Exercício 3 - Servidor com Interface Interativa de Console

Descrição da Solução

O **SimpleServer2Test.java** é uma versão aprimorada do servidor que permite ao usuário **editar ou confirmar** as mensagens antes de enviá-las ao cliente através de uma interface de console interativa.

Características Principais

Funcionalidades Implementadas

1. **Interface de Console Interativa**
   - Menu de opções claro e intuitivo
   - Três modos de resposta disponíveis
   - Feedback visual detalhado
2. **Resposta Padrão Automática**
   - Gera automaticamente: `"Servidor recebeu: [mensagem]"`
   - Exibe para o usuário antes de enviar
   - Pode ser confirmada com um clique
3. **Edição de Resposta**
   - Permite editar a resposta padrão
   - Mantém a mensagem original como base
   - Validação de entrada vazia
4. **Resposta Personalizada**
   - Criar resposta completamente nova
   - Total liberdade de conteúdo
   - Independente da mensagem recebida
5. **Contador de Mensagens**
   - Numeração automática de mensagens
   - Facilita rastreamento da conversação
6. **Formatação Visual Rica**
   - Separadores de seções
   - Identificadores coloridos ([INFO], [OK], [ERRO], etc.)
   - Layout organizado e profissional

Estrutura da Interface de Console

Cabeçalho Inicial

```
======================================================================
        SERVIDOR TCP INTERATIVO - SimpleServer2Test
======================================================================
[VERSAO] 2.0 - Com interface de console interativa
[PORTA] 12345
[FUNCAO] Recebe mensagens e permite edicao antes de responder
======================================================================

[SERVIDOR] Aguardando conexao na porta 12345...
======================================================================
```

Conexão Estabelecida

```
[CONEXAO] Cliente conectado!
[INFO] Endereco IP: 127.0.0.1
[INFO] Porta remota: 54321
======================================================================
[SERVIDOR] Aguardando mensagens do cliente...
```

Recebimento de Mensagem

```
======================================================================
[MENSAGEM #1 RECEBIDA]
======================================================================
Cliente enviou: Ola servidor!
----------------------------------------------------------------------

Resposta padrao gerada:
>>> Servidor recebeu: Ola servidor!
----------------------------------------------------------------------

Opcoes:
  [1] Confirmar e enviar a resposta padrao
  [2] Editar a resposta antes de enviar
  [3] Enviar resposta personalizada

Escolha uma opcao (1/2/3): _
```

Modos de Operação

Modo 1: Confirmar Resposta Padrão

Uso: Quando a resposta automática é adequada.

Fluxo:

```
Escolha uma opcao (1/2/3): 1

[OK] Resposta padrao confirmada.

[ENVIANDO] Resposta ao cliente:
>>> Servidor recebeu: Ola servidor!
[OK] Resposta enviada com sucesso!
======================================================================

[SERVIDOR] Aguardando proxima mensagem...
```

Vantagem: Rápido e eficiente para mensagens simples.

Modo 2: Editar Resposta Padrão

Uso: Quando quer modificar a resposta automática.

Fluxo:

```
Escolha uma opcao (1/2/3): 2

Edite a resposta (atual: "Servidor recebeu: Ola servidor!"): Ola! Recebi sua
mensagem: Ola servidor!

[OK] Resposta editada com sucesso.

[ENVIANDO] Resposta ao cliente:
>>> Ola! Recebi sua mensagem: Ola servidor!
[OK] Resposta enviada com sucesso!
======================================================================

[SERVIDOR] Aguardando proxima mensagem...
```

Vantagem: Mantém contexto da mensagem original, mas permite personalização.

Modo 3: Resposta Personalizada

Uso: Quando quer uma resposta completamente customizada.

Fluxo:

```
Escolha uma opcao (1/2/3): 3

Digite a resposta personalizada: Bem-vindo ao servidor! Como posso ajuda-lo?
```

```
[OK] Resposta personalizada criada.

[ENVIANDO] Resposta ao cliente:
>>> Bem-vindo ao servidor! Como posso ajuda-lo?
[OK] Resposta enviada com sucesso!
======================================================================

[SERVIDOR] Aguardando proxima mensagem...
```

Vantagem: Total liberdade criativa nas respostas.

