

**Universidade:[Estácio]**

**Campus: [1685 Polo Anhaia Melo - São Paulo - SP]**

**Curso: [Desenvolvimento Full Stack]**

**Disciplina: [Nível2: Vamos manter as informações? ]**

**Turma: [9001]**

**Semestre Letivo: [2024.3]**

**Integrantes da Prática: [Fernando Garcia Farias]**

**Servidor Completo e Cliente Assíncrono**

O objetivo foi implementar uma aplicação cliente-servidor em Java, onde o servidor é capaz de receber conexões de múltiplos clientes de forma simultânea utilizando threads. O cliente, por sua vez, envia requisições assíncronas ao servidor e trata suas respostas de forma paralela.

1.Todos os códigos solicitados neste roteiro de aula;

Foram implementados códigos para o servidor e o cliente conforme especificado. O servidor utiliza ServerSocket para aceitar conexões e cria uma thread para cada cliente conectado. O cliente envia requisições ao servidor e trata as respostas em threads separadas, utilizando SwingUtilities.invokeLater para atualizações na interface gráfica.

2.Os resultados da execução dos códigos também devem ser apresentados;

Durante a execução, observou-se que o servidor foi capaz de aceitar múltiplas conexões simultaneamente, cada uma em sua própria thread. O cliente enviou requisições assíncronas ao servidor e tratou as respostas em threads separadas, mantendo a interface gráfica responsiva.

Análise e Conclusão:

1. **Como as Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor?**
   * Threads são utilizadas para processar operações de forma concorrente e assíncrona no cliente. Quando o cliente envia uma requisição ao servidor, ele pode continuar sua execução sem bloquear a interface gráfica. As threads são responsáveis por aguardar a resposta do servidor e processá-la quando recebida, garantindo que a interface permaneça responsiva.
2. **Para que serve o método invokeLater, da classe SwingUtilities?**
   * O método invokeLater é usado para despachar uma tarefa (Runnable) para ser executada na thread de despacho de eventos do Swing. Isso é essencial para atualizações da interface gráfica Swing, pois garante que as modificações nos componentes visuais sejam feitas na thread apropriada, evitando problemas de concorrência e garantindo a consistência da interface.
3. **Como os objetos são enviados e recebidos pelo Socket Java?**
   * Os objetos são enviados e recebidos pelo Socket Java através da serialização e desserialização. A classe ObjectOutputStream é usada para escrever objetos em bytes para o OutputStream do Socket, enquanto a classe ObjectInputStream é usada para ler e desserializar objetos a partir do InputStream do Socket. Isso permite o transporte de objetos complexos pela rede.
4. **Compare a utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java, ressaltando as características relacionadas ao bloqueio do processamento.**
   * No comportamento síncrono, o cliente aguarda pela resposta do servidor após enviar uma requisição, bloqueando a execução até que a resposta seja recebida. Isso pode tornar a interface gráfica não responsiva se a operação de rede demorar. Já no comportamento assíncrono, o cliente envia requisições em threads separadas e continua sua execução normalmente. As respostas são tratadas em threads separadas, garantindo que a interface permaneça interativa e não seja bloqueada por operações de rede.

Essa abordagem assíncrona é particularmente útil em aplicações GUI, onde a responsividade da interface é crucial para a experiência do usuário. O uso adequado de threads e da thread de despacho de eventos do Swing (invokeLater) são fundamentais para garantir uma aplicação fluída e responsiva durante operações de rede intensivas.

https://github.com/Futureforworld/M5