UNIVERSIDADE ESTÁCIO DE SÁ

PROGRAMAÇÃO FULL STACK

PÓLO NORTE SHOPPING

CLEYDSON ASSIS COELHO

MISSÃO PRÁTICA 5 MUNDO 3
PROCEDIMENTO 2

RPG0018 - Por que não paralelizar

https://github.com/cleydsoncoelho/Mundo3Missao5Procedimento2

RIO DE JANEIRO

2025

Objetivos da prática

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

Conforme a metodologia gamificada propõe, a dificuldade das missões aumenta a cada nível, então o Nível 5 é o mais complexo do Mundo atual. Para melhorar a experiência do aluno, orientamos a desenvolver esse nível envolvendo empresas parceiras, colegas ou interações externas, para desenvolver o contato externo e praticar a presencialidade em suas atividades.

Produto.java (CLIENTE)

```
package model;
import java.io.Serializable;
public class Produto implements Serializable {
private static final long serialVersionUID = 1L;
private Integer idProduto;
private String nome;
private Integer quantidade;
private Double precoVenda;

public Produto() {}

public Integer getIdProduto() { return idProduto; }
public void setIdProduto(Integer idProduto) { this.idProduto = idProduto; }

public String getNome() { return nome; }
public void setNome(String nome) { this.nome = nome; }

public Integer getQuantidade() { return quantidade; }
public void setQuantidade(Integer quantidade) { this.quantidade = quantidade; }
```

```
public Double getPrecoVenda() { return precoVenda; }
public void setPrecoVenda(Double precoVenda) { this.precoVenda = precoVenda; }

@Override
public String toString() {
   return "Produto [id=" + idProduto + ", nome=" + nome + ", quantidade=" + quantidade + ",
preco=" + precoVenda + "]";
}
}
```

Produto.java (SERVIDOR)

```
package model;
import java.io.Serializable;
import javax.persistence.*;
@Entity
@Table(name = "Produto")
public class Produto implements Serializable {
private static final long serialVersionUID = 1L;
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "id_produto")
private Integer idProduto;

@Column(name = "nome")
private String nome;

@Column(name = "quantidade")
private Integer quantidade;

@Column(name = "preco_venda")
private Double precoVenda;

public Produto() {}

public Integer getIdProduto() { return idProduto; }
public void setIdProduto(Integer idProduto) { this.idProduto = idProduto; }

public String getNome() { return nome; }
public void setNome(String nome) { this.nome = nome; }

public Integer getQuantidade() { return quantidade; }
public void setQuantidade(Integer quantidade) { this.quantidade = quantidade; }

public Double getPrecoVenda() { return precoVenda; }
public void setPrecoVenda(Double precoVenda) { this.precoVenda = precoVenda; }

@Override
public String toString() {
```

```
   return "Produto [id=" + idProduto + ", nome=" + nome + ", quantidade=" + quantidade + ",
preco=" + precoVenda + "]";
}
}
```

Pessoa.java (CLIENTE)

```
package model;
import java.io.Serializable;
public class Pessoa implements Serializable {
private static final long serialVersionUID = 1L;
private Integer idPessoa;
private String nome;
private String logradouro;
private String telefone;

public Pessoa() {}

public Integer getIdPessoa() { return idPessoa; }
public void setIdPessoa(Integer idPessoa) { this.idPessoa = idPessoa; }

public String getNome() { return nome; }
public void setNome(String nome) { this.nome = nome; }

public String getLogradouro() { return logradouro; }
public void setLogradouro(String logradouro) { this.logradouro = logradouro; }

public String getTelefone() { return telefone; }
public void setTelefone(String telefone) { this.telefone = telefone; }

@Override
public String toString() {
   return "Pessoa [id=" + idPessoa + ", nome=" + nome + "]";
}
}
```

