Matéria:

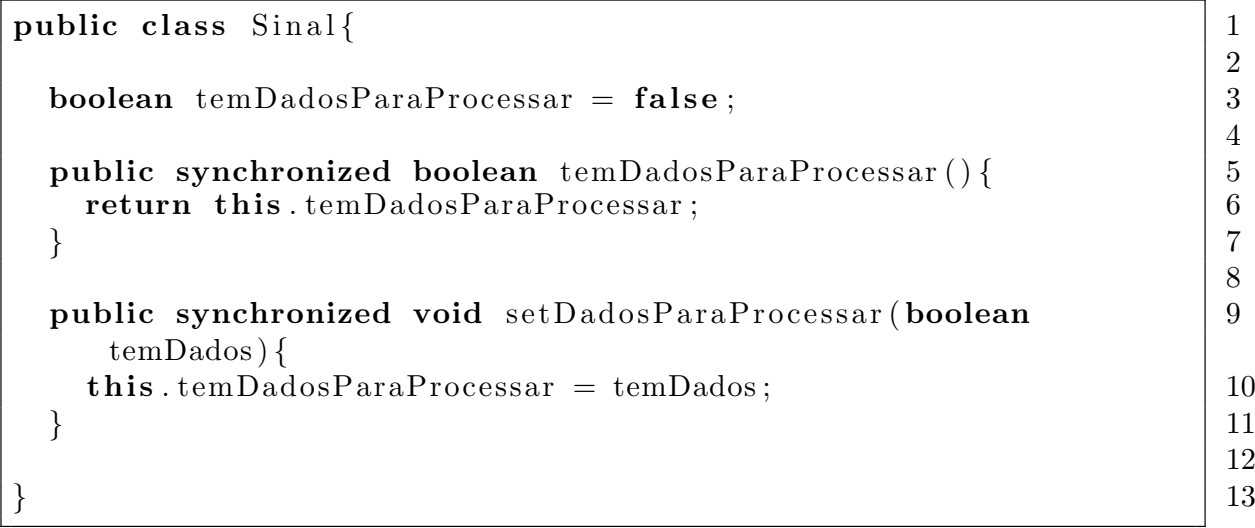
* Sinalização entre *thread* em Java

1. **– Sinalização entre *thread***

A sinalização entre *thread* surge como mais uma forma que a linguagem Java tem de permitir a sincronização da execução de múltiplas *thread* em simultâneo. A versão mais simples, que resulta de implementação de um mecanismo destes pelo próprio programador, solução vulgarmente conhecida como espera ativa.

# – Espera ativa

Uma forma de se implementar um mecanismo de sinalização entre *thread* em Java é por intermédio de um objeto partilhado. A classe Sinal, apresentada na Listagem 1, pode ser utilizada como objeto partilhado para se implementar uma espera ativa.



Listagem 1: Classe *Sinal*

Contudo, a utilização de sinalização deste tipo não é recomendada já que esta ocupará toda a capacidade de processamento (CPU) disponível no momento da sua execução. Uma solução preferível seria uma que permitisse que a *thread* em espera, adormecesse.

