

Exercícios de concorrência

Grupo de Sistemas Distribuídos
Universidade do Minho

Exercícios propostos

Manual

1 Considere um sistema para organizar *raids* num jogo multi-jogador. Cada jogador indica qual o número mínimo de participantes no *raid*. Os participantes no próximo *raid* ficam definidos mal existem jogadores à espera em número suficiente para satisfazer os requisitos de todos eles (em termos de número de participantes). Os jogadores que apareçam mais tarde serão agrupados no *raid* seguinte. Logo que possível, o sistema 1) indica a cada jogador os nomes dos outros jogadores desse *raid*; 2) quando o *raid* pode começar pois o sistema só permite que estejam até R *raids* a decorrer em simultâneo.

Apresente duas classes Java (para serem usadas no servidor) que implementem as interfaces abaixo, tendo em conta que os seus métodos serão invocados num ambiente *multi-threaded*.

```
interface Manager {
    Raid join(String name, int minPlayers) throws InterruptedException;
}
interface Raid {
    List<String> players();
    void waitStart() throws InterruptedException;
    void leave();
}
```

A operação `join` deverá bloquear até estar formado o grupo de participantes no *raid*, devolvendo o objeto que o representa; tem como parâmetro o nome do jogador e o número mínimo de jogadores que o *raid* deve ter. A operação `players` devolve a lista de jogadores presentes no *raid*; `waitStart` deverá bloquear até o *raid* poder começar (só podem estar R a decorrer em simultâneo); `leave` é invocada quando um jogador abandona o *raid*, que termina quando todos os jogadores o tiverem feito.

Simplificação: (max. 60% da cotação) Apresente apenas uma classe Java com a interface `Manager` em que o método `join` devolve diretamente a lista de participantes (`List<String>` em vez de `Raid`) e ignore o limite R . Deve no entanto cumprir os requisitos em termos de número de participantes.

2 Considere um sistema de controlo de acesso a dois recursos. Apenas um recurso pode estar a ser acedido em cada momento e no máximo podem estar T threads a aceder a um recurso. Pretende-se que escreva em Java, fazendo uso de primitivas baseadas em monitores, uma classe que implemente a interface:

```
interface Controller {  
    int request_resource(int i);  
    void release_resource(int i);  
}
```

O `request_resource` tem como parâmetro o recurso pretendido (identificado por um número que pode ser 0 ou 1), e deverá bloquear até o recurso poder ser acedido; `release_resource` é invocado quando uma thread completou o uso do recurso correspondente. Tente evitar *starvation*.