Exercício 2 - Interface Gráfica para SimpleClientTest

Modificações Implementadas

Requisitos Atendidos

1. Eliminação de Console e JOptionPane

- Removido: `System.out.println()`
- Removido: `System.err.println()`
- Removido: `JOptionPane.showInputDialog()`
- Removido: `JOptionPane.showMessageDialog()`
- Todas as interações agora são através da interface gráfica

2. Campos Específicos Criados

- **Campo de Entrada de Mensagem** (`JTextField`)
- **Área de Respostas do Servidor** (`JTextArea`)
- **Área de Status da Comunicação** (`JTextArea`)

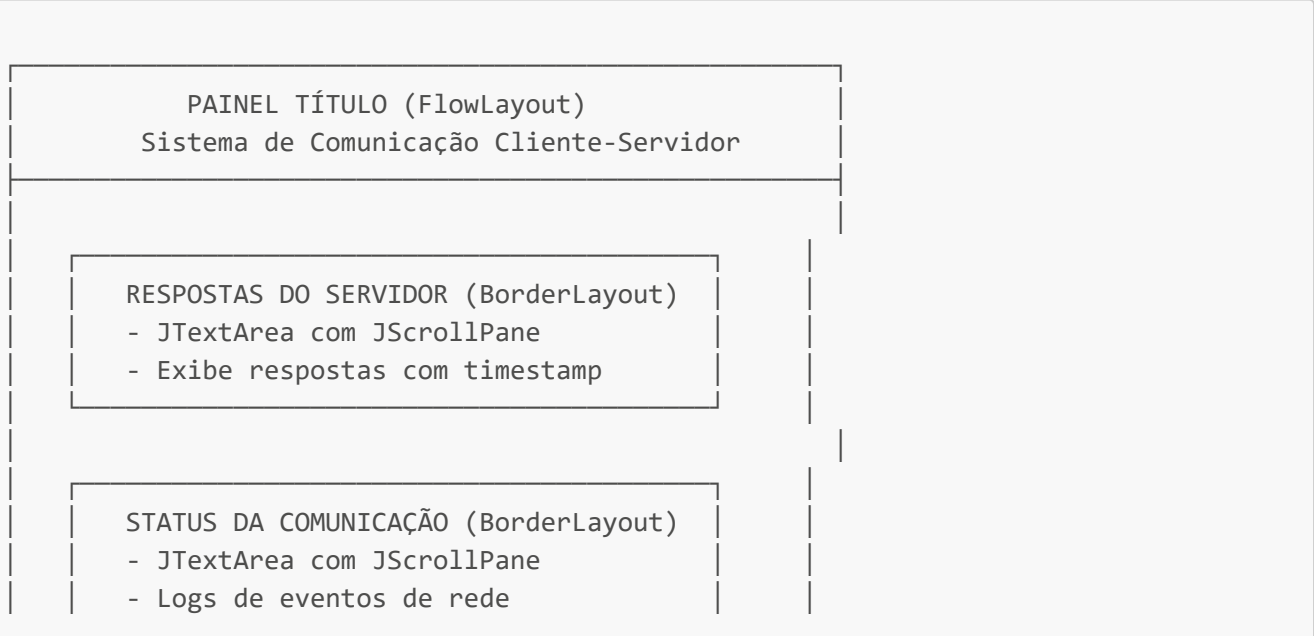
3. Botões e Funcionalidades

- **Botão Enviar** - Envia mensagem ao servidor
- **Botão Limpar** - Limpa todos os campos
- **Botão Sair** - Encerra conexão e aplicação

Arquitetura da Interface

Layout Utilizado

A interface utiliza uma combinação de layouts para organização profissional:



```
│   └──────────────────────────────────────────┘     │
│                                                    │
├────────────────────────────────────────────────────┤
│  Mensagem: [_____________________________]         │
│                                                    │
│     [Enviar]    [Limpar]    [Sair]                 │
└────────────────────────────────────────────────────┘
```

Hierarquia de Containers

```
JFrame (BorderLayout)
├── NORTH: painelTitulo (FlowLayout)
│   └── JLabel com título
│
├── CENTER: painelCentral (GridLayout 2x1)
│   ├── painelRespostas (BorderLayout)
│   │   └── JScrollPane
│   │       └── txtAreaRespostas (JTextArea)
│   │
│   └── painelStatus (BorderLayout)
│       └── JScrollPane
│           └── txtAreaStatus (JTextArea)
│
└── SOUTH: painelInferior (BorderLayout)
    ├── CENTER: painelEntrada (BorderLayout)
    │   ├── WEST: JLabel "Mensagem:"
    │   └── CENTER: txtMensagem (JTextField)
    │
    └── SOUTH: painelBotoes (FlowLayout)
        ├── btnEnviar
        ├── btnLimpar
        └── btnSair
```

