Guião N.º 3B

Partilha de objectos entre *threads* (em Java)

Sistemas Distribuídos

Ricardo Costa
rcosta@estg.ipp.pt

ESTGF | POLITÉCNICO DO PORTO

Outubro 2017

1 *Thread-safe*

Tal como foi visto no guião anterior, todas as *threads* de um mesmo processo partilham o contexto desse processo, i.e., as várias *threads* partilham o mesmo segmento de memória. Tal significa que que todos os objectos instanciados pelo processo estão directamente acessíveis às várias *threads*.

A partilha de objectos entre *threads* surge como uma grande mais valia mas implica que o programador proteja a utilização de todos os objectos partilhados. Todos os recursos partilhados devem ter alguma forma de condicionamento de acesso que possibilite a sua utilização partilhada sem que daí advenham problemas relativos à concorrência no acesso aos objectos partilhados, nomeadamente, acessos de escrita simultâneos. Quando um objecto (ou recurso) está preparado para ser utilizado em contexto multi-*thread* de forma segura diz ser um objecto *thread-safe*.

1.1 Partilha de objectos (memória) entre *threads* em Java

Uma forma simples de se partilhar um objecto entre *threads* é por recurso ao construtor da classe da "nossa" *thread*. A Listagem 1 demonstra esta

abordagem.

```
1  public class Worker extends Thread{
2      ArrayList frases;
3      int numero;
4
5      public Worker(ArrayList f,int n) {
6          super("Worker");
7          this.frases=f;
8          this.numero=n;
9      }
10
11     public void run() {
12         for(int i=0;i<5;i++) {
13             try {
14                 frases.add("Frase "+i+" da thread "+numero);
15                 Thread.sleep(500+i*10);
16             } catch (Exception ex) {}
17         }
18     }
19 }
```

Listing 1: Classe Worker

O construtor da classe *Worker* (linha 5) recebe dois argumentos. Realça-se em particular o parametro *f* (objecto do tipo ArrayList) que será um objecto partilhado por todas as *threads* para armazenarem centralmente uma lista de frases (linha 14). A classe *ObjSharing* (ver Listagem 2) utiliza a classe *Worker*.

```
1  public class ObjSharing {
2      public static void main(String[] args) {
3          int NThreads=5;
4          ArrayList asFrases=new ArrayList();
5
6          for (int i=0;i<NThreads;i++)
7              new Worker(asFrases,i).start();
8
9          for (int j=0;j<6;j++) {
10             try {
11                 Thread.sleep(1000);
12             } catch (Exception ex) { }
13             for(int k=0;k<asFrases.size();k++)
14                 System.out.println(asFrases.get(k));
15         }
16     }
17 }
```

Listing 2: Classe ObjSharing

1.2 Acesso concorrente a objectos partilhados

A utilização em simultâneo de um objecto por várias *threads* pode levar a que a informação, guardada no objecto partilhado, fique num estado incongruênte. Para evitar que tal aconteça, existem na linguagem Java vários mecanismos para o efeito. Em particular a utilização de métodos sincronizados (ou em inglês, *synchronized methods*) é exemplificada de seguida.

```
1  public class SynchronizedArrayList {
2      private ArrayList list=new ArrayList();
3
4      public synchronized void add(Object o) {
5          list.add(o);
6      }
7
8      public synchronized ArrayList get() {
9          return list;
10     }
11 }
```

Listing 3: Exemplo de métodos sincronizados em Java

A Listagem 3 apresenta a classe *SynchronizedArrayList* que poderá ser utilizada em subsituição do objecto asFrases da classe *ObjSharing*. A motivação para esta subsituição assenta em dois factores:

1. A classe *ArrayList* não é *thread-safe*, pelo que não implementa nenhum mecanismo de sincronização de acesso.

2. A classe *SynchronizedArrayList* implementa os métodos necessários (adição e consulta) pela classe *ObjSharing* em modo sincronizado (ver linhas 4 e 8), garantindo a sua utilização segura em ambientes multi-*threaded*. Tal é conseguido pela simples adicção da palavra *synchronized* na declaração dos métodos, encarregando-se o Java de assegurar a sua utilização sincronizada.

Realça-se contudo que existem várias soluções alternativas para conseguir este propósito. Por exemplo, usando como forma de armazenar a informação outra classe já disponível que seja *thread-safe* (e.g.: classe Vector). Existem ainda outros mecanismos que poderão ser adoptados (ver Secção 3).

2 Exercícios

1. Teste as classes *Worker* e *ObjSharing*, tal como apresentadas.

2. Teste as classes *Worker* e *ObjSharing*, removendo os períodos de espera (*sleeps*).

3. Faça com que a sua solução para exercício da alínea anterior seja *thread-safe*.

4. Retome o exercício de *chat* em rede do Guião anterior. Desenvolva, no mínimo, duas classes: Servidor e ProcessaCliente.

   A classe **Servidor** deve ser agora *multi-threaded*, com duas *threads* com o seguinte comportamento:

   (a) Aguardar por pedidos de ligação de clientes, instanciando um nova *thread* para cada um.

   (b) Periodicamente, envia todas as mensagens recebidas de todos os clientes, para todos os clientes.

   A classe **ProcessaCliente** recebe mensagens do seu cliente, adicionando-lhes a data e hora, bem como o IP do cliente. Devendo de seguida, armazenar a mensagem centralmente.

   Teste o Servidor recorrendo ao programa *Telnet*.

3 Recursos *online*

- The Java Tutorial, Synchronization
  `http://download.oracle.com/javase/tutorial/essential/concurrency/sync.html`