Tratamento de Casos Especiais

1. Comando "sair"

Comportamento:

```
======================================================================
[MENSAGEM #5 RECEBIDA]
======================================================================
Cliente enviou: sair
----------------------------------------------------------------------

[AVISO] Cliente solicitou encerramento da conexao.
[ACAO] Enviando confirmacao de encerramento...
[OK] Confirmacao enviada. Encerrando servidor...

======================================================================
[ENCERRADO] Servidor encerrado com sucesso.
[INFO] Todos os recursos foram liberados.
======================================================================
```

Características:

- Detecta automaticamente o comando "sair"
- Não solicita confirmação/edição
- Envia resposta de encerramento automática
- Fecha recursos adequadamente

---

2. Opção Inválida

Comportamento:

```
Escolha uma opcao (1/2/3): 5

[AVISO] Opcao invalida. Usando resposta padrao.
```

```
[ENVIANDO] Resposta ao cliente:
>>> Servidor recebeu: Mensagem teste
[OK] Resposta enviada com sucesso!
```

Características:

- Aceita qualquer entrada diferente de 1, 2 ou 3
- Usa resposta padrão como fallback
- Não interrompe o fluxo
- Informa o usuário da decisão

---

3. Entrada Vazia na Edição

Comportamento:

```
Escolha uma opcao (1/2/3): 2

Edite a resposta (atual: "Servidor recebeu: teste"): [ENTER]

[AVISO] Edicao vazia. Usando resposta padrao.

[ENVIANDO] Resposta ao cliente:
>>> Servidor recebeu: teste
[OK] Resposta enviada com sucesso!
```

Características:

- Detecta se usuário pressiona Enter sem digitar
- Reverte para resposta padrão
- Evita envio de mensagem vazia

---

4. Entrada Vazia na Personalização

Comportamento:

```
Escolha uma opcao (1/2/3): 3

Digite a resposta personalizada: [ENTER]

[AVISO] Resposta vazia. Usando resposta padrao.

[ENVIANDO] Resposta ao cliente:
>>> Servidor recebeu: teste
[OK] Resposta enviada com sucesso!
```

Características:

- Valida entrada do usuário
- Fallback para resposta padrão
- Garante que sempre há resposta

Exemplo de Conversação Completa

Cenário: Múltiplas mensagens com diferentes modos

Mensagem 1: Confirmação Padrão

Console do Servidor:

```
======================================================================
[MENSAGEM #1 RECEBIDA]
======================================================================
Cliente enviou: Ola servidor!
----------------------------------------------------------------------

Resposta padrao gerada:
>>> Servidor recebeu: Ola servidor!
----------------------------------------------------------------------

Opcoes:
  [1] Confirmar e enviar a resposta padrao
  [2] Editar a resposta antes de enviar
  [3] Enviar resposta personalizada

Escolha uma opcao (1/2/3): 1

[OK] Resposta padrao confirmada.

[ENVIANDO] Resposta ao cliente:
>>> Servidor recebeu: Ola servidor!
[OK] Resposta enviada com sucesso!
======================================================================

[SERVIDOR] Aguardando proxima mensagem...
```

Mensagem 2: Edição de Resposta

Console do Servidor:

```
======================================================================
[MENSAGEM #2 RECEBIDA]
======================================================================
```

```
Cliente enviou: Como voce esta?
----------------------------------------------------------------------

Resposta padrao gerada:
>>> Servidor recebeu: Como voce esta?
----------------------------------------------------------------------

Opcoes:
  [1] Confirmar e enviar a resposta padrao
  [2] Editar a resposta antes de enviar
  [3] Enviar resposta personalizada

Escolha uma opcao (1/2/3): 2

Edite a resposta (atual: "Servidor recebeu: Como voce esta?"): Sim, recebi sua
pergunta: Como voce esta? Estou funcionando perfeitamente!

[OK] Resposta editada com sucesso.

[ENVIANDO] Resposta ao cliente:
>>> Sim, recebi sua pergunta: Como voce esta? Estou funcionando perfeitamente!
[OK] Resposta enviada com sucesso!
======================================================================

[SERVIDOR] Aguardando proxima mensagem...
```

Mensagem 3: Resposta Personalizada

Console do Servidor:

```
======================================================================
[MENSAGEM #3 RECEBIDA]
======================================================================
Cliente enviou: Qual e o seu nome?
----------------------------------------------------------------------

Resposta padrao gerada:
>>> Servidor recebeu: Qual e o seu nome?
----------------------------------------------------------------------

Opcoes:
  [1] Confirmar e enviar a resposta padrao
  [2] Editar a resposta antes de enviar
  [3] Enviar resposta personalizada

Escolha uma opcao (1/2/3): 3

Digite a resposta personalizada: Meu nome e SimpleServer2Test, um servidor TCP
interativo criado em Java!
```

```
[OK] Resposta personalizada criada.

[ENVIANDO] Resposta ao cliente:
>>> Meu nome e SimpleServer2Test, um servidor TCP interativo criado em Java!
[OK] Resposta enviada com sucesso!
======================================================================

[SERVIDOR] Aguardando proxima mensagem...
```

