Exclusão Mútua

Grupo de Sistemas Distribuídos
Universidade do Minho

1 Objectivos

Evitar *corridas*, pela utilização de mecanismos que garantam *exclusão mútua* na execução de *secções críticas*. Observar a ocorrência de *deadlock*, e implementar solução que garanta a sua ausência.

2 Mecanismos

Mecanismo nativo de exclusão mútua, via *lock* reentrante (que permite que uma thread invoque aninhadamente/recursivamente métodos/blocos sobre o mesmo objecto), disponível em todos os objectos, em duas variantes:

- método synchronized, que usa o lock de `this`:
  ```
  synchronized T m(...) { ... }
  ```
- bloco synchronized, que usa o lock de *obj*:
  ```
  synchronized (obj) { ... }
  ```

3 Exercícios propostos

1. Modifique o exercício do guião anterior de incremento concorrente de um contador partilhado, de modo a garantir a execução correcta do programa.
2. Implemente uma classe Banco que ofereça os métodos da interface abaixo, para crédito, débito e consulta do saldo total de um conjunto de contas. Considere um número fixo de contas, definido no construtor do Banco, com saldo inicial nulo. Utilize exclusão mútua ao nível do objecto Banco.

   ```
   interface Bank {
     void deposit(int id, int val) throws InvalidAccount;
     void withdraw(int id, int val) throws InvalidAccount, NotEnoughFunds;
     int totalBalance(int accounts[]) throws InvalidAccount;
   }
   ```

3. Acrescente o método transferir à classe Banco:

   ```
   void transfer(int from, int to, int amount) throws InvalidAccount, NotEnoughFunds;
   ```

   Considere a viabilidade de este ser implementado simplesmente como a composição sequencial das operações de débito e crédito já implementadas.
4. Reimplemente a classe Banco utilizando exclusão mútua ao nível das contas inviduais, evitando fazer lock do banco todo.