Matheus Bicalho Costa Lemberg – 202208610773

Polo Santa Inês – Belo Horizonte – MG

Turma 2022.3

Vamos integrar sistemas

Objetivos da prática

Criar servidores Java com base em Sockets.
Criar clientes síncronos para servidores com base em Sockets.
Criar clientes assíncronos para servidores com base em Sockets.
Utilizar Threads para implementação de processos paralelos.
No final do exercício, o aluno terá criado um servidor Java baseado em Socket, com acesso ao
banco de dados via JPA, alem de utilizar os recursos nativos do Java para implementação de
clientes síncronos e assíncronos. As Threads serão usadas tanto no servidor, para viabilizar
múltiplos clientes paralelos, quanto no cliente, para implementar a resposta assíncrona.

Análise e conclusão

Como funciona as classes Socket e ServerSocket?

As classes Socket e ServerSocket são usadas em comunicação de rede no Java. Socket representa um endpoint de comunicação bidirecional entre dispositivos, permitindo a troca de dados. ServerSocket aguarda e aceita conexões de clientes, criando um Socket dedicado para cada conexão estabelecida. Enquanto o Socket permite a comunicação entre cliente e servidor, o ServerSocket aguarda por conexões de clientes, estabelecendo a comunicação quando uma requisição é feita. Ambas as classes são fundamentais para a construção de aplicativos que envolvem a comunicação em rede, facilitando a troca de informações entre dispositivos conectados em uma rede.

Qual a importância das portas para a conexão com servidores?

As portas são cruciais na conexão com servidores, identificando diferentes serviços. Elas permitem que múltiplos serviços funcionem em um único servidor, direcionando solicitações para o serviço correto.

Para que servem as classes de entrada e saída ObjectInputStream e ObjectOutputStream, e por que os objetos transmitidos devem ser serializáveis?

As classes ObjectInputStream e ObjectOutputStream em Java são usadas para serializar e desserializar objetos durante a transmissão por meio de fluxos de entrada e saída. Elas permitem a escrita e leitura de objetos em bytes, facilitando a comunicação entre diferentes aplicações. É crucial que os objetos transmitidos sejam serializáveis para serem convertidos em bytes e reconstruídos em objetos originais em outro local. A serialização permite a conversão de objetos em uma sequência de bytes, viabilizando a transmissão pela rede ou armazenamento em arquivos, tornando-os interoperáveis entre diferentes plataformas e garantindo integridade na transferência de dados.

Por que, mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados?

Mesmo utilizando classes de entidades JPA no cliente, o isolamento do acesso ao banco de dados é garantido devido à camada de persistência. As classes de entidades JPA encapsulam a lógica de acesso aos dados, representando entidades do banco de dados e não sendo diretamente responsáveis pela conexão ou operações no banco. Essas classes representam apenas estruturas de dados, enquanto a comunicação com o banco e a execução de operações são realizadas pela camada de persistência e gerenciador de entidades, fornecidos pelo provedor JPA. Isso mantém a segurança e integridade dos dados, isolando a lógica de acesso ao banco de dados da lógica de negócios do cliente.

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

Threads podem ser empregadas para tratamento assíncrono das respostas do servidor ao criar processos paralelos. Ao receber as respostas do servidor, as Threads permitem que a aplicação continue a executar outras tarefas enquanto aguarda as respostas. Essa abordagem evita bloqueios no fluxo principal do programa, melhorando a responsividade e desempenho da aplicação. Cada resposta recebida pode ser tratada por uma Thread separada, permitindo o processamento simultâneo de múltiplas respostas. No entanto, é necessário um gerenciamento adequado de Threads para evitar problemas de concorrência e garantir a integridade dos dados, utilizando estruturas de sincronização ou ferramentas adequadas para controle e coordenação das Threads.

Para que serve o método invokeLater, da classe SwingUtilities?

O método invokeLater da classe SwingUtilities em Java serve para agendar a execução de uma determinada tarefa para ser realizada de forma assíncrona na Thread de despacho de eventos (Event Dispatch Thread - EDT) Swing. Ele permite que operações relacionadas à interface gráfica sejam realizadas de maneira segura e consistente, garantindo que essas operações sejam executadas na EDT. Isso é crucial para evitar problemas de concorrência e garantir a correta

atualização e manipulação de elementos da interface gráfica em aplicações Swing, proporcionando uma experiência de usuário mais estável e responsiva.

Como os objetos são enviados e recebidos pelo Socket Java?

Em Java, os objetos são enviados e recebidos pelo Socket através da utilização das classes ObjectInputStream e ObjectOutputStream. Ao enviar um objeto, ele é convertido em bytes pelo ObjectOutputStream, enviado pelo Socket e, do outro lado da conexão, o ObjectInputStream os reconstrói, permitindo a transmissão e recepção dos objetos pela rede de forma serializada e desserializada, respectivamente.

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

O comportamento síncrono em clientes com Socket Java implica em operações bloqueantes, onde o cliente aguarda pela resposta do servidor antes de prosseguir, bloqueando o processamento. Por outro lado, o comportamento assíncrono permite que o cliente continue sua execução enquanto aguarda pela resposta do servidor, evitando bloqueios e permitindo que outras operações sejam realizadas simultaneamente. Com o comportamento assíncrono, o cliente pode ser mais responsivo, executando tarefas adicionais sem esperar pela finalização das operações no Socket, tornando o processo mais eficiente e dinâmico.