Componentes da Interface

1. Painel de Título

```
- Background: Azul (RGB: 51, 102, 153)
- Fonte: Arial, Bold, 18pt
- Cor do texto: Branco
- Conteúdo: "Sistema de Comunicação Cliente-Servidor"
```

2. Área de Respostas do Servidor

```
- Tipo: JTextArea (não editável)
- Fonte: Monospaced, 12pt
- Bordas: TitledBorder azul
- Scroll: Vertical e Horizontal automáticos
- Auto-scroll: Para última mensagem
- Formato: [HH:mm:ss] Resposta do servidor
```

3. Área de Status da Comunicação

```
- Tipo: JTextArea (não editável)
- Fonte: Monospaced, 11pt
- Background: Verde claro (RGB: 240, 255, 240)
- Bordas: TitledBorder verde
- Scroll: Vertical e Horizontal automáticos
- Auto-scroll: Para última mensagem
- Formato: [HH:mm:ss] Status da operação
```

4. Campo de Entrada de Mensagem

```
- Tipo: JTextField
- Fonte: Arial, 12pt
- Altura: 30px
- Funcionalidade: Pressionar Enter também envia mensagem
- Foco automático: Após enviar ou limpar
```

5. Botões

Botão Enviar

```
- Cor: Verde (RGB: 51, 153, 102)
- Texto: Branco, Bold, 12pt
- Dimensões: 100x35px
- Ação: Envia mensagem ao servidor
- Atalho: Enter no campo de texto
```

Botão Limpar

```
- Cor: Laranja (RGB: 255, 153, 51)
- Texto: Branco, Bold, 12pt
- Dimensões: 100x35px
- Ação: Limpa todos os campos (mensagem, respostas, status)
```

Botão Sair

```
- Cor: Vermelho (RGB: 204, 51, 51)
- Texto: Branco, Bold, 12pt
- Dimensões: 100x35px
- Ação: Confirma saída e encerra aplicação
- Confirmação: JOptionPane de confirmação
```

Funcionalidades Implementadas

1. Conexão Automática ao Servidor

Comportamento:

```
- Ao iniciar a aplicação, tenta conectar automaticamente
- Se conectado: Exibe status de sucesso
- Se falhou: Exibe erro e desabilita envio
```

Status exibidos:

```
[10:30:15] Tentando conectar ao servidor em localhost:12345...
[10:30:15] ✓ Conectado ao servidor com sucesso!
[10:30:15] Endereço: 127.0.0.1
[10:30:15] Porta local: 54321
[10:30:15] ----------------------------------
```

2. Envio de Mensagens

Fluxo:

```
1. Usuário digita mensagem no campo
2. Clica "Enviar" ou pressiona Enter
3. Sistema valida mensagem (não vazia)
4. Envia ao servidor
5. Atualiza status: "→ Enviando: 'mensagem'"
6. Aguarda resposta em thread separada
7. Exibe resposta na área apropriada
8. Atualiza status: "← Resposta recebida"
9. Limpa campo de entrada
10. Retorna foco ao campo
```

Validações:

```
- Verifica se está conectado
- Verifica se mensagem não está vazia
- Trata erros de I/O
```

3. Recebimento de Respostas

Características:

```
- Processamento assíncrono (thread separada)
- Não bloqueia interface durante espera
- Timestamp automático
- Auto-scroll para última resposta
```

Exemplo de exibição:

```
[10:31:20] Servidor recebeu: Olá servidor!
[10:31:25] Servidor recebeu: Como vai?
[10:31:30] Servidor recebeu: Tudo bem
```

4. Sistema de Status

Tipos de mensagens:

```
✓ Sucesso    - Operações bem-sucedidas
→ Enviando   - Mensagem sendo enviada
← Recebendo  - Resposta recebida
✗ Erro       - Falhas e exceções
```

Eventos registrados:

```
- Tentativa de conexão
- Conexão estabelecida
- Envio de mensagens
- Recebimento de respostas
- Erros de rede
- Encerramento de conexão
- Campos limpos
```

5. Botão Limpar

Ação:

```
- Limpa campo de entrada de mensagem
- Limpa área de respostas
- Limpa área de status
- Adiciona mensagem "Campos limpos"
- Retorna foco ao campo de entrada
```

6. Botão Sair

Fluxo:

```
1. Exibe JOptionPane de confirmação
2. Se confirmado:
   a. Envia "sair" ao servidor
   b. Registra no status
   c. Fecha recursos (streams, socket)
   d. Encerra aplicação (System.exit(0))
3. Se cancelado:
   - Retorna à aplicação normalmente
```

