01

Revisitando a nossa lista

Transcrição

Nesse capítulo vamos ver mais um exemplo de `wait - notify`, para consolidar o aprendizado. De certa forma, esse capítulo é uma revisão do capítulo anterior.

Lembrando, `wait` e `notify` devem ser chamados dentro de um bloco `synchronized`. Um thread fica na espera e devolve a chave. Outra thread notifica e faz com que a primeira thread continue executando. Os dois métodos são da classe `Object` e servem para que duas threads se comuniquem independentemente.

Exemplo de lista

Já falamos sobre o problema, quando dois threads começam a manipular uma lista. Vimos como resolver isso - sincronizando o acesso. Vimos também que a classe `Vector` é *thread safe* e a classe `ArrayList` não. Vamos continuar com esse exemplo para utilizar `wait` e `notify`.

O primeiro passo é voltar a utilizar a implementação da nossa lista, pois queremos alterá-la.

Por exemplo, a nossa `TarefaAdicionarElemento` vai voltar a usar a `Lista` invés da `java.util.List`:

```
public class TarefaAdicionarElemento implements Runnable {

    private Lista lista;
    private int numeroDoThread;

    public TarefaAdicionarElemento(Lista lista, int numeroDoThread) {
        this.lista = lista;
        this.numeroDoThread = numeroDoThread;
    }

    @Override
    public void run() {

        for (int i = 0; i < 100; i++){
            lista.adicionarElementos("Thread " + numeroDoThread + " - " +  i);
        }
    }

}
```

A nossa motivação para este capítulo é imprimir os elementos da lista através de um novo thread. Então, vamos apagar o `for` que realiza isso e criar um thread com a tarefa. A classe `Principal` ficará assim:

```
public class Principal {

    public static void main(String[] args) throws InterruptedException {

        Lista lista = new Lista();
```

```
        for (int i = 0; i < 10; i++) {
            new Thread(new TarefaAdicionarElemento(lista, i)).start();
        }

        new Thread(new TarefaImprimir(lista)).start();
    }
}
```

A implementação da tarefa não tem segredo, ela imprime todos os elementos da lista:

```
public class TarefaImprimir implements Runnable {

private Lista lista;

    public TarefaImprimir(Lista lista) {
        this.lista = lista;
    }

    @Override
    public void run() {
        for (int i = 0; i < lista.tamanho(); i++) {
            System.out.println(i + " - " + lista.pegaElemento(i));
        }
    }
}
```

Ao rodar a classe `Principal`, tudo parece estar funcionando! Para realmente conseguir ver o problema, vamos simular uma adição mais demorada. Novamente usamos um método `Thread.sleep(..)` para atrasar a execução. Na classe `Lista`:

```
public class Lista {

    //atributos omitidos

    public syncronized void adicionaElementos(String elemento) {
        this.elementos[indice] = elemento;
        this.indice++;

        try{
            Thread.sleep(10);
        } catch(InterruptedException e) {
            e.printStackTrace();
        }
    }

    //outros métodos omitidos
}
```

Vamos testar de novo e rodar a classe `Principal` e agora sim aparecem um monte de `null` na lista:

```
0 - Thread 0 - 0
1 - null
```

```
2 - null
3 - null
4 - null
….
```

O problema é que o nosso thread que imprime a lista foi tão rápido que não deu tempo para preencher tudo! Idealmente o thread que imprime deve esperar até que a lista realmente esteja preenchida! E nós já sabemos implementar isso!

Wait e notify na lista

Vamos começar com `notify`. Quando a lista chegar no último elemento, vamos notificar o outro thread:

```
public class Lista {

    //atributos omitidos

    public syncronized void adicionaElementos(String elemento) {
        this.elementos[indice] = elemento;
        this.indice++;

        try{
            Thread.sleep(10);
        } catch(InterruptedException e) {
            e.printStackTrace();
        }

        if (this.indice == this.tamanho()) {
            System.out.println("lista tá cheia, notificando");
            this.notify();
        }
    }

    //outros métodos omitidos
}
```

Agora falta o outro thread **esperar**. Ele só deve imprimir os elementos quando for notificado.

```
public class TarefaImprimir implements Runnable {

//atributo e construtor omitido

    @Override
    public void run() {

        try {
            System.out.println("esperando, aguardando notificacao");
            lista.wait();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        for (int i = 0; i < lista.tamanho(); i++) {
            System.out.println(i + " - " + lista.pegaElemento(i));
```

```
        }
    }
}
```

Vale o teste e rodar a nossa classe `Principal`, no entanto recebemos uma exceção:

```
Exception in thread "Thread-10" java.lang.IllegalMonitorStateException
    at java.lang.Object.wait(Native Method)
    at java.lang.Object.wait(Object.java:502)
    at br.com.alura.lista.TarefaImprimir.run(TarefaImprimir.java:15)
    at java.lang.Thread.run(Thread.java:745)
```

Essa exceção aconteceu pois devemos chamar o método `wait()` dentro de um bloco `synchronized`. Isso também é valido para o método `notify()`.

Sabemos que o `synchronized` precisa pegar uma chave/mutex de um objeto! A pergunta é de qual objeto devemos pegar a chave? Repare que `this` seria o objeto `TarefaImprimir`, que não interessa à gente, pois não é acessado em paralelo. O objeto sincronizado é a lista, e esse será a nossa chave. Uma vez a chave obtida pelo `synchronized`, podemos também chamar o método `wait()`, usando a mesma lista!

Veja como fica o código:

```
public class TarefaImprimir implements Runnable {

//atributo e construtor omitido

    @Override
    public void run() {
    synchronized (lista) { //obtendo a chave da lista
            try {
                System.out.println("esperando, aguardando notificacao");
                lista.wait(); //devolvendo a chave e esperando
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            for (int i = 0; i < lista.tamanho(); i++) {
                System.out.println(i + " - " + lista.pegaElemento(i));
            }
        }
    }
}
```

Ao executar, pode ser que o thread para imprimir a lista fique esperando primeiro, para ser notificado depois:

```
esperando, aguardando notificacao
lista tá cheia, notificando
0 - Thread 0 - 0
1 - Thread 9 - 0
2 - Thread 7 - 1
3 - Thread 5 - 1
….
```