Disciplina: Nível 5 - Por Que Não Paralelizar
Semestre: 1º - 2024
Aluno: DAVIDSON PEREIRA DE OLIVEIRA
Matrícula: 2022.11.54585-5
Repositório no GIT: dev-davidson/RPG0018-Por-que-n-o-paralelizar (github.com)

Relatório discente de acompanhamento

👉 1º Procedimento | Criando o Servidor e Cliente de Teste

Códigos:

Servidor (CadastroServer):

```
import java.io.*;
import java.net.*;
import controller.MovimentoJpaController;
import controller.PessoaJpaController;
import model.Movimento;

public class CadastroThreadV2 extends Thread {
    private MovimentoJpaController ctrlMov;
    private PessoaJpaController ctrlPessoa;
    private Socket s1;

    public CadastroThreadV2(MovimentoJpaController ctrlMov, PessoaJpaController ctrlPessoa, Socket s1) {
        this.ctrlMov = ctrlMov;
        this.ctrlPessoa = ctrlPessoa;
        this.s1 = s1;
    }

    public void run() {
        try {
            ObjectOutputStream out = new ObjectOutputStream(s1.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(s1.getInputStream());


            String comando = (String) in.readObject();


            if (comando.equals("E") || comando.equals("S")) {
                String tipo = comando;
```

```
                String idPessoa = (String) in.readObject();
                String idProduto = (String) in.readObject();
                int quantidade = (int) in.readObject();
                double valorUnitario = (double) in.readObject();


                Movimento movimento = new Movimento();
                movimento.setTipo(tipo);
                movimento.setPessoa(ctrlPessoa.findPessoa(idPessoa));
                movimento.setProduto(ctrlProduto.findProduto(idProduto));
                movimento.setQuantidade(quantidade);
                movimento.setValorUnitario(valorUnitario);


                ctrlMov.create(movimento);


                if (tipo.equals("E")) {

                } else {

                }
            }
        } catch (IOException | ClassNotFoundException ex) {
            ex.printStackTrace();
        }
    }
}
```

```
// Main.java
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import controller.MovimentoJpaController;
import controller.PessoaJpaController;

public class Main {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("CadastroServerPU");
        MovimentoJpaController ctrlMov = new MovimentoJpaController(emf);
        PessoaJpaController ctrlPessoa = new PessoaJpaController(emf);

        try (ServerSocket ss = new ServerSocket(4321)) {
            while (true) {
                Socket s = ss.accept();
                CadastroThreadV2 ct = new CadastroThreadV2(ctrlMov, ctrlPessoa, s);
                ct.start();
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Cliente (CadastroClient):

```
import java.io.*;
import java.net.*;
import controller.MovimentoJpaController;
import controller.PessoaJpaController;
import model.Movimento;

public class CadastroThreadV2 extends Thread {
    private MovimentoJpaController ctrlMov;
    private PessoaJpaController ctrlPessoa;
    private Socket s1;

    public CadastroThreadV2(MovimentoJpaController ctrlMov, PessoaJpaController ctrlPessoa, Socket s1) {
        this.ctrlMov = ctrlMov;
        this.ctrlPessoa = ctrlPessoa;
        this.s1 = s1;
    }

    public void run() {
        try {
            ObjectOutputStream out = new ObjectOutputStream(s1.getOutputStream());
            ObjectInputStream in = new ObjectInputStream(s1.getInputStream());


            String comando = (String) in.readObject();


            if (comando.equals("E") || comando.equals("S")) {
                String tipo = comando;
```

```
                String idPessoa = (String) in.readObject();
                String idProduto = (String) in.readObject();
                int quantidade = (int) in.readObject();
                double valorUnitario = (double) in.readObject();


                Movimento movimento = new Movimento();
                movimento.setTipo(tipo);
                movimento.setPessoa(ctrlPessoa.findPessoa(idPessoa));
                movimento.setProduto(ctrlProduto.findProduto(idProduto));
                movimento.setQuantidade(quantidade);
                movimento.setValorUnitario(valorUnitario);


                ctrlMov.create(movimento);


                if (tipo.equals("E")) {

                } else {

                }
            }
        } catch (IOException | ClassNotFoundException ex) {
            ex.printStackTrace();
        }
    }
}
```

```
// Main.java
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import controller.MovimentoJpaController;
import controller.PessoaJpaController;

public class Main {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory("CadastroServerPU");
        MovimentoJpaController ctrlMov = new MovimentoJpaController(emf);
        PessoaJpaController ctrlPessoa = new PessoaJpaController(emf);

        try (ServerSocket ss = new ServerSocket(4321)) {
            while (true) {
                Socket s = ss.accept();
                CadastroThreadV2 ct = new CadastroThreadV2(ctrlMov, ctrlPessoa, s);
                ct.start();
            }
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }
}
```

Análise e Conclusão:

a) Como as Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor?

As Threads podem ser utilizadas para lidar com as respostas assíncronas do servidor ao cliente. Em um ambiente de rede, é comum que o cliente precise aguardar a resposta do servidor após enviar uma solicitação. Usando Threads, o cliente pode continuar a funcionar sem ficar bloqueado esperando pela resposta. Assim, uma Thread pode ser dedicada a aguardar e processar as respostas do servidor enquanto outras Threads do cliente continuam a executar outras tarefas.

b) Para que serve o método invokeLater, da classe SwingUtilities?

O método invokeLater da classe SwingUtilities é usado para garantir que determinado código seja executado na thread de despacho de eventos do Swing, também conhecida como a Event Dispatch Thread (EDT). No contexto de interfaces gráficas Swing, todas as interações com os componentes da interface devem ser feitas na EDT para garantir a consistência e a segurança da interface gráfica.

c) Como os objetos são enviados e recebidos pelo Socket Java?

Para enviar e receber objetos pelo Socket Java, é necessário usar fluxos de entrada e saída de dados (InputStream e OutputStream) em conjunto com ObjectInputStream e ObjectOutputStream. O processo envolve a serialização dos objetos antes de serem enviados pelo Socket e a desserialização dos objetos recebidos no lado receptor.

d) Compare a utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java, ressaltando as características relacionadas ao bloqueio do processamento.

Comportamento Síncrono: No comportamento síncrono, o cliente envia uma solicitação ao servidor e aguarda bloqueado até receber uma resposta. Durante esse tempo de espera, o cliente não pode realizar outras tarefas.
Comportamento Assíncrono: No comportamento assíncrono, o cliente envia uma solicitação ao servidor, mas não espera bloqueado por uma resposta. Enquanto aguarda pela resposta do servidor, o cliente pode continuar a executar outras tarefas. Quando a resposta do servidor chega, ela é processada em uma Thread separada, permitindo que o cliente continue sua execução sem bloqueio.