Pessoa.java (SERVIDOR)

```
package model;
import java.io.Serializable;
import javax.persistence.*;
@Entity
@Table(name = "Pessoa")
@Inheritance(strategy = InheritanceType.JOINED)
public class Pessoa implements Serializable {
private static final long serialVersionUID = 1L;
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "id_pessoa")
private Integer idPessoa;
```

```
@Column(name = "nome")
private String nome;

@Column(name = "logradouro")
private String logradouro;

@Column(name = "telefone")
private String telefone;

public Pessoa() {}

public Integer getIdPessoa() { return idPessoa; }
public void setIdPessoa(Integer idPessoa) { this.idPessoa = idPessoa; }

public String getNome() { return nome; }
public void setNome(String nome) { this.nome = nome; }

public String getLogradouro() { return logradouro; }
public void setLogradouro(String logradouro) { this.logradouro = logradouro; }

public String getTelefone() { return telefone; }
public void setTelefone(String telefone) { this.telefone = telefone; }

@Override
public String toString() {
   return "Pessoa [id=" + idPessoa + ", nome=" + nome + "]";
}
}
```

PessoaFisica.java (CLIENTE)

```
package model;
import java.io.Serializable;
public class PessoaFisica extends Pessoa implements Serializable {
private static final long serialVersionUID = 1L;
private String cpf;

public PessoaFisica() {}

public String getCpf() { return cpf; }
public void setCpf(String cpf) { this.cpf = cpf; }

@Override
public String toString() {
   return "PessoaFisica [id=" + getIdPessoa() + ", nome=" + getNome() + ", cpf=" + cpf + "]";
}
}
```

PessoaFisica.java (SERVIDOR)

```
package model;
import java.io.Serializable;
import javax.persistence.*;
@Entity
@Table(name = "PessoaFisica")
public class PessoaFisica extends Pessoa implements Serializable {
private static final long serialVersionUID = 1L;
@Column(name = "cpf")
private String cpf;

public PessoaFisica() {}

public String getCpf() { return cpf; }
public void setCpf(String cpf) { this.cpf = cpf; }

@Override
public String toString() {
   return "PessoaFisica [id=" + getIdPessoa() + ", nome=" + getNome() + ", cpf=" + cpf + "]";
}
}
```

PessoaJuridica.java (CLIENTE)

```
package model;
import java.io.Serializable;
public class PessoaJuridica extends Pessoa implements Serializable {
private static final long serialVersionUID = 1L;
private String cnpj;

public PessoaJuridica() {}

public String getCnpj() { return cnpj; }
public void setCnpj(String cnpj) { this.cnpj = cnpj; }

@Override
public String toString() {
   return "PessoaJuridica [id=" + getIdPessoa() + ", nome=" + getNome() + ", cnpj=" + cnpj +
"]";
}
}
```

PessoaJuridica.java (SERVIDOR)

```
package model;
import java.io.Serializable;
import javax.persistence.*;
```

```
@Entity
@Table(name = "PessoaJuridica")
public class PessoaJuridica extends Pessoa implements Serializable {
private static final long serialVersionUID = 1L;
@Column(name = "cnpj")
private String cnpj;

public PessoaJuridica() {}

public String getCnpj() { return cnpj; }
public void setCnpj(String cnpj) { this.cnpj = cnpj; }

@Override
public String toString() {
   return "PessoaJuridica [id=" + getIdPessoa() + ", nome=" + getNome() + ", cnpj=" + cnpj +
"]";
}
}
```

Usuario.java (CLIENTE)

```
package model;
import java.io.Serializable;
public class Usuario implements Serializable {
private static final long serialVersionUID = 1L;
private Integer idUsuario;
private String nome;
private String login;
private String senha;

public Usuario() {}

public Integer getIdUsuario() { return idUsuario; }
public void setIdUsuario(Integer idUsuario) { this.idUsuario = idUsuario; }

public String getNome() { return nome; }
public void setNome(String nome) { this.nome = nome; }

public String getLogin() { return login; }
public void setLogin(String login) { this.login = login; }

public String getSenha() { return senha; }
public void setSenha(String senha) { this.senha = senha; }

@Override
public String toString() {
   return "Usuario [id=" + idUsuario + ", login=" + login + "]";
}
}
```

