Execução em Paralelo e Controlo de Concorrência Simples

- controlo de concorrência simples (synchronized: métodos ou bloco)
- controlo de concorrência mais elaborado

- wait/notify - ver API em java.lang.Object
- semáforos
  - `java.util.concurrent.Semaphore`
- outras classes especiais para este efeito...

---

exercício 1: "contas bancárias" e execução em paralelo
Obtenha o código inicial:

- netbeans
- eclipse

a) Estude o código da classe `sd.cc.Conta`, sem alterar o código.

b) Veja também a classe `ThreadsBancarias` e note que operações são realizadas por cada thread ativada.
Repare que a conta bancária é a mesma para todas as threads...
No os valores usados, em cada propósito, para as threads.

exercício 2
Compile a classe ThreadsBancarias e de seguida execute a aplicação resultante, observando com atenção o output emitido.
Nota alguma inconsistência nos resultados de cada thread?
Repita várias vezes a execução.

exercício 3
Independentemente do que observou, introduza uma demora na zona de processamento das operações sobre o saldo.
Primeiro descomente o o código `sleep`() no método `credito`() da classe `Conta`.
Compile e execute...
Nota alguma incoerência?

Descomente também o código `sleep`() no método `debito`().
Compile e execute...
Nota alguma incoerência?
Volte a executar, o resultado **pode variar em função do escalonador** de Threads da JVM!

exercício 4
Resolva o problema de acesso concorrente do ponto anterior com uma **pequena alteração** na classe `Conta`.
Repita a execução e confirme que o problema está resolvido.

Execute várias vezes para ver outras sequências possíveis no escalonamento das Threads.
ver aqui

---

exercício 5

Remova a alteração anterior.
Introduza uma forma de controlar o acesso à mesma zona crítica, com base no lock sobre o objeto conta.

```
synchronized ( lockObject ) {
 ...
}
```

ver aqui

Outras formas de sincronizar: `wait / notify` sobre objetos disputados pelas threads

API java.lang.Object

Exercício 6
a) Altere a ordem pela qual são ativadas as Threads... lance primeiro p3.

b) Altere o método `debito()`:
Se não existir saldo, **então** esperar 5 segundos e tentar novamente...
Se nessa altura não houver saldo suficiente, exibir uma mensagem de aviso e cancelar a operação. Teste a sua solução.

Exercício 7
Pense numa forma mais eficiente de concretizar a operação de débito assim que existir saldo suficiente, **sem ter de esperar** mais X segundos até ao final do período especificado... `wait() / notify()`.

Mais leituras:

- https://dzone.com/articles/the-evolution-of-producer-consumer-problem-in-java
- https://docs.oracle.com/javase/specs/jls/se7/html/jls-17.html (sobre wait e notify)