7. Comando "sair" no Campo

Comportamento especial:

```
- Se usuário digitar "sair" no campo
- Envia ao servidor normalmente
- Aguarda 1 segundo (Timer)
- Encerra conexão automaticamente
- Mantém aplicação aberta para visualização
```

8. Fechamento da Janela

WindowListener:

```
- Detecta quando usuário clica no X da janela
- Chama método encerrarConexao()
- Fecha todos os recursos antes de sair
- Previne vazamento de recursos
```

Melhorias de UX (User Experience)

1. **Timestamps**

- Todas as mensagens possuem timestamp

- Formato: HH:mm:ss (24 horas)
- Facilita rastreamento temporal

2. **Auto-scroll**

- Áreas de texto rolam automaticamente
- Sempre exibe a última mensagem
- Usuário não precisa rolar manualmente

3. **Cores Semânticas**

```
- Azul:    Informação (respostas)
- Verde:   Status/logs
- Laranja: Ação de limpeza
- Vermelho: Ação destrutiva (sair)
```

4. **Feedback Visual**

```
- Botões com cores distintas
- Áreas com bordas coloridas
- Símbolos no status (✓, ✗, →, ←)
```

5. **Atalhos de Teclado**

```
- Enter: Envia mensagem
- Alt+E: Botão Enviar (mnemonic automático)
- Alt+L: Botão Limpar
- Alt+S: Botão Sair
```

6. **Look and Feel do Sistema**

```
UIManager.setLookAndFeelClassName()
```

- Interface se adapta ao sistema operacional
- Windows: aparência Windows
- Linux: aparência Linux/GTK
- Mac: aparência macOS

7. **Desabilitação Inteligente**

```
- Se não conectar ao servidor:
  * Campo de mensagem desabilitado
```

```
  * Botão Enviar desabilitado
  * Mensagem explicativa no campo
```

8. **Thread Safety**

```
SwingUtilities.invokeLater()
```

- Atualizações de UI na EDT (Event Dispatch Thread)
- Previne problemas de concorrência
- Interface responsiva

Tratamento de Erros

1. Servidor Não Disponível

```
[10:30:15] Tentando conectar ao servidor em localhost:12345...
[10:30:16] ✗ ERRO: Não foi possível conectar ao servidor
[10:30:16]   Verifique se o servidor está em execução!
```

Ação: Desabilita campo de mensagem e botão Enviar

2. Mensagem Vazia

```
[10:31:20] ✗ Mensagem vazia não pode ser enviada!
```

Ação: Apenas exibe aviso, usuário pode tentar novamente

3. Erro de I/O durante Envio

```
[10:32:10] ✗ ERRO ao enviar mensagem: Connection reset
```

Ação: Registra erro, permite nova tentativa

4. Erro ao Receber Resposta

```
[10:32:15] ✗ ERRO ao receber resposta: Socket closed
```

Ação: Registra erro, indica problema na conexão

5. Não Conectado

```
[10:33:00] ✗ Não conectado ao servidor!
```

Ação: Avisa usuário que não há conexão ativa

Como Executar

1. Pré-requisitos

```
- JDK 8 ou superior
- SimpleServerTest.java em execução
```

2. Compilação

```
javac SimpleClientTest.java
```

3. Execução

```
java SimpleClientTest
```

4. Sequência Recomendada

```
1. Inicie SimpleServerTest.java
2. Aguarde mensagem "Servidor aguardando conexão na porta 12345..."
3. Inicie SimpleClientTest.java
4. Interface gráfica abrirá automaticamente
5. Verifique mensagem de conexão bem-sucedida
6. Digite mensagens e clique "Enviar"
```

Exemplo de Uso Completo

Cenário: Conversação com o Servidor

Passo 1: Inicialização

```
[10:30:00] Tentando conectar ao servidor em localhost:12345...
[10:30:00] ✓ Conectado ao servidor com sucesso!
```

```
[10:30:00] Endereço: 127.0.0.1
[10:30:00] Porta local: 54321
[10:30:00] ----------------------------------
```

Passo 2: Primeira Mensagem

Campo de entrada: `Olá servidor!` **Clica:** Enviar

Status:

```
[10:30:15] → Enviando: "Olá servidor!"
[10:30:15] ← Resposta recebida
```

Respostas:

```
[10:30:15] Servidor recebeu: Olá servidor!
```

Passo 3: Segunda Mensagem

Campo de entrada: `Como você está?` **Clica:** Enviar

Status:

```
[10:30:20] → Enviando: "Como você está?"
[10:30:20] ← Resposta recebida
```

Respostas:

```
[10:30:15] Servidor recebeu: Olá servidor!
[10:30:20] Servidor recebeu: Como você está?
```

