

Sincronização em programas concorrentes: Problema do banheiro unisex

Vinícius Campos Tinoco Ribeiro

Outubro - 2017

Introdução

No contexto da programação concorrente, um dos principais desafios é garantir a corretude na sincronização de procesos e *threads* distintos. Então, este trabalho tem como objetivo solucionar um problema conhecido no mundo da concorrência, através de dois mecanismos diferentes.

O problema em questão, denominado como o **problema do banheiro unisex**, trata-se da situação em que pessoas de sexos diferentes compartilham o mesmo banheiro, porém só é permitido que pessoas do mesmo sexo utilizem ao mesmo tempo, sendo necessário a criação de um mecanismo de controle para esta situação. Objetivando solucionar este problema, propõe-se neste trabalho duas soluções utilizando **semáforos** e **monitores**, sendo uma solução para cada mecanismo, implementadas na linguagem **java**.

1 Solução proposta

Visando as boas práticas de programação, tentou-se modularizar o projeto, ao passo que partes específicas do projetos pudessem ser removidos e/ou alterados sem causar danos a solução existente. Então, foi definida uma simples arquitetura em camadas, mostrada na Figura 1.

O projeto possui uma arquitetura de três camadas, onde temos a camada base, chamada de *Model*, a qual abriga a classe *Person*, que refere-se a homens e mulheres, assim como métodos de impressões específicas de uma pessoa. Já a camada *Business* é a responsável por implementar o algoritmo de gerenciamento de pessoas no banheiro. Nessa camada, existem três subcamadas:

- **Controller:** Nesta subcamada existe uma classe abstrata (*BathroomManager*) comum para os dois mecanismos de concorrência. Então, as classes *BathroomManagerSemaphores* e *BathroomManagerMonitors* estendem *BathroomManager*, as quais implementam o gerenciamento através de semáforos e monitores, respectivamente.
- **Threads:** Esta subcamada possui duas *threads*: *JobIn* e *JobOut*. A primeira é responsável por tentar fazer uma pessoa entrar no banheiro a cada segundo, além de que é responsável por disparar a *thread* **JobOut**. Já a segunda, **JobOut**, é responsável por fazer as pessoas saírem do banheiro, além de inserir novas pessoas que estavam na fila de espera, seja porque o banheiro estava cheio, ou porque eram de sexos diferentes.

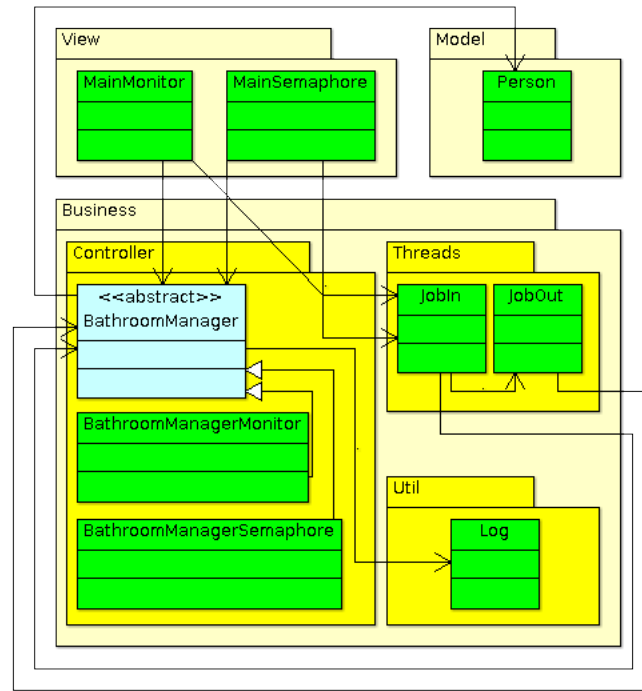


Figura 1 – Diagrama de classes do projeto

- **Util:** Esta camada possui uma única classe, *Log*, que se responsabiliza por realizar as impressões do histórico de eventos.

Por fim, a camada *View* é responsável por executar as soluções com semáforo e monitor.

