	UNIVERSIDADE FEDERAL DA PARAÍBA CENTRO DE INFORMÁTICA	
	Disciplina	Linguagem de Programação II
	Semestre	2017.1
	Professor	Bruno Jefferson de Sousa Pessoa

Primeiro Exercício-Programa: *Locks* e Algoritmos Justos

1. O Problema da Montanha Russa

Suponha que existam *n* passageiros e um carro em uma montanha russa. Os passageiros, repetidamente, esperam para dar uma volta no carro. O carro tem capacidade para *C* passageiros, com $C < n$. O carro só pode partir quando estiver cheio. Após dar uma volta na montanha russa, cada passageiro passeia pelo parque de diversões e depois retorna à montanha russa para a próxima volta.

Tanto o carro quanto os passageiros devem ser representados por threads. A implementação das threads dos passageiros devem basear-se no seguinte pseudocódigo:

```
thread passageiro[i = 1 to n]  {
      while (!fechouParque) {
            entraNoCarro(); // incrementa contador
            esperaVoltaAcabar();
            saiDoCarro(); // decrementa contador
            passeiaPeloParque(); // tempo aleatório
      }
}
```

Da mesma forma, a implementação da thread do carro deverá a seguinte estrutura:

```
thread carro {
    while (existemPassageirosNoParque) {
        esperaEncher();
        daUmaVolta();
        esperaEsvaziar();
        volta++; // Indicador para o fechamento do parque.
    }
}
```

2. Instruções gerais

- O presente Exercício-Programa(EP) deve ser desenvolvido em grupo de no máximo 3 (três) alunos.
- O EP deve ser entregue no dia **13/09/2017**.
- Deverá ser entregue um arquivo zipado, contendo o código fonte e o arquivo executável, no formato descrito a seguir:
  - LP2-EP1-grupo.zip
  - Ex.: LP2-EP1-jose-maria.zip
- O EP será apresentado pelo grupo no dia de sua entrega.

3. Instruções para implementação

- Este Exercício-Programa deve ser desenvolvido em Java ou em C++, utilizando a implementação de Threads pertencente ao seu núcleo.
- *Locks* devem ser utilizados para a sincronização do acesso às seções críticas.
- A implementação deverá atender às quatro propriedades de uma solução para o problema da seção crítica, a saber: Exclusão mútua, Ausência de *deadlock*, Ausência de atraso desnecessário e Entrada eventual.
- Para atender à propriedade da Entrada eventual, o programa deverá basear-se no **Algoritmo da Padaria** apresentado em sala de aula.
- Para realizar instruções de incremento e decremento de forma atômica em C++, utilizem o método `fetch_add()`. Em Java, utilizem o método abaixo:

```
// Em java
public static synchronized int FetchAndAdd(MeuInteiro var, int incr) {
      int tmp = var.getValor();
      var.setValor(var.getValor() + incr);

      return tmp;
}
```

- O carro só pode partir se estiver cheio.
- A saída do seu programa deve ser bem planejada, de forma a mostrar o que está acontecendo a cada momento, sem ficar carregada demais.
- No passeio pelo parque, as threads devem dormir um tempo aleatório dentro de um intervalo preestabelecido que tenha relação com o tempo da volta no carro, o qual deve ser constante.