Coordenação I

6.1 O banquete de javalis

Programar o seguinte problema “produtor/consumidor”. Num banquete de javalis assados existem vários cozinheiros e glutões concorrentes e uma mesa com capacidade limitada (e.g., só cabem na mesa 10 javalis). Cada cozinheiro repetidamente produz um javali assado e coloca-o na mesa. Caso encontre a mesa cheia fica à espera que a mesa fique com espaço disponível para depositar o seu javali. Cada glutão repetidamente retira da mesa um javali e consome-o. Caso encontre a mesa vazia fica à espera que a mesa contenha de novo algum javali. Sincronizar o recurso mesa por forma a coordenar devidamente os vários cozinheiros e glutões. Utilizar ainda mensagens apropriadas para observação do comportamento dos vários atores no banquete. Cada Javali deve ser identificado pelo cozinheiro que o produziu e por um número de ordem.

Versão 1: Considerar que os cozinheiros/glutões produzem/consomem um número fixo de javalis.

Versão 2: Considerar que os cozinheiros/glutões ficam a funcionar um determinado tempo (p.ex. 10s), após o que são interrompidos e param.

6.2 Lingotes de Ouro

Um lingote típico de ouro é construído com ~12.5 kg de ouro. Para fazer um lingote, são precisos diversos pedaços de ouro extraídos da natureza.

Crie uma aplicação que simule uma escavadora de ouro, uma balança e um ourives. A escavadora recolhe pequenos pedaços de ouro (cada pedaço pesa até 1 kg). Quando a escavadora acha o ouro, é posto numa balança. Quando 12.5 kg de ouro ou mais estão na balança, o ourives recolhe o ouro para fazer o lingote.Ambas a escavadora e o ourives devem ser threads. A escavadora pode acrescentar ouro na balança apenas se houver menos do que 12.5 kg de ouro na balança. Por outro lado, o ourives pode apenas recolher o ouro quando ficam 12.5 kg ou mais na balança. Depois de o ourives recolher o ouro da balança, leva-lhe 3 segundos a transformar o ouro num lingote.

Utilize um double para representar a quantidade actual de ouro na balança. Deverá criar uma janela com o campo de texto e um botão. O campo de texto deve mostrar o peso do ouro que está na balança.

Quando o utilizador pressiona o botão “Stop”, a escavadora e o ourives devem ser interrompidos e terminar. Antes de terminar, o ourives deve escrever na consola o número total de lingotes criados.

A sua solução deve enviar mensagens que contenham o id da thread (Thread.currentThread().toString()) bem como o tipo de operação (ex: “inicio do run”) para a consola sempre que uma thread faz:

- Inicio e fim do método run
- Antes de tratar um InteruptedException
- Antes de fazer um join
- Antes e depois de fazer um wait ou um sleep

Exemplo:System.out.println(Thread.currentThread().toString() + “ - inicio do run.”);

6.3 Cadeia de distribuição

Crie uma aplicação que simule um *fornecedor,* um *distribuidor* e 5 *retalhistas.*
O fornecedor fornece sucessivamente produtos ao distribuidor. O distribuidor tem capacidade para armazenar todos os produtos recebidos numa única lista (use uma LinkedList). Quando há 10 ou mais produtos na lista, o distribuidor pode vender os produtos. A venda é feita aos retalhistas que adquirem todos os produtos do distribuidor. Assim, após uma aquisição, a lista de produtos do distribuidor é reinicializada (fica vazia).

Cada produto gerado pelo fornecedor deve ser representado por um objecto do tipo inteiro, com valores aleatórios de zero a nove (classe Integer).

Deverá criar uma janela com um campo de texto e um botão (“stop”). O campo de texto deve mostrar a lista actualizada de produtos existentes no distribuidor. Para escrever os conteúdos da lista no campo de texto basta aplicar o método toString à lista, por exemplo:

```
// assumindo as variáveis previamente declaradas... JTextField textfield = new JTextField();
// escreve-se os conteúdos da lista no campo de texto assim... textfield.setText(lote.toString(
```

Quando o utilizador pressiona o botão “stop”, os retalhistas e o fornecedor devem terminar. No entanto, o fornecedor só deve ser terminado após se ter a certeza que os retalhistas terminaram. Quando um retalhista termina deve escrever o número de lotes adquiridos, por exemplo:

```
Retalhista 0: Comprei um total de 7 lotes. Retalhista 1: Comprei um total de 30 lotes.
Retalhista 2: Comprei um total de 50 lotes. Retalhista 3: Comprei um total de 56 lotes. Reta
```

page1image18200

A sua solução deve enviar mensagens que contenham o id da thread (Thread.currentThread().toString()) bem como o tipo de operação (ex: “inicio do run”) para a consola sempre que uma thread faz:

- Inicio e fim do método run
- Antes de tratar um InteruptedException
- Antes de fazer um join
- Antes e depois de fazer um wait ou um sleep

Exemplo:

```
System.out.println(Thread.currentThread().toString() + “ - inicio do run.”);
```

6.4 Barbeiro

Uma barbearia consiste numa sala de espera com 3 cadeiras e numa sala do barbeiro apenas com a cadeira.

Caso não existam clientes para serem atendidos o barbeiro deve ficar à espera.

Se um cliente entra na barbearia e todas as cadeiras estão ocupadas, então o cliente sai da loja, e regressa após um tempo entre 3 e 10 segundos. Se o barbeiro está ocupado, mas existem cadeiras disponíveis na sala de espera o cliente senta-se numa das cadeiras livres. Se o barbeiro está inactivo, à espera de clientes, o cliente deve notifica-lo da sua presença.

Os clientes devem constantemente cortar o cabelo e depois dormir um tempo aleatório entre 1 e 10 segundos.

Escreva um programa para coordenar o barbeiro e os clientes.

A barbearia deve ainda ter uma janela onde existem 1 campos de texto e um botão. No campo de texto deve indicar o no de cadeiras de espera ocupadas. O botão (“Stop”), quando pressionado deve terminar todas as threads activas. Quando terminado os clientes devem dizer quantas vezes cortaram o cabelo e o barbeiro quantos cortes efectuou.

Teste o seu programa com 10 clientes e um barbeiro.

A sua solução deve enviar mensagens que contenham o id da thread (Thread.currentThread().toString()) bem como o tipo de operação (ex: “inicio do run”) para a consola sempre que uma thread faz:

- Inicio e fim do método run
- Antes de tratar um InteruptedException
- Antes de fazer um join
- Antes e depois de fazer um wait ou um sleep

Exemplo:

```
System.out.println(Thread.currentThread().toString() + “ - inicio do run.”);
```

Solução Banquete

Arquivos anexados: Banquete.zip (7,602 KB)