Mensagem 4: Encerramento

Console do Servidor:

```
======================================================================
[MENSAGEM #4 RECEBIDA]
======================================================================
Cliente enviou: sair
----------------------------------------------------------------------

[AVISO] Cliente solicitou encerramento da conexao.
[ACAO] Enviando confirmacao de encerramento...
[OK] Confirmacao enviada. Encerrando servidor...

======================================================================
[ENCERRADO] Servidor encerrado com sucesso.
[INFO] Todos os recursos foram liberados.
======================================================================
```

Como Executar

Pré-requisitos

```
- JDK 8 ou superior
- SimpleClientTest.java (interface gráfica do Exercício 2)
```

Compilação

```
javac SimpleServer2Test.java
```

Execução

```
java SimpleServer2Test
```

Sequência Recomendada

1. **Inicie o servidor:**

```
java SimpleServer2Test
```

2. **Aguarde a mensagem:**

```
[SERVIDOR] Aguardando conexao na porta 12345...
```

3. **Inicie o cliente (Exercício 2):**

```
java SimpleClientTest
```

4. **Envie mensagens pelo cliente e responda pelo console do servidor**

Comparação com o Servidor Original

Aspecto	SimpleServerTest	SimpleServer2Test
Resposta	Automática	Editável pelo usuário
Interação	Nenhuma	Console interativo
Opções	0	3 modos diferentes
Personalização	Não	Sim, total
Feedback	Básico	Detalhado e visual
Controle	Passivo	Ativo
Usabilidade	Simples	Avançada
Flexibilidade	Baixa	Alta

Estrutura do Código

Método Principal (main)

```
1. Inicialização
   ├── Criação do ServerSocket (porta 12345)
   ├── Exibição do cabeçalho
   └── Aguarda conexão
```

```
2. Aceitação do Cliente
   ├── Accept() bloqueante
   ├── Exibe informações do cliente
   └── Configura streams I/O

3. Loop de Mensagens
   ├── Recebe mensagem (readLine)
   ├── Incrementa contador
   ├── Verifica comando "sair"
   ├── Gera resposta padrão
   ├── Exibe menu de opções
   ├── Processa escolha do usuário
   │   ├── Opção 1: Confirma padrão
   │   ├── Opção 2: Edita resposta
   │   ├── Opção 3: Cria personalizada
   │   └── Padrão: Usa resposta padrão
   ├── Envia resposta (println)
   └── Aguarda próxima mensagem

4. Encerramento
   ├── Fecha Scanner
   ├── Fecha streams
   ├── Fecha sockets
   └── Exibe mensagem de encerramento
```

Método Auxiliar

```
exibirCabecalho()
├── Imprime banner do servidor
├── Exibe versão e porta
└── Mostra descrição da funcionalidade
```

Recursos Técnicos

1. Scanner para Entrada do Usuário

```
Scanner scanner = new Scanner(System.in);
String opcao = scanner.nextLine().trim();
```

- Captura entrada do console
- `.trim()` remove espaços em branco
- Validação de entrada

2. Switch-Case para Menu

```
switch (opcao) {
    case "1": // Confirmar
    case "2": // Editar
    case "3": // Personalizar
    default:  // Padrão
}
```

