Exercício 1 (Modificador synchronized)

Considere o problema de múltiplas threads acessando e modificando concorrentemente um recurso compartilhado. Considere, por exemplo, que múltiplas threads precisam imprimir, de forma consistente, mensagens na classe SlowConsole apresentada abaixo:

```
class SlowConsole {
     public void println(String msg) {
          if (msg == null) throw new NullPointerException("Invalid parameter");
          for (int i=0; i<msg.length; i++)  slowPutchar(msg.charAt(i));
          slowPutchar('\n');
     }
     private void slowPutchar(char c) {
          System.out.print(c);
          try { Thread.sleep(100); }
          catch (InterruptedException ie) {}
     }
}
```

Atividades Propostas:

1. Qual o problema com esta implementação da classe SlowConsole? Construa o programa Java onde a classe SlowConsole imprime na tela de forma inconsistente (código do exemplo).

2. Proponha uma alteração a essa classe utilizando o modificador synchronized para resolver o problema de impressão inconsistente.

Exercício 2 (Executors)

Considere um serviço que responde a diversas requisições e que precisa acessar dados compartilhados.

```
package bigserver;
import java.util.concurrent.*;

public class BigServer {
    public class ServeOne implements Runnable {
        @Override
        public void run() {
            long max = ThreadLocalRandom.current().nextLong(20000, 100000);
            long sum = 0;
            for (long i = 1; i < max; i++) {
                sum += i;
            }
        }
    }

    public void run() {
        for(int i = 0; i < 5000; i++) {
            Thread t = new Thread(new ServeOne());
            t.start();
        }
    }

    public static void main(String[] args) {
        (new BigServer()).run();
    }
}
```

Atividades Propostas:

1. Modifique o exemplo para que utilize um *pool* de threads

2. Verifique o desempenho de cada uma das versões **e explique** quais fatores influenciam nesta.

Exercício 3 (wait/notify)

Considere as classes abaixo:

```
class Buffer {
        private Queue<String> data = new LinkedList<String>();
        public String remove() {}
        public void add(String s) {}
}
class Output implements Runnable {
        private Queue<String> data = new LinkedBlockingQueue<String>();
        public void add(String s) {}
        //Coloca um elemento do buffer na saida
        public void run() {
        }
}
class Produtor implements Runnable {
        // Produz um elemento e coloca no buffer
        public void run() {
        }
        Produtor(Buffer b) {}
}
class Consumidor implements Runnable {
        //Retira um elemento do buffer, processa e adiciona NA CLASSE de saida
        public void run() {
        }
        Consumidor(Buffer b, Output o) {}
}
```

Atividades Propostas:

1. Implemente os métodos incompletos e uma versão com ao menos duas threads de cada classe (Produtor, Consumidor, Output) utilizando um buffer limitado e wait/notify para a coordenação do buffer.