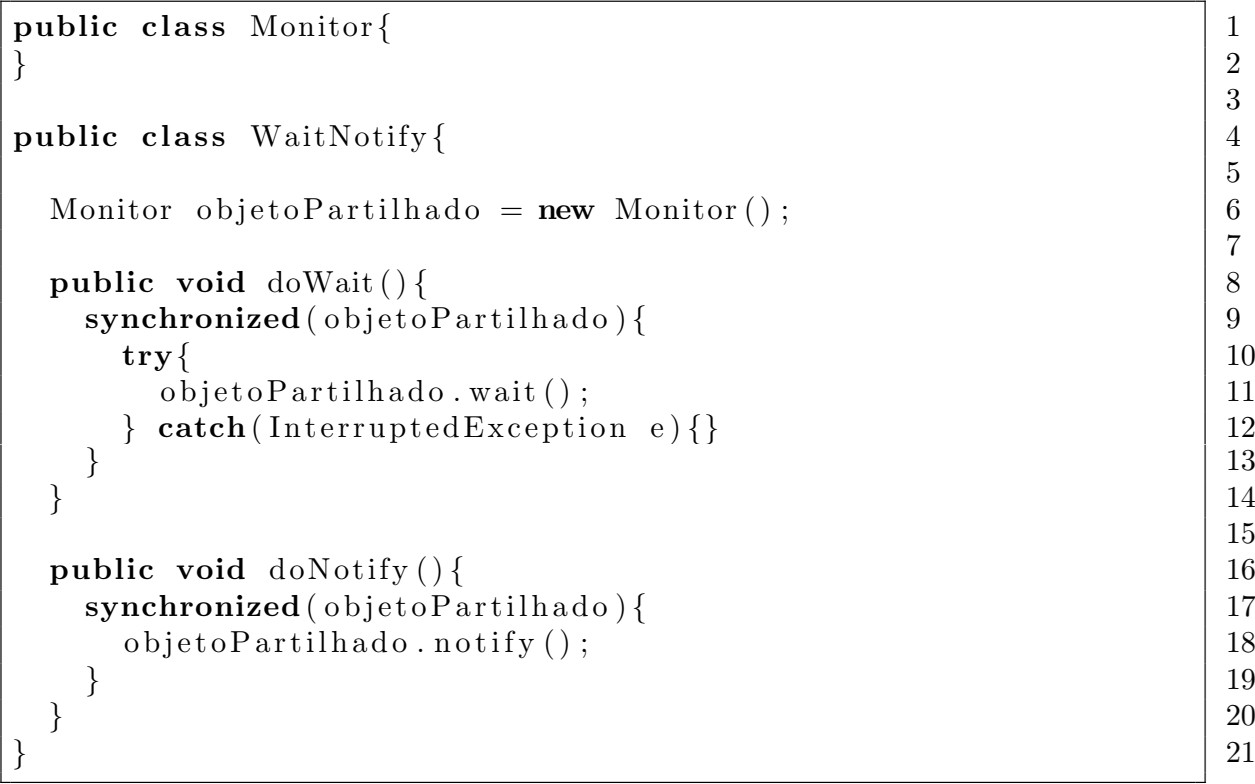
* 1. **– Métodos *wait()*, *notify()* e *notifyAll()***

A linguagem Java, na sua classe *java.lang.Object*, dispõe destes três métodos que possibilitam o adormecer de uma, ou mais, *thread*. Assim, uma *thread* que invoque o método *wait()* de um qualquer objeto, fica suspensa até que uma outra *thread* invoque o método *notify()* desse mesmo objeto. O método *notifyAll()* funciona da mesma forma que o

método *notify()* só que, em vez de despertar a próxima *thread*, desperta todas as *thread*

suspensas no mesmo objeto.

De notar que a thread que queria invocar qualquer um destes métodos, deve primeiramente obter o acesso exclusivo ao objeto. Ou seja, os métodos *wait()*, *notify()* e *notifyAll()* **devem ser invocados de dentro de um bloco** sincronizado[1](#_bookmark0) (synchronized).

A Listagem 2 mostra como se pode utilizar estes métodos para construir uma classe que pode ser utilizada com forma de sinalização. Assim, tanto o método *doWait()*, como o método *doNotify*, executam dentro de blocos sincronizados referentes a um mesmo objeto Monitor.

Listagem 2: Classe *WaitNotify*

# – Sinalização perdida

Quanto se usam os métodos de sinalização descritos na seção anterior é necessário ter em atenção que estes métodos não guardam informação de estado, ou histórico. Tal poderá levantar alguns problemas se este comportamento não for tido em consideração no desenvolvimento das aplicações. A situação mais comum surge quando, de duas *thread* T1 e T2, por exemplo, a *thread* T1 invoca o método *notify()* antes da T2 ter chamado o método wait(). Tal resulta na perda da invocação feita por T1, já que naquele momento não existirá nenhuma outra *thread* suspensa. Uma solução simples passa por adaptar as classes utilizadas para sinalização, como a apresentada na Listagem 2, para guardar um histórico de invocações.

