10

Mãos à obra: Usando Callable

Chegou a hora de implementar o que o instrutor fez neste capítulo para que no final você tenha um projeto completo igual ao dele. Para manter a sensação de desafio, serão fornecidos para você passos gerais que devem ser implementados. Sinta-se livre para consultar o vídeo e a explicação textual sempre que desejar.

Seguem os passos gerais:

1 - Crie as classes `ComandoC2ChamaWS` e `ComandoC2AcessaBanco`, que devem implementar `Callable<String>`.

```
public class ComandoC2ChamaWS implements Callable<String> {

    @Override
    public String call() throws Exception {
        return null;
    }
}
```

Igualmente na outra tarefa: `ComandoC2ChamaWS`

2) Agora faça a implementação de cada comando. Primeiro devemos receber a saída do cliente no construtor; por exemplo, na classe `ComandoC2ChamaWS`:

```
private PrintStream saida;

public ComandoC2ChamaWS(PrintStream saida) {
          this.saida = saida;
}
```

Igualmente na classe `ComandoC2ChamaWS`.

3) No método `call()`, vamos devolver uma mensagem para o cliente, esperar por 15s (*sleep*) e gerar um número aleatório como resultado. Novamente, o exemplo da classe `ComandoC2ChamaWS`:

```
@Override
public String call() throws Exception {
    System.out.println("Servidor recebeu comando c2 - WS");
    saida.println("Processando comando c2 - WS");

    Thread.sleep(15000);

    int numero = new Random().nextInt(100) + 1;
    System.out.println("Servidor finalizou comando c2 - WS");
    return Integer.toString(numero);
}
```

O mesmo será feito no `ComandoC2AcessaBanco`, mas não se esqueça de alterar as mensagens.

4) Agora precisamos adicionar uma instância de cada comando no pool de thread que já temos. Para tal, abra a classe `DistribuirTarefa` e procure dentro do `switch` o `case "c2"`. Nesse `case` crie uma instância do `ComandoC2ChamaWS` e outra do `ComandoC2AcessaBanco`:

```
ComandoC2ChamaWS c2WS = new ComandoC2ChamaWS(saidaCliente);
ComandoC2AcessaBanco c2Banco = new ComandoC2AcessaBanco(saidaCliente);
```

Observação: Não usaremos mais o comando `ComandoC2`, pode apagar.

5) Ainda nesse `case`, submeta cada comando ao nosso `pool`, recebendo como retorno um `Future`:

```
Future<String> futureWS = this.threadPool.submit(c2WS);
Future<String> futureBanco = this.threadPool.submit(c2Banco);
```

6) Verifique se tudo está compilando. Se estiver ok, pode rodar o servidor e cliente (parando tudo antes, claro). Ao submeter o comando `c2` já estamos executando os comandos `ComandoC2ChamaWS` e `ComandoC2AcessaBanco` mas *ainda não juntamos os resultados*. Faremos isso, no próximo exercício.