DEPARTAMENTO
DE CIÊNCIAS E TECNOLOGIAS

Paradigma da Programação

Adaptado dos exercícios do Prof. Paulo Cabrita

Objectivos:

No final da aula o aluno deverá saber:
- Como garantir o acesso robusto a recursos partilhados;
- Como se sincronizam métodos;
- Que utilizar threads e identificar problemas de sincronização é delicado, pois muitas vezes não se manifestam;
- Utilizar o join para esperar que um thread termine;

Ferramentas usadas: NetBeans e Java 8

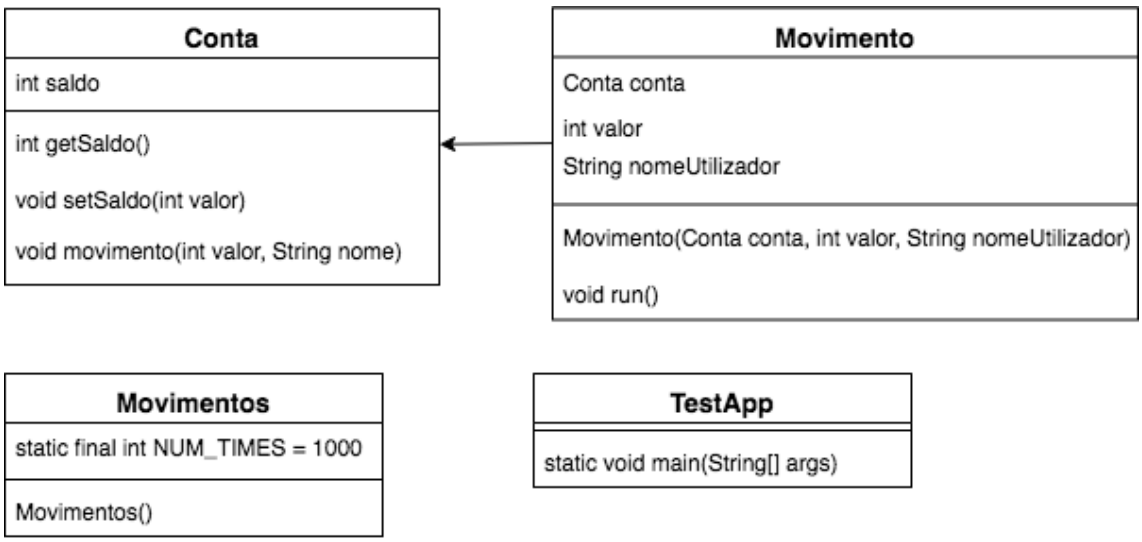
Material de apoio:

- Tutorial: https://www.tutorialspoint.com/java/

- Material apresentados nas aulas teórica

Exercícios:

1. Simular o acesso a uma conta bancária remota com a interface: int getSaldo(), void setSaldo(int valor), e void movimento(int valor, String nome).
   - Fazer um número significativo de depósitos (1000) e o mesmo número de levantamentos e no mesmo valor para evidenciar a necessidade de sincronização.
2. Resolva o problema anterior sincronizando o acesso à conta partilhada.



Exemplo de código para as classes Movimentos e TestApp:

```
public class Movimentos {
   static final int NUM_TIMES = 1000;
   public Movimentos() {
      long tempoInicial = System.currentTimeMillis();
      Conta conta = new Conta();
      Movimento[] depositos = new Movimento[NUM_TIMES];
      Movimento[] levantamentos = new Movimento[NUM_TIMES];
      for(int i=0;i< NUM_TIMES; i++){
         depositos[i] = new Movimento(conta, 1000, "Deposito "+i);
         levantamentos[i] = new Movimento(conta, -1000, "\t\rt\tLevantamento "+i);
      }
      for(int i=0;i< NUM_TIMES; i++){ //Espera que todos os threads terminem
         try {
            depositos[i].join();
            levantamentos[i].join();
         } catch (InterruptedException ex) {
           System.out.println(ex);
         }
      }
       System.out.println("O saldo final é " + conta.getSaldo());
       long time = System.currentTimeMillis()-tempoInicial;
       System.out.println("Demorei "
                                 + new SimpleDateFormat("mm:ss:SS").format(new Date(time)));
   }
}
```

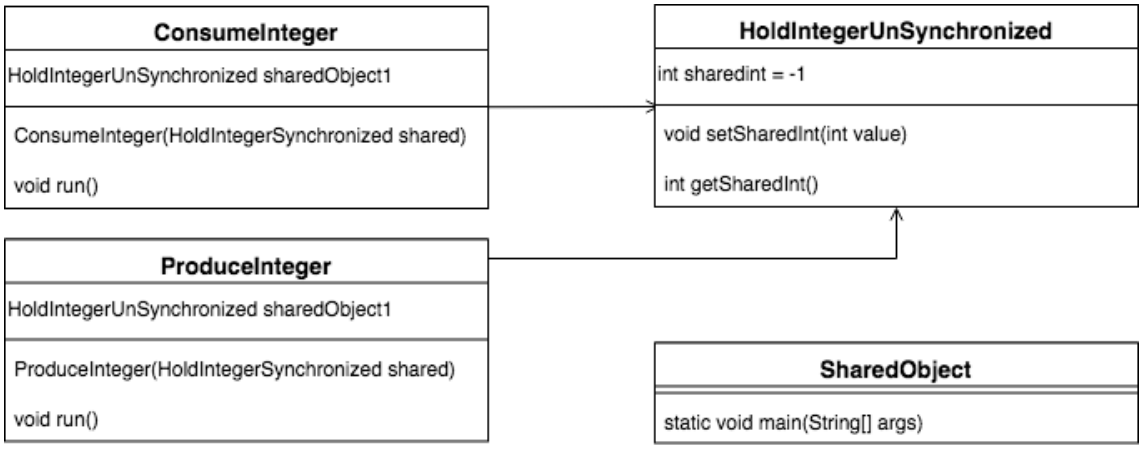
Nº de vezes que serão feitos os levantamentos e depósitos

Registar o tempo de execução da aplicação

Mesmo nº de depósitos e levantamentos

```
public class TestApp {
   public static void main(String[] args) {
      new Movimentos(); } }
```

3. Considere um relacionamento produtor/consumidor em que uma thread produtora deposita uma seqüência de números (utilizamos 1,2, 3,...) em um variável compartilhada. A thread consumidora lê esses dados da memória compartilhada e imprime os dados. O objetivo é imprimir o que o produtor produz à medida que ele produz.
   Desenvolva duas aplicações Java usando threads:
   a) As threads não são sincronizadas
   b) As threads estão sincronizadas



Não esqueça de enviar os ficheiros com a resolução dos exercícios até o fim da aula!
(Em pares ou individualmente)