Usuario.java (SERVIDOR)

```
package model;
import java.io.Serializable;
import javax.persistence.*;
@Entity
@Table(name = "Usuario")
public class Usuario implements Serializable {
private static final long serialVersionUID = 1L;
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "id_usuario")
private Integer idUsuario;

@Column(name = "nome")
private String nome;

@Column(name = "login")
private String login;

@Column(name = "senha")
private String senha;

public Usuario() {}

public Integer getIdUsuario() { return idUsuario; }
public void setIdUsuario(Integer idUsuario) { this.idUsuario = idUsuario; }

public String getNome() { return nome; }
public void setNome(String nome) { this.nome = nome; }

public String getLogin() { return login; }
public void setLogin(String login) { this.login = login; }

public String getSenha() { return senha; }
public void setSenha(String senha) { this.senha = senha; }

@Override
public String toString() {
   return "Usuario [id=" + idUsuario + ", login=" + login + "]";
}
}
```

Movimento.java (CLIENTE)

```
package model;
import java.io.Serializable;
import java.util.Date;
```

```
public class Movimento implements Serializable {
private static final long serialVersionUID = 1L;
private Integer idMovimento;
private Usuario usuario;
private String tipo;
private Pessoa pessoa;
private Produto produto;
private Integer quantidade;
private Double valorUnitario;
private Date dataMovimento;

public Movimento() {}

public Integer getIdMovimento() { return idMovimento; }
public void setIdMovimento(Integer idMovimento) { this.idMovimento = idMovimento; }

public Usuario getUsuario() { return usuario; }
public void setUsuario(Usuario usuario) { this.usuario = usuario; }

public String getTipo() { return tipo; }
public void setTipo(String tipo) { this.tipo = tipo; }

public Pessoa getPessoa() { return pessoa; }
public void setPessoa(Pessoa pessoa) { this.pessoa = pessoa; }

public Produto getProduto() { return produto; }
public void setProduto(Produto produto) { this.produto = produto; }

public Integer getQuantidade() { return quantidade; }
public void setQuantidade(Integer quantidade) { this.quantidade = quantidade; }

public Double getValorUnitario() { return valorUnitario; }
public void setValorUnitario(Double valorUnitario) { this.valorUnitario = valorUnitario; }

public Date getDataMovimento() { return dataMovimento; }
public void setDataMovimento(Date dataMovimento) { this.dataMovimento = dataMovimento;
}

@Override
public String toString() {
   return "Movimento [id=" + idMovimento + ", tipo=" + tipo +
         ", produto=" + (produto != null ? produto.getNome() : "null") +
         ", qtd=" + quantidade + "]";
}
}
```

Movimento.java (SERVIDOR)

```
package model;
```

```
import java.io.Serializable;
import java.util.Date;
import javax.persistence.*;
@Entity
@Table(name = "Movimento")
public class Movimento implements Serializable {
private static final long serialVersionUID = 1L;
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "id_movimento")
private Integer idMovimento;

@ManyToOne
@JoinColumn(name = "id_usuario")
private Usuario usuario;

@Column(name = "tipo")
private String tipo;

@ManyToOne
@JoinColumn(name = "id_pessoa")
private Pessoa pessoa;

@ManyToOne
@JoinColumn(name = "id_produto")
private Produto produto;

@Column(name = "quantidade")
private Integer quantidade;

@Column(name = "valor_unitario")
private Double valorUnitario;

@Column(name = "data_movimento")
@Temporal(TemporalType.TIMESTAMP)
private Date dataMovimento;

public Movimento() {}

public Integer getIdMovimento() { return idMovimento; }
public void setIdMovimento(Integer idMovimento) { this.idMovimento = idMovimento; }

public Usuario getUsuario() { return usuario; }
public void setUsuario(Usuario usuario) { this.usuario = usuario; }

public String getTipo() { return tipo; }
public void setTipo(String tipo) { this.tipo = tipo; }

public Pessoa getPessoa() { return pessoa; }
public void setPessoa(Pessoa pessoa) { this.pessoa = pessoa; }
```

```
public Produto getProduto() { return produto; }
public void setProduto(Produto produto) { this.produto = produto; }

public Integer getQuantidade() { return quantidade; }
public void setQuantidade(Integer quantidade) { this.quantidade = quantidade; }

public Double getValorUnitario() { return valorUnitario; }
public void setValorUnitario(Double valorUnitario) { this.valorUnitario = valorUnitario; }

public Date getDataMovimento() { return dataMovimento; }
public void setDataMovimento(Date dataMovimento) { this.dataMovimento = dataMovimento;
}

@Override
public String toString() {
   return "Movimento [id=" + idMovimento + ", tipo=" + tipo +
         ", produto=" + (produto != null ? produto.getNome() : "null") +
         ", qtd=" + quantidade + "]";
}
}
```

