Projeto Prático – Chat com Java Sockets

Professor:	Alexandre Huff
Disciplina:	Programação Orientada a Objetos 2
Meio para entrega:	Moodle

Descrição da Atividade

A atividade consiste em implementar uma aplicação de chat (bate-papo) em Java utilizando Sockets. A aplicação é composta de um **Cliente** que os usuários devem utilizar para trocar mensagens e de um **Servidor** que deverá centralizar e gerenciar todas as conexões das aplicações cliente.

Entrega: Compactar o diretório **src** e enviar pelo Moodle.

Especificação

Servidor

- Classe `Servidor`
  - Implementa o método `main()` da aplicação Servidor
  - O servidor da aplicação deve aguardar conexões dos clientes na porta 50123.
  - Múltiplas requisições devem ser atendidas simultaneamente pelo servidor. Crie uma thread para cada participante.
  - Armazene cada um dos clientes conectados (participantes) em uma lista que deve ser percorrida para enviar a mensagem para todos os participantes. *Atente-se para a possibilidade da lista ser modificada enquanto está sendo percorrida.*
  - Crie **um** *worker thread* (`FixedThreadPool`) que será responsável por enviar cada mensagem à lista de participantes.
    Vamos chamar este *worker thread* de "fofoqueiro".
- Classe `Participante` – `Runnable`
  - Implementa as threads responsáveis por manter a comunicação com as aplicações cliente.
  - Ao receber uma mensagem do respectivo cliente, o participante deve criar uma tarefa (`ServicoMensagem`) e solicitar para o *worker thread* "fofoquerio" executá-la.
  - A thread do participante encerra quando receber a mensagem "`##sair##`" da aplicação cliente.

- Classe `ServicoMensagem – Runnable`
  - Atributos:
    - apelido (*emissor*)
    - texto
  - Representa a tarefa que deve ser executada pelo *worker thread* "fofoqueiro".
  - A tarefa basicamente é percorrer a lista de clientes conectados no servidor e enviar a mensagem para cada um deles (inclusive para o emissor da mensagem). *Lembre-se que o acesso a esta lista é concorrente, participantes podem vir a ser adicionados ou removidos do chat enquanto uma mensagem estiver sendo enviada aos participantes.*
  - A mensagem enviada deve ser apresentada no console do servidor como um log, no seguinte formato:
    `27/04/2021 22:50 FINE (apelido do remetente) - Mensagem`

Cliente

- Classe `Cliente`
  - Implementa o método `main()` da aplicação Cliente.
  - A aplicação cliente deve receber o endereço IP do servidor e o apelido do participante via linha de comando ao ser executada.
  - O cliente deve ler a mensagem digitada pelo usuário e enviar para o servidor. O servidor se encarrega de replicar a mensagem para todos os clientes, inclusive para o emissor da mensagem.
  - Execute a aplicação cliente preferencialmente em uma janela conforme a Figura 1.
  - Ao fechar a janela, a aplicação cliente deve enviar a mensagem `"##sair##"` para o servidor.

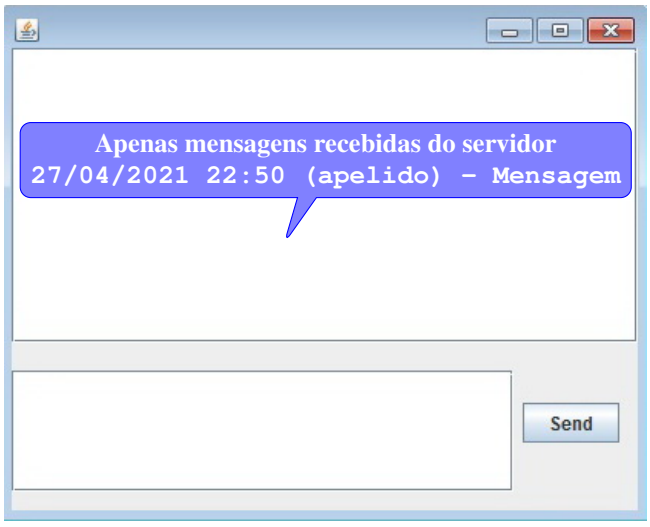

Figura 1: Exemplo de janela da aplicação cliente.

Ponto Extra: Depois que as aplicações Cliente e Servidor estiverem funcionando, empacote-as em arquivos JAR (`Servidor.jar` e `Cliente.jar`). Este é o tipo de arquivo padrão utilizado para fazer a implantação de aplicações Java. Também facilitará a execução de vários clientes simultâneos. **OBS:** Não é necessário entregar o arquivo `JAR`, apenas os fontes.

Importante: Durante o desenvolvimento, apresente mensagens de *log* que permitem identificar os passos que ocorrem durante a execução do servidor e dos clientes, de modo que seja possível visualizar o que está acontecendo. Lembre-se, este é o *log* do desenvolvedor e é tão importante quanto as mensagens apresentadas para os usuários da sua aplicação.

- Utilize a classe `java.util.logging.Logger` para gerar as mensagens de log no console.
- É **obrigatório** utilizar pelo menos os níveis de *log* FINE, INFO e ERROR.
- **Não** utilizar `System.out.println()`.
- Quando o código estiver pronto, altere o nível de log para INFO. Assim não é necessário comentar as linhas de *log* de depuração e que você não quer que apareça na tela.