- Estrutura clara e legível
- Fácil manutenção
- Extensível para novas opções

3. Formatação Visual

```
"=".repeat(70)  // Linha dupla
"-".repeat(70)  // Linha simples
```

- Java 11+ (método repeat)
- Separadores visuais claros
- Layout profissional

4. Identificadores de Status

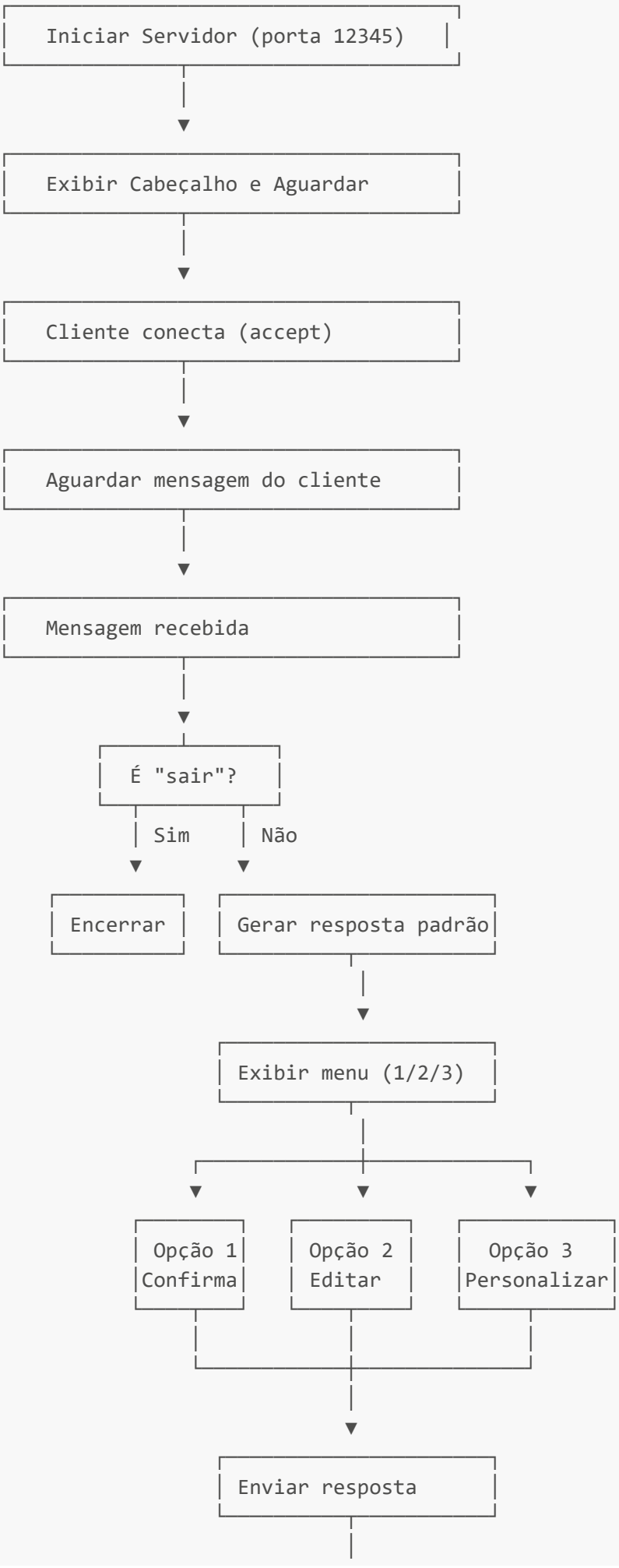
```
[SERVIDOR]  - Mensagens do servidor
[CONEXAO]   - Eventos de conexão
[INFO]      - Informações gerais
[OK]        - Operação bem-sucedida
[AVISO]     - Alerta/warning
[ERRO]      - Erro/exceção
[ENVIANDO]  - Envio de dados
[ENCERRADO] - Finalização
```

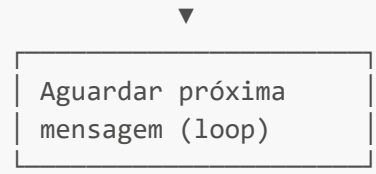
5. Contador de Mensagens

```
int contadorMensagens = 0;
contadorMensagens++;
System.out.println("[MENSAGEM #" + contadorMensagens + " RECEBIDA]");
```

- Rastreamento de ordem
- Útil para debug
- Facilita documentação

Fluxograma do Servidor


Iniciar Servidor (porta 12345)
Exibir Cabeçalho e Aguardar
Cliente conecta (accept)
Aguardar mensagem do cliente
Mensagem recebida
É "sair"?
Sim
Não
Encerrar
Gerar resposta padrão
Exibir menu (1/2/3)
Opção 1 Confirma
Opção 2 Editar
Opção 3 Personalizar
Enviar resposta



Possíveis Extensões

1. **Histórico de Conversação**

```
List<String> historico = new ArrayList<>();
historico.add("Cliente: " + mensagemRecebida);
historico.add("Servidor: " + respostaFinal);
```

2. **Salvar Logs em Arquivo**

```
FileWriter logger = new FileWriter("servidor_log.txt", true);
logger.write("[" + timestamp + "] " + mensagem + "\n");
```

3. **Respostas Pré-definidas**

```
Map<String, String> respostasPredefinidas = new HashMap<>();
respostasPredefinidas.put("oi", "Ola! Bem-vindo!");
respostasPredefinidas.put("ajuda", "Como posso ajuda-lo?");
```

4. **Comandos Especiais**

```
if (mensagem.startsWith("/")) {
    processarComando(mensagem);
}
```

5. **Múltiplos Clientes**

```
while (true) {
    Socket client = serverSocket.accept();
    new Thread(() -> handleClient(client)).start();
}
```

Casos de Uso

Uso Educacional

- Ensino de sockets TCP
- Demonstração de I/O interativo
- Prática de fluxo de controle

Desenvolvimento/Debug

- Testar respostas diferentes
- Simular cenários variados
- Validar comportamento do cliente

Prototipagem

- Testar mensagens antes de automatizar
- Avaliar experiência do usuário
- Refinar protocolo de comunicação

Desenvolvido para: Disciplina de Linguagens de Programação
Data: 21/10/2025
Tecnologia: Java Sockets, Console I/O, Scanner