#7

Mãos à obra: Distribuindo comandos

Para distribuir os comandos siga os seguintes passos:

1) Cada comando passará a ter uma classe dedicada que implementará a interface `Runnable`. Então crie as classes `ComandoC1` e `ComandoC2`. Segue o exemplo da classe `ComandoC1`:

```
//dentro do pacote br.com.caelum.servidor
public class ComandoC1 implements Runnable {

    @Override
    public void run() {
        // vem mais aqui
    }
}
```

A classe `ComandoC2` fica igual a classe `ComandoC1` (tirando o nome, claro).

2) Adicione nas classes `ComandoC1` E `ComandoC2` um novo atributo do tipo `PrintStream`. Implemente o construtor nas classes:

```
public class ComandoC1 implements Runnable {

    private PrintStream saida; //novo, atributo representa a saida do

    public ComandoC1(PrintStream saida) { //novo construtor
        this.saida = saida;
    }

    //método run() omitido
}
```

3) Use o `PrintStream` no método `run()` de cada classe "Comando" para imprimir uma mensagem ao cliente sobre a execução do comando:

```
//na classe ComandoC1
@Override
public void run() {
    //será impresso no console do servidor
    System.out.println("Executando comando c1");

    //essa mensagem será enviada para cliente
        saida.println("Comando c1 executado com sucesso!");
}
```

Faça o mesmo na classe `ComandoC2`, só alterando a mensagem para `c2`!

4) Para simular que os comandos executam algo demorado (por exemplo: acesso ao banco, geração de relatório, chamadas de web services, etc), use o método `Thread.sleep(20000)` dentro do `run()` do comando (lembrando que o método `sleep()` precisa de um tratamento de exceção):

```
//na classe ComandoC1
@Override
public void run() {
    //será impresso no console do servidor
    System.out.println("Executando comando c1");

    try {
        Thread.sleep(20000);
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }

    //essa mensagem será enviada para o cliente
    this.saida.println("Comando c1 executado com sucesso!");
}
```

Use `Thread.sleep(5000);` no método `run()` da classe `ComandoC2`. Repare que `ComandoC2` demora apenas 5 segundos.

5) Os comandos estão prontos, mas para poder executá-los devemos inicializar novas threads. Para tal, abra a classe `ServidorTarefas` e procure a linha que instancia a classe `DistribuirTarefas` (está dentro do método `rodar()`). Nela adicione o `threadPool` no construtor:

```
DistribuirTarefas distribuirTarefas = new DistribuirTarefas(threadPool
```

6) O código para de compilar pois o construtor da classe `DistribuirTarefas` está errado agora. Para arrumar, abra a classe `DistribuirTarefas` e adicione um novo atributo `threadPool` e receba-o no construtor da classe:

```
public class DistribuirTarefas implements Runnable {

    private Socket socket;
    private ServidorTarefas servidor;
    private ExecutorService threadPool; //novo atributo

    //recebendo o threadPool
    public DistribuirTarefas(ExecutorService threadPool, Socket socket
        this.threadPool = threadPool; //nova atribuicao
        this.socket = socket;
        this.servidor = servidor;
    }

    //método run() omitido
}
```

7) Com o pool em mãos podemos finalmente "rodar" os comandos. Ainda na classe `DistribuirTarefas`, dentro do método `run()`, crie para cada comando uma nova instância e passe a passe para o pool através do método `execute()`. Isso deve ser feito dentro do `case` do comando:

```
//na classe DistribuirTarefas, dentro do método rodar(), dentro do swi
case "c1": {
    saidaCliente.println("Confirmação do comando c1");

    //novo
    ComandoC1 c1 = new ComandoC1(saidaCliente); //criando comando
    this.threadPool.execute(c1); //executando comando pelo pool

    break;
}
```

E para o comando c2:

```
case "c2": {
    saidaCliente.println("Confirmação do comando c2");

    //novo
    ComandoC2 c2 = new ComandoC2(saidaCliente);
    this.threadPool.execute(c2);

    break;
}
```

8) Tudo deve estar compilando. Para testar, primeiro *verifique se não há nenhum servidor ou cliente rodando*. Depois rode a classe `ServidorTarefas` e a classe `ClienteTarefas`.

9) Tente enviar comando pelo console do cliente, como `c1` ou `c2`. Lembrando que os comandos estão sendo executados em threads dedicadas, ou seja, em paralelo, demorando 20s para o comando `c1` e 5s para o comando `c2`: