UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
Prof. Vagner

Threads em Java
Atividade 2

Exercício 1

- Defina uma classe Contador como uma subclasse de Thread, que imprime números de 0 a 10. Crie a classe TesteContador que deve definir o método *main()* que cria e inicia a execução da *thread* Contador. Teste o resultado executando a classe TesteContador.
- Altere as classes Contador e TesteContador de modo que a classe Contador seja definida como uma implementação da interface Runnable. Teste o resultado.
- Altere o método *main()* da classe TesteContador para criar duas ou mais *threads* Contador e inicialize a execução das mesmas.

  Dica: Na última parte do exercício, onde é pedido para alterar o método *main()* da classe TesteContador, você deve inicialmente criar as *threads* e em seguida inicializá-las, de modo a obter um melhor resultado na visualização da execução concorrente.

Exercício 2

O cenário é um depósito de caixas. Um produtor armazena as caixas que vai produzindo e um consumidor retira essas mesmas caixas. A classe apresentada abaixo corresponde ao Depósito, com os respectivos métodos *armazenar()* e *retirar()*.

```
public class Deposito {
      private int items=0;
      private final int capacidade=10;
      public int retirar() {
            if (items>0) {
                  items--;
                  System.out.println("Caixa retirada: Sobram "+items+" caixas");
                  return 1; }
            return 0;
      }
      public int colocar () {
            if (items<capacidade) {
                  items++;
                  System.out.println("Caixa armazenada: Passaram a ser "+items+"
                                          caixas");
                  return 1; }
            return 0;
      }
      public static void main(String[] args) {
            Deposito dep = new Deposito();
            Produtor p = new Produtor(d, 2);
            Consumidor c = new Consumidor(d, 1);
            //inicia o produtor
            //...
            //inicia o consumidor
```

```
            //...
            System.out.println("Execução do main da classe Deposito terminada!");
}
```

2.1

Crie uma classe Produtor que funcione como uma *thread* e que vai invocando o método `armazenar()` da classe Depósito, acrescentando caixas ao depósito. A classe Produtor deve receber, através do construtor, uma referência ao objeto `dep` onde os métodos vão ser invocados e um inteiro correspondente ao tempo em segundos entre as produções de caixas. Defina a classe `Produtor` como sendo uma classe que implementa o método Runnable.

Crie uma classe Consumidor que funcione como uma *thread* e que vai invocando o método `retirar()` da classe Depósito, retirando caixas do depósito. A classe Consumidor deve receber, através do construtor, uma referência ao objeto `dep` onde os métodos vão ser invocados e um inteiro correspondente ao tempo em segundos entre as retiradas de caixas. Defina a classe Consumidor como sendo uma classe que implementa o método Runnable.

Perceba que a thread Produtor deve produzir itens sucessivamente, enquanto que a thread Consumidor deve consumi-los sucessivamente e que ambas estarão rodando em paralelo.

Execute o sistema e faça experiências:
a) Adicione à classe Consumidor mensagens que permitam identificar o que cada objeto Consumidor está fazendo e, em particular, se está bloqueado à espera de caixas para retirar.
b) Altere o número de consumidores ou de produtores e os tempos médios entre produções e consumos.

2.2

A existência de threads concorrentes exige a necessidade de sincronização. Cada objeto, em Java, tem associado um monitor que garante o acesso exclusivo às seções críticas do objeto, ou seja, às áreas compartilhadas pelas threads. O programador precisa assinalar a seção crítica usando synchronized. Um bloco de código sincronizado é uma região que apenas pode ser executada por uma thread de cada vez.

Analise a classe Depósito e identifique possíveis problemas de concorrência. Altere a implementação da classe usando synchronized.

2.3

Além da sincronização, é importante permitir a coordenação entre as threads. Os métodos utilizados são wait(), notify() e notifyAll(). Utilizando estes métodos, altere a implementação do sistema para permitir uma coordenação adequada entre os objetos.