02

Buscando em arquivo texto

Transcrição

Nesse capítulo vamos mostrar mais um exemplo para trabalhar com threads. O nosso objetivo é fazer uma busca em vários arquivos de texto, ao mesmo tempo, claro!

Para o exemplo seguinte: No mundo de desenvolvimento de software ágil existe um manifesto que descreve os princípios do desenvolvimento ágil (que ficou muito famoso!). Eu peguei uma lista com os nomes dos autores desse manifesto. Além disso, peguei também mais dois arquivos com os nomes das pessoas que assinaram esse manifesto. Para quem gostaria de saber mais sobre essa manifesto ágil, pode acessar o site:

http://agilemanifesto.org/ (http://agilemanifesto.org/)

Você pode baixar esses três arquivos aqui (https://s3.amazonaws.com/caelum-online-public/threads1/nomes.zip)

Vamos criar um novo projeto no Eclipse e chamar essa projeto `busca-textual`. Vamos colocar esses 3 arquivos vamos colocar na raiz do projeto `busca-textual`.

Vamos começar a criar as classes para realizar a busca. A primeira classe vai inicializar os nossos threads. Vamos chamar essa de `Principal`, no pacote `br.com.alura.threads`:

```
public class Principal {

    public static void main(String[] args) {
    }
}
```

Nessa classe, vamos declarar o nome que queremos procurar:

```
public static void main(String[] args) {

    String nomeProcurado = "Jon";
}
```

Pesquisando paralelamente

Além disso, vamos criar para cada arquivo um novo `Thread`. Cada Thread receberá no construtor a tarefa para buscar o nome no arquivo. A tarefa, por sua vez, precisa do nome do arquivo e a String com o nome procurado:

```
public static void main(String[] args) {

    String nomeProcurado = "Jon";

    Thread threadAutores = new Thread(new TarefaBuscaNome("autores.txt",
            nomeProcurado));
    Thread threadAssinaturas1 = new Thread(new TarefaBuscaNome(
            "assinaturas1.txt", nomeProcurado));
```

```
        Thread threadAssinaturas2 = new Thread(new TarefaBuscaNome(
                "assinaturas2.txt", nomeProcurado));

    }
```

O nosso código não compila ainda pois é preciso criar a classe `TarefaBuscaNome`. Vamos utilizar o Eclipse para criar classe, basta digitar **CTRL + 1** e pedir para ele gerar automaticamente a classe:

```
public class TarefaBuscaNome implements Runnable {

@Override
    public void run() {

    }
}
```

Repare que o Eclipse automaticamente sugere a interface `Runnable`. Também já vamos criar os atributos e gerar o construtor:

```
public class TarefaBuscaNome implements Runnable {

    private String nomeArquivo;
    private String nome;

    public TarefaBuscaNome(String nomeArquivo, String nome) {
        this.nomeArquivo = nomeArquivo;
        this.nome = nome;
    }

    @Override
    public void run() {

    }
}
```

Ainda falta implementar o método `run`. Nesse método vamos utilizar a classe `Scanner`. A classe recebe no construtor um `File`. Depois disso podemos iterar em cima de cada linha verificando no laço se a linha possui o nome procurado:

```
@Override
public void run() {

    try {
        Scanner scanner = new Scanner(new File(nomeArquivo));

        int numeroLinha = 1;

        while (scanner.hasNextLine()) {

            String linha = scanner.nextLine();

            if (linha.contains(nome)) {
                System.out.println(nomeArquivo + " - " + numeroLinha
```

```
                        + " - " + linha);
                }

                numeroLinha++;

            }

            scanner.close();

        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        }

    }
```

Repare que imprimimos o nome do arquivo, o número da linha e a linha, caso encontrarmos o nome no arquivo.

Agora tudo deveria compilar, no entanto falta ainda inicializar os nossos threads. Isso também já sabemos, basta chamar o método `start()` para cada thread. Voltando para a classe `Principal` :

```
public class Principal {

    public static void main(String[] args) {

        String nomeProcurado = "Jon";

        Thread threadAutores = new Thread(new TarefaBuscaNome("autores.txt",
                nomeProcurado));
        Thread threadAssinaturas1 = new Thread(new TarefaBuscaNome(
                "assinaturas1.txt", nomeProcurado));
        Thread threadAssinaturas2 = new Thread(new TarefaBuscaNome(
                "assinaturas2.txt", nomeProcurado));

        threadAutores.start();
        threadAssinaturas1.start();
        threadAssinaturas2.start();
}
}
```

Vamos testar a nossa busca e executar a classe `Principal` . Os resultados deveriam aparecer no console. Já conseguimos executar uma busca em paralelo! Ótimo.

Qual é a ordem?

Até agora houveram poucas novidades nesse capítulo, apenas revisamos os conceitos, usando um exemplo mais real.

Agora faça o seguinte teste: execute algumas vezes a nossa classe `Principal` , sempre verificando os resultados no console. Repare que no meu caso o primeiro resultado mudou. Ou seja, o que os nossos threads realmente acham como o primeiro, segundo ou terceiro elemento varia, não há uma ordem de execução! Não sabemos qual thread é executado primeiro!

Isso é uma característica dos threads e devemos estar conscientes disso. Usando threads, perdemos o controle da ordem de execução! Veremos ainda algumas formas para manipular a ordem, mas em geral, quando o thread realmente é

executado depende da máquina virtual e do sistema operacional. O desenvolvedor tem pouco controle sobre isso! Vamos testar isso nos exercícios desse capítulo. Vamos começar?