→ Multithreaded Servers

1 – Implemente o Servidor da data e hora do sistema estudado na aula teórica ("multithreaded server".

2 – Modifique o programa anterior de modo a conseguir demonstrar experimentalmente que o servidor pode servir vários clientes em simultâneo.

3 – Construa uma aplicação cliente / servidor em que o servidor receba do cliente dois arrays de inteiros, calcule a sua soma e devolva o resultado ao cliente. O servidor deverá poder servir vários clientes simultaneamente.

→ Sincronização e Transferência de controlo entre Threads

4 - Suponha dois processos **p1** e **p2** que partilham uma variável comum, **variavelPart**. Pretende-se construir um exemplo que ilustre a violação de uma secção crítica, sem usar qualquer tipo de mecanismo de sincronização,

Considere que o processo p1 possui duas variáveis locais, x e y, inicializadas com valores simétricos, e que dentro de um ciclo infinito transfere a quantidade armazenada em **variavelPart** de x para y. O processo 2 vai, em cada iteração, incrementar a variável a.

Pretende-se que a condição x + y = 0 seja verdadeira durante toda a execução do programa. Quando, no processo p1, se detecta que a secção crítica foi violada (porque x + y **!=** 0) o processo deve terminar e acabar o programa.

a) Supondo a estrutura que se segue para os processos p1 e p2, comece por criar duas classes que permitam criar os processos p1 e p2. Estes dois processos deverão, por exemplo, partilhar um valor do tipo array de inteiros com um elemento. (Porque não um valor do tipo int ?)

b) Construa uma classe de teste que, instanciando os processos p1 e p2, lhe permita simular a violação da secção crítica.

c) Para que o processo p2 termine, após a violação da secção crítica, transforme-o numa Thread daemon.

Processo 1

```
...
 x=M;
 y= - M;
 While (true){
          //secção crítica  1
      x=x – variavelPart;
      y=y+ variavelPart;
      <parte restante 1>
      if (x+y != 0 ){
           print "Secção crítica violada
           break;
      }
    ...
 }
```

Processo 2

```
...
While (true){
      //secção crítica  2
       variavelPart=variavelPart +1;
      <parte restante 2>
 }
…
```

d) Pretende-se agora garantir a execução de cada secção crítica em exclusão mútua. Para isso, transforme a variável partilhada num objecto partilhado e sincronize o acesso ao objecto através da instrução synchronized.

5 - O **problema do Produtor / Consumidor** consiste em dois processos, o Produtor e o Consumidor, e um bloco de memória (o "Buffer") comum a ambos os processos. O Produtor gera dados que coloca no Buffer, de onde são retirados pelo Consumidor. Os itens de dados têm que ser retirados pela mesma ordem em que foram colocados. A existência do Buffer permite, por exemplo, que variações na velocidade a que os dados são produzidos não tenham reflexo directo na velocidade a que os mesmos são obtidos pelo Consumidor.

- Implemente o problema anterior, considerando que o Buffer tem uma capacidade finita, o que significa que o Produtor é suspenso quando o Buffer está cheio. Analogamente, o Consumidor deverá ser suspenso quando o Buffer está vazio. Considere ainda que os valores armazenados no Buffer são do tipo String.

6 - Usando os mecanismos de sincronização do Java implemente uma classe semáforo, e teste-a no exercício 4.

7 – Suponha que uma sala de cinema pretendia um pequeno programa que lhe permitisse gerir a venda de bilhetes em diferentes postos de venda. A sala tem uma lotação fixa, e a cada bilhete vendido é atribuído um número sequencial, por ordem de aquisição. Pretende-se poder:
- consultar o nome do filme em exibição;

- consultar o número de bilhetes disponíveis para venda;
- vender um bilhete (indicando ao utilizador o número correspondente ao bilhete em venda).

a)Construa uma classe, SalaCinema, que lhe permita realizar estas operações.

b)Para testar o programa começou-se por simular a sua execução concorrente, sendo cada posto de venda uma Thread que acede à classe anterior. Construa a classe que simula o posto de venda, PostoVenda, e uma classe de teste onde é criada a sala de cinema e os 3 postos de venda.

8 – Pretende-se uma aplicação para gerir o dinheiro em caixa de um clube recreativo. Para isso construa as seguintes classes:

a) uma classe contaBancaria que deverá permitir:
- consultar o saldo disponível em cada instante;
- simular um levantamento, cada vez que um utilizador pretenda levantar uma quantia menor ou igual à existente;
- simular um depósito.

b) uma classe *financiador* que periodicamente vai depositando quantias num objecto do tipo contaBancaria.

c) uma classe utilizador que periodicamente vai levantando quantias do objecto do tipo contaBancaria.

d) Para testar as classes anteriores construa uma classe teste em que um objecto do tipo *contaBancaria* seja partilhado concorrentemente por um objecto do tipo *financiador* e por pelo menos 3 objectos do tipo utilizador.

9 – Construa uma classe em JAVA que funcione como uma caixa de correio para comunicação **unidireccional** de valores inteiros entre dois, e apenas dois, processos. Essa caixa de correio deverá ter uma dimensão limitada por um valor à escolha do utilizador. Além da classe anterior elabore um exemplo da sua utilização, definindo duas classes que comuniquem através dessa caixa de correio, e construa ainda uma classe de teste de todas as classes anteriores.

10 - Implemente uma classe, ClassePartilhada, que contenha uma variável, T do tipo inteiro, que poderá ser manipulada por várias Threads. Considere que a variável T em cada instante pode ter ou não um valor atribuído. Para acesso a esta variável devem ser definidos os seguintes métodos:

output (t) – se a variável T não contém qualquer valor então é-lhe atribuído o valor t, caso contrário, o processo que executa o método output é suspenso até que T não possua nenhum valor;
input – devolve o valor da variável T, caso esta contenha uma valor, senão o processo que executa o método input é suspenso;
read – consulta o valor da variável T;
try_input – versão não bloqueante do método input;
try_read – versão não bloqueante do método read;

11 – "Readers-Writers Problem"

a) Construa uma classe em Java, RW, que possua um campo inteiro, XPTO, e dois métodos: ler e escrever. O método ler deve devolver o valor da variável XPTO; o método escrever deve adicionar o valor 100 à variável XPTO e seguidamente subtrair o mesmo valor à variável XPTO.

b) Pretende-se que um objecto da classe RW seja partilhado por vários processos (Threads) de dois tipos:

- processos Leitores – que lêem o valor da variável XPTO usando o método ler;
- processos Escritores – que alteram a variável XPTO usando o método escrever.

- Construa as classes Leitor e Escritor. Cada uma destas classes deve ter uma Thread de execução própria em que, num ciclo infinito, vão respectivamente lendo e alterando valores do objecto partilhado.

c) Construa uma classe de teste que crie um objecto do tipo RW, 3 objectos do tipo Leitor e 2 objectos do tipo Escritor. Estude o comportamento do seu programa

d) Pretende-se que modifique as classes anteriores de forma a que os vários processos Leitores possam executar concorrentemente o método ler, mas que quando um processo Escritor executar o método escrever o faça em exclusão mútua. Isto é, quando um processo está a escrever, nenhum outro pode em simultâneo ler ou escrever a variável XPTO.