ThreadClient.java (CLIENTE)

```
package cadastroclient;
import javax.swing.*;
import java.io.ObjectInputStream;
import java.util.List;
public class ThreadClient extends Thread {
private ObjectInputStream entrada;
private JTextArea textArea;
public ThreadClient(ObjectInputStream entrada, JTextArea textArea) {
   this.entrada = entrada;
   this.textArea = textArea;
}

public void run() {
   try {
      while (true) {
         Object obj = entrada.readObject();
         System.out.println("Objeto recebido: " + obj);
         System.out.println("Classe: " + (obj != null ? obj.getClass() : "null"));
         if (obj instanceof String) {
            String msg = (String) obj;
            SwingUtilities.invokeLater(() -> textArea.append(msg + "\n"));
         } else if (obj instanceof List) {
            List<?> lista = (List<?>) obj;
            StringBuilder sb = new StringBuilder();
            for (Object item : lista) {
               sb.append(item.toString()).append("\n");
```

```
            }
            SwingUtilities.invokeLater(() -> textArea.append(sb.toString()));
        } else {
            SwingUtilities.invokeLater(() -> textArea.append("Objeto desconhecido recebido: " +
obj + "\n"));
        }
      }
   } catch (Exception e) {
      e.printStackTrace();
      SwingUtilities.invokeLater(() -> textArea.append("Conexão encerrada!\n"));
   }
}
}
```

SaidaFrame.java (CLIENTE)

```
package cadastroclient;
import javax.swing.*;
public class SaidaFrame extends JDialog {
public JTextArea texto;
public SaidaFrame() {
   setTitle("Saída do Servidor");
   setBounds(100, 100, 500, 400);
   setModal(false);
   texto = new JTextArea();
JScrollPane scroll = new JScrollPane(texto);
   getContentPane().add(scroll);
   setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
}
```

CadastroClientV2.java (CLIENTE)

```
package cadastroclient;

import java.io.*;
import java.net.Socket;
import javax.swing.*;

public class CadastroClientV2 {
   public static void main(String[] args) {
      try {
         Socket socket = new Socket("localhost", 12345);
         ObjectOutputStream saida = new ObjectOutputStream(socket.getOutputStream());
         ObjectInputStream entrada = new ObjectInputStream(socket.getInputStream());

         SaidaFrame frame = new SaidaFrame();
         frame.setVisible(true);

         ThreadClient thread = new ThreadClient(entrada, frame.texto);
         thread.start();
```

```
        BufferedReader teclado = new BufferedReader(new InputStreamReader(System.in));

        // Login
        System.out.print("Login: ");
        String login = teclado.readLine();
        System.out.print("Senha: ");
        String senha = teclado.readLine();

        saida.writeObject(login);
        saida.writeObject(senha);

        while (true) {
           System.out.print("\nMenu: [L]istar, [E]ntrada, [S]aída, [X] Sair\n");
           String op = teclado.readLine();
           saida.writeObject(op);

           if ("X".equalsIgnoreCase(op)) break;
        }

        saida.close();
        entrada.close();
        socket.close();

     } catch (Exception e) {
        e.printStackTrace();
     }
  }
}
```

MainServer.java (SERVIDOR)

```
package cadastroserver;

import java.net.ServerSocket;
import java.net.Socket;

public class MainServer {
   public static void main(String[] args) {
      try {
         ServerSocket serverSocket = new ServerSocket(12345);
         System.out.println("Servidor iniciado na porta 12345...");
         while (true) {
            Socket socket = serverSocket.accept();
            System.out.println("Cliente conectado: " + socket.getInetAddress());
            // Cria uma thread para tratar o cliente
            new TrataCliente(socket).start();
         }
      } catch (Exception e) {
         e.printStackTrace();
```

```
    }
  }
}
```