Passo 4: Limpar Campos

Clica: Limpar

Status:

```
[10:30:25] Campos limpos
[10:30:25] ----------------------------------
```

Respostas: *(área vazia)*

Passo 5: Nova Mensagem

Campo de entrada: `Teste após limpar` **Pressiona:** Enter

Status:

```
[10:30:25] Campos limpos
[10:30:25] ----------------------------------
[10:30:30] → Enviando: "Teste após limpar"
[10:30:30] ← Resposta recebida
```

Respostas:

```
[10:30:30] Servidor recebeu: Teste após limpar
```

Passo 6: Encerrar

Campo de entrada: `sair` **Clica:** Enviar

Status:

```
[10:30:35] → Enviando: "sair"
[10:30:35] ← Resposta recebida
[10:30:35] Encerrando conexão...
[10:30:36] ✓ Conexão encerrada
```

Respostas:

```
[10:30:30] Servidor recebeu: Teste após limpar
[10:30:35] Servidor recebeu: sair
```

Código-fonte Organizado

Estrutura de Classes

```
public class SimpleClientTest extends JFrame {
    // 1. ATRIBUTOS
    private JTextField txtMensagem;           // Campo de entrada
    private JTextArea txtAreaRespostas;       // Área de respostas
    private JTextArea txtAreaStatus;          // Área de status
    private JButton btnEnviar;                // Botão enviar
    private JButton btnLimpar;                // Botão limpar
```

```
    private JButton btnSair;                    // Botão sair
    private Socket socket;                      // Socket de conexão
    private BufferedReader in;                  // Stream de entrada
    private PrintWriter out;                    // Stream de saída
    private boolean conectado;                  // Flag de conexão

    // 2. CONSTRUTOR
    public SimpleClientTest()

    // 3. MÉTODOS DE INTERFACE
    private void inicializarComponentes()     // Cria GUI

    // 4. MÉTODOS DE REDE
    private void conectarAoServidor()         // Estabelece conexão
    private void encerrarConexao()            // Fecha recursos

    // 5. MÉTODOS DE AÇÃO
    private void enviarMensagem()             // Envia ao servidor
    private void limparCampos()               // Limpa interface
    private void sair()                       // Encerra aplicação

    // 6. MÉTODOS AUXILIARES
    private void desabilitarEnvio()           // Desabilita envio
    private void adicionarResposta()          // Adiciona resposta
    private void adicionarStatus()            // Adiciona status

    // 7. MAIN
    public static void main(String[] args)    // Ponto de entrada
}
```

Padrões de Design Utilizados

1. MVC (Model-View-Controller)

```
- Model: Socket, BufferedReader, PrintWriter
- View: JFrame, JTextArea, JTextField, JButton
- Controller: ActionListeners, métodos de ação
```

2. Observer Pattern

```
- ActionListeners nos botões
- WindowListener no JFrame
- ActionListener no JTextField
```

3. Singleton (implícito)

```
- Apenas uma instância da aplicação por execução
- Socket único para comunicação
```

4. Template Method

```
- SwingUtilities.invokeLater()
- Timer para ações atrasadas
```

Comparação: Antes vs Depois

Aspecto	Antes (JOptionPane)	Depois (Interface Gráfica)
Interação	Uma mensagem por vez	Múltiplas mensagens contínuas
Histórico	Não disponível	Histórico completo com timestamps
Status	Console (invisível)	Área dedicada na interface
Usabilidade	Janelas modais (bloqueantes)	Interface fluida e responsiva
Profissionalismo	Aparência básica	Interface moderna e organizada
Feedback	Limitado	Detalhado e visual
Erros	Console ou dialog	Exibição clara na área de status

Possíveis Extensões Futuras

1. **Múltiplos Servidores**

- ComboBox para selecionar servidor
- Campo para IP e porta customizados

2. **Histórico Persistente**

- Salvar conversas em arquivo
- Botão "Salvar Histórico"

3. **Formatação de Mensagens**

- Rich text (negrito, itálico)
- Emojis
- Cores personalizadas

4. **Notificações**

- Som ao receber resposta

- Notificação de sistema
- Badge com contador

5. **Auto-reconexão**

- Detectar perda de conexão
- Tentar reconectar automaticamente
- Fila de mensagens pendentes

6. **Criptografia**

- SSL/TLS para comunicação segura
- Senha de autenticação

7. **Chat em Grupo**

- Suporte a múltiplos clientes
- Lista de usuários online
- Mensagens privadas

Desenvolvido para: Disciplina de Linguagens de Programação
Data: 21/10/2025
Tecnologias: Java Swing, Sockets TCP, Event-Driven Programming