
           if (values[index].lives > 0) {
               values[index].lives -= 1
               return values[index].value
           }
           index += 1
       }
       return null
    }
}
```

   A título de exemplo, o contentor construído por **`Container(Value("isel", 3), Value("pc", 4))`** retorna, através do método **`consume`**, a string **`"isel"`** três vezes e a string **`"pc"`** quatro vezes. Depois disso, todas as chamadas a **`consume`** retornam **`null`**.

3. Considere a seguinte implementação não *thread-safe* de um contentor de objectos com contagem de utilizações, que automaticamente chama a função **`close`** quando essa contagem de utilizações é zero. Realize, sem utilizar *locks*, uma versão *thread-safe* desta classe.

```
class UnsafeUsageCountedHolder<T : Closeable>(value: T) {
   private var value: T? = value
   // the instance creation counts as one usage
   private var useCounter: Int = 1

   fun tryStartUse(): T? {
       if (value == null) return null
       useCounter += 1
       return value
   }

   fun endUse() {
       if (useCounter == 0) throw IllegalStateException("Already closed")
       if (--useCounter == 0) {
           value?.close()
           value = null
       }
   }
}
```

4. Implemente a funcão **`fun <T> any(futures: List<CompletableFuture<T>>): CompletableFuture<T>`** que, dado uma lista não vazia de *futures*, retorna um *future* que se completa:
   - Com sucesso, quando um qualquer *future* da lista se completar com sucesso. O valor do *future* retornado deve ser o valor do *future* da lista que se completou.
   - Com excepção, quando todos os *futures* da lista se completarem com excepção. A excepção do *future* retornado deve agregar as excepções de todos os *futures* da lista.

   Esta funcionalidade é semelhante à da função **`Promise.any`** da linguagem JavaScript.

   Valoriza-se a minimização da aquisição de *locks* na implementação desta função.

Data limite de entrega: 14 de maio de 2023

ISEL, 17 de abril de 2023