TrataCliente.java (SERVIDOR)

```
package cadastroserver;

import java.io.*;
import java.net.Socket;
import java.util.List;
import model.*;

public class TrataCliente extends Thread {
  private Socket socket;
  private ObjectInputStream entrada;
  private ObjectOutputStream saida;

  public TrataCliente(Socket socket) {
    this.socket = socket;
  }

  public void run() {
    try {
      saida = new ObjectOutputStream(socket.getOutputStream());
      entrada = new ObjectInputStream(socket.getInputStream());

      // Login simples (exemplo):
      String login = (String) entrada.readObject();
      String senha = (String) entrada.readObject();
      if (login.equals("op1") && senha.equals("op1")) {
        saida.writeObject("Login OK");
      } else {
        saida.writeObject("Login inválido!");
        socket.close();
        return;
      }

      while (true) {
        String comando = (String) entrada.readObject();
        if (comando.equalsIgnoreCase("L")) {
          // Envia a lista de produtos
          List<Produto> produtos = ProdutoDAO.listarTodos(); // método fictício, veja
abaixo
          saida.writeObject(produtos);
        } else if (comando.equalsIgnoreCase("X")) {
          break;
        } else {
          saida.writeObject("Comando não implementado!");
        }
```

```
        }

      } catch (Exception e) {
        e.printStackTrace();
      } finally {
        try { socket.close(); } catch (Exception ex) { }
      }
    }
}
```

MainMovementserver.java

```
package cadastroserver;

import java.io.*;
import java.net.Socket;
import java.util.List;
import model.*;

public class TrataCliente extends Thread {
   private Socket socket;
   private ObjectInputStream entrada;
   private ObjectOutputStream saida;

   public TrataCliente(Socket socket) {
      this.socket = socket;
   }

   public void run() {
      try {
         saida = new ObjectOutputStream(socket.getOutputStream());
         entrada = new ObjectInputStream(socket.getInputStream());

         // Login simples (exemplo):
         String login = (String) entrada.readObject();
         String senha = (String) entrada.readObject();
         if (login.equals("op1") && senha.equals("op1")) {
            saida.writeObject("Login OK");
         } else {
            saida.writeObject("Login inválido!");
            socket.close();
            return;
         }

         while (true) {
            String comando = (String) entrada.readObject();
            if (comando.equalsIgnoreCase("L")) {
               // Envia a lista de produtos
               List<Produto> produtos = ProdutoDAO.listarTodos(); // método fictício, veja
abaixo
```

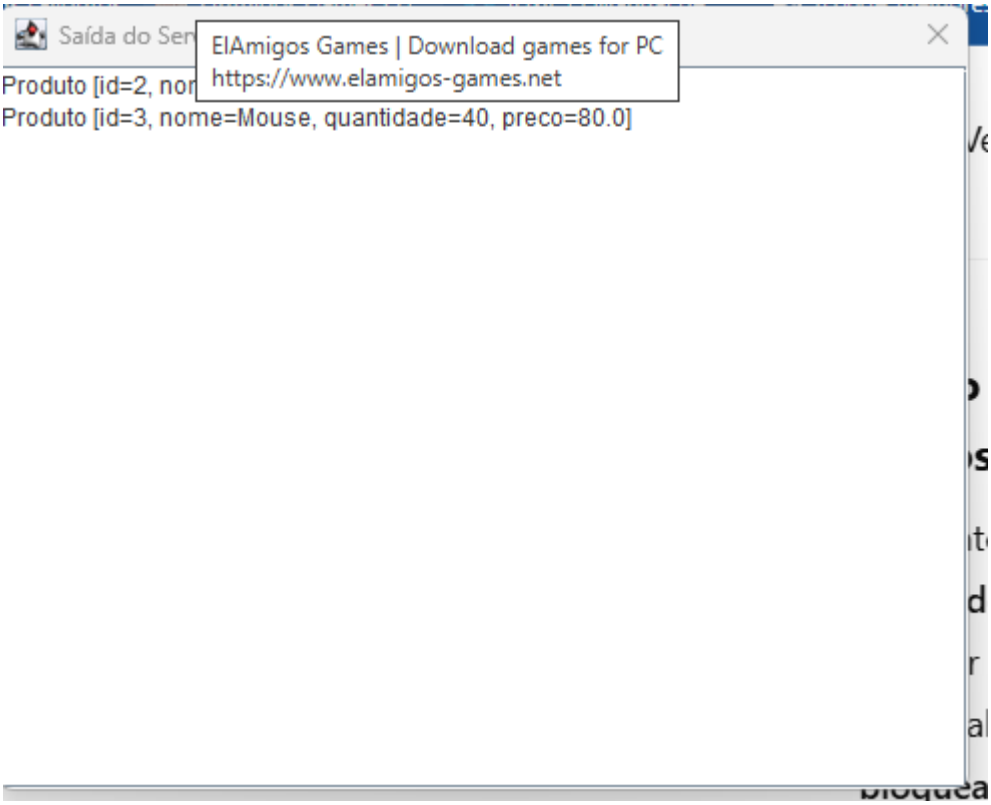
```
                saida.writeObject(produtos);
            } else if (comando.equalsIgnoreCase("X")) {
                break;
            } else {
                saida.writeObject("Comando não implementado!");
            }
        }

    } catch (Exception e) {
        e.printStackTrace();
    } finally {
        try { socket.close(); } catch (Exception ex) { }
    }
  }
}
```