1.1 Estratégia

Foi adotada uma estratégia simples, mas que solucionasse o problema, estratégia que consiste nos seguintes algoritmos:

Algoritmo 1: PROBLEMA DO BANHEIRO UNISEX (ENTRADA)

Entrada: *capacidade*

```

1 início
2   se banheiro estiver vazio então
3     banheiro.add(nova_pessoa);
4     imprime(nova_pessoa);
5   senão
6     se banheiro.primeira_pessoa.sexo == nova_pessoa.sexo e banheiro.size <
        capacidade e pessoas_esperando não estiver vazio então
7       banheiro.add(nova_pessoa);
8       imprime(nova_pessoa);
9     senão
10      pessoas_esperando.add(nova_pessoa);
11    fim
12  fim
13 fim

```

Algoritmo 2: PROBLEMA DO BANHEIRO UNISEX (SAÍDA)

Entrada: *capacidade*

```
1 início
2   se banheiro não estiver vazio então
3       ordenar(banheiro);
4       enquanto banheiro não for vazio e banheiro.tempo ==
           banheiro.primeira_pessoa.tempo faça
5           banheiro.remove(banheiro.primeira_pessoa);
6           imprime(pessoa_removida);
7       fim
8       se banheiro.size < capacidade e pessoas_esperando não for vazio então
9           enquanto pessoas_esperando não estiver vazio e
               pessoas_esperando.primeira_pessoa.sexo ==
               banheiro.primeira_pessoa.sexo e banheiro.size < capacidade faça
10              banheiro.add(pessoas_esperando.primeira_pessoa);
11              pessoas_esperando.remove(pessoas_esperando.primeira_pessoa);
12              imprime(pessoa_adicionada);
13          fim
14      fim
15  fim
16 fim
```

2 Mecanismos utilizados

Para solucionar o problema do banheiro unisex, foram utilizados dois mecanismos de sincronização: **semáforos** e **monitores**, os quais já foram mencionadas anteriormente.

2.1 Semáforos

Para este mecanismo, utilizou-se um **semáforo binário**, de modo que todas as operações que se refiram ao banheiro tenham seus acessos de forma atômica, impossibilitando a inconsistência de dados, além de deadlocks, livelocks e starvations. Todos os métodos da classe *BathroomManagerSemaphores* utilizam o semáforo, os quais possuem a seguinte estrutura:

```
1 try{
2     semaphore.acquire();
3     código do método;
4 } catch(exception) {
5     erro;
6 } finally {
7     semaphore.release();
8 }
```

2.2 Monitores

Neste mecanismo, foi utilizado a palavra reservada do java, **synchronized**, em todos os métodos da classe **BathroomManagerMonitor** que sobrecarregam os métodos da classe abstrata **BathroomManager**. Além disso, todas as variáveis voláteis, informando que as variáveis irão ser disputadas concorrentemente. As implementações dos corpos dos métodos são exatamente os mesmos utilizados com semáforos, trocando apenas o uso de *try-catches* e semáforos por métodos **synchronizeds**.

Para evitar livelocks, foi garantido que não existem ciclos de chamadas de métodos “sincronizados”, além de atribuir as variáveis como voláteis.

3 Fluxo de execução

Ao executar os programas, o *BathroomManager* é instanciado através de uma das classes concretas, já mencionadas acima. Além disso, a classe *JobIn* é iniciada e tem como parâmetros a referência do *BathroomManager* e o tempo limite que as pessoas podem ficar no banheiro(valor utilizado como referência para a geração randômica do tempo de cada indivíduo no banheiro).

Na *thread JobIn*, a cada 1s uma nova pessoa tenta ter acesso ao banheiro e a *thread JobOut* é inicializada, caso ainda não tenha sido. A *thread JobOut* também recebe a referência para o *BathroomManager* e é responsável por remover as pessoas do banheiro quando o tempo de cada pessoa é esgotado.

Considerações finais

Este relatório apresentou uma solução para o problema do banheiro unisex, utilizando dois mecanismos de sincronização de *processos/threads*, **semáforos** e **monitores**. O processo de desenvolvimento foi facilitado devido a arquitetura em camadas utilizada, com isso, os corpos dos métodos foram os mesmos para os dois mecanismos, mudando apenas as partes que se referem a cada um dos mecanismos.

Para maiores informações sobre como executar o projeto, assim como instruções de execução, basta acessar este [link](#) no github.