

→ Transferência de controlo entre Threads

1- 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.

2 - Usando os mecanismos de sincronização do Java implemente uma classe semáforo, e teste-a no exercício 3 da Folha prática 6.

3 – “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.

4 – 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.

5 – 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*.

7 - 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 um 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*;

8 – Suponha agora que queríamos transformar em aplicações distribuídas “reais” os problemas dos exercícios 4 (sala de cinema) e 5 (caixa do clube recreativo), o que teríamos que fazer?