Resultados:.

```
run:
Login: opl
Senha: opl
Login OK

Menu: [L]istar, [E]ntrada, [S]aída, [X] Sair
L
```



Análise e Conclusão – Procedimento 1

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

No contexto de clientes e servidores Java utilizando sockets, **Threads permitem que o cliente fique ouvindo o servidor em paralelo à execução do programa principal**. Ao criar uma Thread dedicada para receber respostas do servidor (exemplo: ThreadClient), o programa principal pode continuar capturando entradas do usuário ou atualizando a interface gráfica **sem ficar bloqueado aguardando dados da rede**. Assim, as respostas do servidor são tratadas de forma assíncrona: a thread escuta o socket e, quando recebe um objeto, processa ou exibe a mensagem **enquanto o cliente segue rodando normalmente**.

Para que serve o método invokeLater, da classe SwingUtilities?

O método invokeLater da classe SwingUtilities serve para **garantir que uma determinada ação (Runnable)** seja executada na **Thread de eventos do Swing** (Event Dispatch Thread, EDT). Como componentes de interface gráfica do Swing **devem ser atualizados apenas pela EDT**, qualquer modificação na interface feita de dentro de threads paralelas (como a thread de rede) precisa ser "agendada" via invokeLater. Isso evita bugs de concorrência e garante o funcionamento correto da interface gráfica.

Como os objetos são enviados e recebidos pelo Socket Java?

Objetos em Java são enviados e recebidos via socket usando as classes ObjectOutputStream (para envio) e ObjectInputStream (para leitura). O objeto deve implementar a interface Serializable. O envio ocorre assim:

- **No cliente/servidor:**

```
ObjectOutputStream saida = new ObjectOutputStream(socket.getOutputStream());

saida.writeObject(objeto);
```

- **Na outra ponta:**

```
ObjectInputStream entrada = new ObjectInputStream(socket.getInputStream());

Object recebido = entrada.readObject();
```

A serialização converte o objeto em bytes, transmite pelo socket, e o destinatário reconstrói o objeto original na memória usando a desserialização.

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

- **Comportamento síncrono:**
  O cliente faz uma requisição ao servidor e **fica bloqueado** (parado, aguardando) até receber a resposta. O método de leitura (readObject, readLine, etc) bloqueia a thread principal, **impedindo que o usuário faça outras ações ou que a interface gráfica responda** enquanto espera a rede.

- **Comportamento assíncrono:**
  O cliente cria uma **Thread separada** só para receber respostas do servidor. Assim, o restante do programa (entrada do usuário, atualização de tela, etc) **segue executando normalmente**, sem bloqueios.
  A thread de rede notifica ou atualiza a interface sempre que uma nova mensagem chegar.