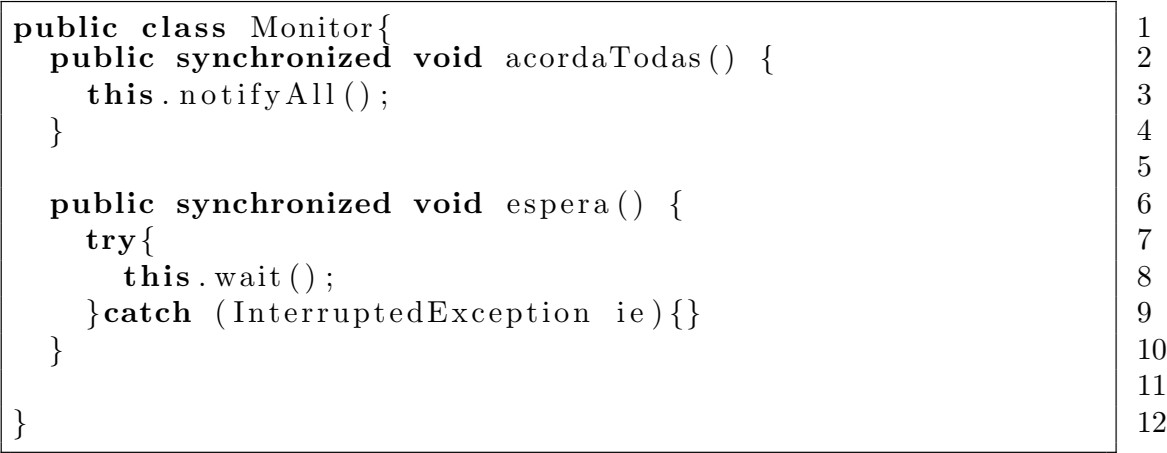
1 Quando executados fora de um bloco sincronizado, a JVM lança uma exceção *IllegalMonitorStateException.*

# – Falsos despertares

Não tão pouco frequentemente, a JVM ativa *thread* suspensas numa invocação *wait()* sem razão aparente. A estas situações dá-se o nome de *falsos despertares*. A solução para estas situações passa pela adição de variáveis de sinalização (*flags*) booleanas à classe utilizada para sincronização.

# 2– Exercícios

* + 1. Considerando um objeto partilhado como Monitor apresentado na Listagem 3.



Listagem 3: Classe *WaitNotify*

Apresente uma versão alterada da classe **Janela** (Listagem 4) para que esta crie oito *thread*, cada uma com a sua janela. Muito embora as janelas sejam criadas sequencialmente, espera-se que estas apareçam todas de imediato, mas que apenas iniciem a atualização da sua *label* cinco segundos após a sua criação. Todas as *thread* devem iniciar a atualização das respetivas *labels* em simultâneo.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Listagem 4: Classe *Janela*

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

* + 1. Crie uma nova versão da classe **Janela** que, em vez de acordar todas as *thread*

em simultâneo, as acorde com desfasamento de um segundo entre cada *thread*.;

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

* + 1. Crie uma terceira versão da classe **Janela** que garanta que as várias *thread* são acordadas por ordem (Th1, Th2, ...), mantendo o desfasamento de um segundo entre elas. Poderá também ser necessário efetuar alterações à classe Monitor;

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

* + 1. Escreva um programa que lança uma (segundo) *thread*. Ao *thread* principal (ou produtor) compete escrever um carácter num *buffer*, de 100 em 100 milissegundos, até ao máximo de 10 carateres. Ao segundo *thread* (ou consumidor) compete ler o caráter armazenado no buffer. O buffer só deve ter capacidade para armazenar um único caráter. Tal implica que a *thread* principal só deve escrever no buffer quando este estiver vazio e que a segunda *thread* só deve ler do *buffer* quando lá existe um caráter.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente