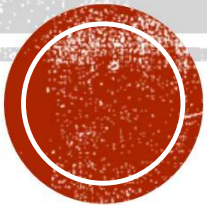
AULA 09 DE 20

12/04/2022



TRANSFERÊNCIA DE OBJETOS

- Como vimos no exercício anterior, é comum em uma transmissão de dados, termos que enviar mais de uma informação.
- Anteriormente enviamos o **nome do usuário** e a **mensagem**, ou nossa **operação matemática simples**, de forma concatenada, utilizando um caractere especial como delimitador.
- Uma outra alternativa seria termos o nome do usuário, a mensagem e outras informações que desejarmos, em um **objeto**, enviando não mais a mensagem pela rede, e sim o objeto com todo seu conteúdo.
- Para isso vamos utilizar o recurso de **serialização de objetos**.
- **Serialização** é a técnica que permite **transformar o estado de um objeto** em uma **sequencia de bytes**.
- Uma vez que temos uma **sequencia de bytes** podemos fazer com ela o que precisarmos, como por exemplo: salvar em banco de dados, salvar em um arquivo, enviar via rede etc etc
- Depois basta **desserializar** para **recriar o objeto na memória** e poder manipular novamente seu conteúdo!

SERIALIZAÇÃO DE OBJETOS EM PYTHON

- Vez por outra precisamos enviar dados via rede, seja através de uma já tradicional conexão HTTP, ou até mesmo através de um socket.
- É bem comum que esses dados estejam representados em nosso programa através de uma instância de uma classe por nós mesmos definida.
- No entanto, na hora de enviar esse objeto pela rede, é preciso que tenhamos esses dados representados de forma contínua e, muitas vezes, de uma forma que possa ser lida por um sistema diferente do sistema no qual o objeto foi criado.
- Para atender a esses requisitos, é necessária a **serialização de dados**, que trata da **representação de objetos ou estruturas de dados em um formato que permita que estas sejam armazenado em um disco (ou enviadas pela rede) para posterior recriação do objeto em memória**.

SERIALIZAÇÃO DE OBJETOS EM PYTHON

- Em **Python**, existem diferentes mecanismos disponíveis para serialização de dados, entre eles:
  - **Pickle**
    - Provê a serialização de objetos Python, transformando objetos quaisquer em sequências de bytes.
    - Possui o melhor desempenho
    - **b'\x80\x03c__main__\nMensagem\nq\x00)\x81q\x01}q\x02(X\x07\x00\x00\x00usuarioq\x03X\x07\x00\x00\x00lucianoq\x04X\x03\x00\x00\x00msgq\x05X\x06\x00\x00\x00coelhoq\x06ub.'**
  - Struct
    - Faz o meio de campo entre objetos Python e estruturas em C.
  - **JSON**
    - Talvez seja hoje o formato para dados intercambiáveis mais utilizado. Esse formato de dados é muito usado em serviços web, e também para o transporte de dados usando outros protocolos.
    - OBS: por padrão não serializa classes criadas pelo usuário!
    - **{"usuario":"luciano", "msg":"coelho"}**
  - Shelve
    - Provê um tipo de dados com uma interface similar a de um dicionário (chamado de shelf), e que agrega a funcionalidade de persistir esse dicionário em um arquivo para uso posterior.
- A escolha vai depender do tipo de aplicação exigida.
- Para nosso uso com sokets, **pickle** e **json** são mais recomendados.

SERIALIZAÇÃO COM PICKLE

- **Pickle** é o formato de serialização de objetos nativo do Python.
- Permite serializar classes criadas pelo usuário nativamente.
- Sua interface provê quatro métodos:
  - **dump()** serializa em um arquivo aberto (objeto do tipo file).
  - **dumps()** serializa para uma cadeia de caracteres.
  - **load()** deserializa de um objeto aberto do tipo file.
  - **loads()** deserializa de uma cadeia de caracteres.

- Pickle suporta, por padrão, um protocolo textual.
- Mas também um protocolo binário, mais eficiente, mas não legível.

TcpServidorPythonPickle.py

```python
import socket, pickle

class Mensagem(object):
    def __init__(self):
        self.usuario = ''
        self.msg     = ''

def Main():
    host = "0.0.0.0"
    port = 10000
    socketTCP = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    socketTCP.bind((host,port))
    socketTCP.listen(1)
    print('Servidor TCP: {}:{}'.format(host,port))

    conn, addr = socketTCP.accept()
    print ("Conexão realizada por: " + str(addr))

    #recebe a mensagem do usuário
    data = conn.recv(4096)
    #mostra a mensagem serializada
    print (data)
    #desserializa a mensagem recebida, disponibilizando o objeto novamente na memoria
    objetoRecebido = pickle.loads(data)
    #mostra os dados do objeto
    print (objetoRecebido.usuario)
    print (objetoRecebido.msg)
    conn.close()

if __name__ == '__main__':
    Main()
```


```python
import socket, pickle

class Mensagem(object):
    def __init__(self):
        self.usuario = ''
        self.msg     = ''

def Main():
    host = '127.0.0.1'
    port = 10000
    mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    mySocket.connect((host,port))

    # cria o objeto
    objEnviar = Mensagem()
    objEnviar.usuario = input("Nome: ")
    objEnviar.msg = input("Mensagem: ")

    # serializa o objeto
    data_string = pickle.dumps(objEnviar)
    # envia o objeto serializado para o servidor
    mySocket.send(data_string)

    mySocket.close()


if __name__ == '__main__':
    Main()
```

TcpClientePythonPickle.py

SERIALIZAÇÃO COM JSON

- O **JSON** (**Javascript Object Notation**) é o formato mais usado para integração entre sistemas.
- Toda a linguagem de programação tem sua maneira de manipulá-lo.
- O Python tem uma biblioteca, chamada **json**, muito boa e simples de usar para lidar com este tipo de formatação de dados.
- A biblioteca json tem 4 métodos básicos, tanto para serializar um objeto, quanto para deserializar-lo.
  - **dump()** serializa em um arquivo aberto (objeto do tipo file).
  - **dumps()** serializa para uma cadeia de caracteres.
  - **load()** deserializa de um objeto aberto do tipo file.
  - **loads()** deserializa de uma cadeia de caracteres.

TcpServidorPythonJSON.py

```python
import socket, json

class Mensagem(object):
    def __init__(self):
        self.usuario = ''
        self.msg     = ''

def Main():
    host = "0.0.0.0"
    port = 10000
    socketTCP = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    socketTCP.bind((host,port))
    socketTCP.listen(1)
    print('Servidor TCP: {}:{}'.format(host,port))
    conn, addr = socketTCP.accept()
    print ("Conexão realizada por: " + str(addr))
    #recebe a mensagem do usuário
    data = conn.recv(4096)
    #mostra a mensagem serializada
    print (data)

    #desserializa a mensagem recebida, disponibilizando o objeto novamente na memoria
    objetoRecebido = json.loads(data)
    print( objetoRecebido )

    #mostra os dados do objeto
    print( objetoRecebido.get("usuario") )
    print( objetoRecebido.get("msg") )
    conn.close()

if __name__ == '__main__':
    Main()
```


```python
import socket, json

class Mensagem(object):
    def __init__(self):
        self.usuario = ""
        self.msg     = ""

def Main():
    host = '127.0.0.1'
    port = 10000
    mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    mySocket.connect((host,port))

    # cria o objeto
    objEnviar = Mensagem()
    objEnviar.usuario = input("Nome: ")
    objEnviar.msg = input("Mensagem: ")

    # serializa o objeto
    data_string = json.dumps(objEnviar.__dict__, indent=0)
    print(data_string)

    # __dict__ é um atributo de guardar atributos de instância nos objetos

    # envia o objeto serializado para o servidor
    mySocket.send( bytes(data_string,encoding="utf-8") )

    mySocket.close()

if __name__ == '__main__':
    Main()
```

TcpClientePythonJSON.py

- https://yzhong-cs.medium.com/serialize-and-deserialize-complex-json-in-python-205ecc636caa
- https://realpython.com/python-json/
- https://realpython.com/lessons/serializing-json-data
- https://stackoverflow.com/questions/47391774/python-send-and-receive-objects-through-sockets
- https://pythonhelp.wordpress.com/2013/07/20/serializacao-de-objetos-em-python/
- https://blog.softhints.com/python-convert-object-to-json-3-examples/
- https://medium.com/@renatojlelis/trabalhando-com-json-no-python-1eb0f97c0c50
- https://stackoverflow.com/questions/10252010/serializing-class-instance-to-json
- https://pythontic.com/serialization/json/introduction
- https://pt.stackoverflow.com/questions/206294/pegar-dados-na-estrutura-json-com-python

EXEMPLO: TRANSFERÊNCIA DE OBJETOS

Documento.java

```java
import java.io.Serializable;

public class Documento implements Serializable {

    private String autor;
    private String titulo;

    public Documento(String titulo, String autor) {
        this.autor = autor;
        this.titulo = titulo;
    }

    public String toString() {
        return titulo + " de " + autor;
    }
}
```

EXEMPLO: TRANSFERÊNCIA DE OBJETOS

```java
import java.io.IOException;
import java.io.ObjectInputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class TcpServidorJavaObjeto {
    public static void main(String args[]) throws ClassNotFoundException, IOException {
        int porta = 2016;
        try (var escuta = new ServerSocket(porta)) {
            System.out.println("*** Servidor ***");
            System.out.println("*** Porta de escuta (listen): " + porta);
            while (true) {
                // accept bloqueia ate que chegue um pedido de conexao de um cliente
                Socket cliente = escuta.accept();
                System.out.println("*** conexao aceita de (remoto): " + cliente.getRemoteSocketAddress());
                // quando chega, cria nova thread para atender em especial o cliente
                ObjectInputStream ois = new ObjectInputStream(cliente.getInputStream());
                while (true) {
                    try {
                        Documento doc = (Documento) ois.readObject();
                        System.out.println(doc.toString());
                    } catch (IOException e) {
                        break;
                    }
                }
            }
        }
    }
}
```

TcpServidorJavaObjeto.java

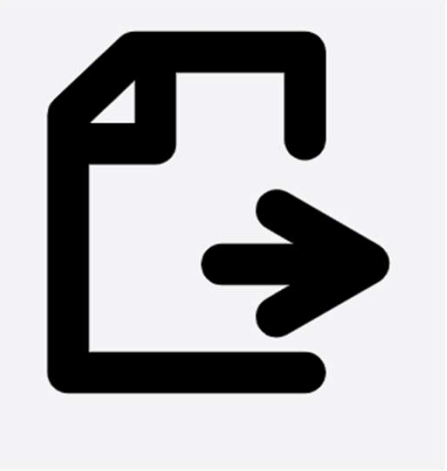
EXEMPLO: TRANSFERÊNCIA DE OBJETOS

TcpClienteJavaObjeto.java

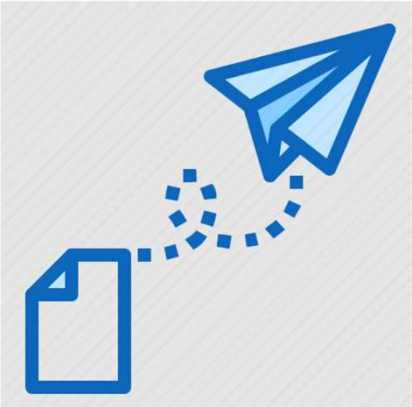
```java
import java.io.EOFException;
import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.net.UnknownHostException;

public class TcpClienteJavaObjeto {
    public static void main(String args[]) {
        Socket s = null;
        try {
            // conecta o socket aa porta remota
            s = new Socket("localhost", 2016);
            ObjectOutputStream oos = new ObjectOutputStream(s.getOutputStream());
            var d1 = new Documento("Memórias de Um Legionário", "Dado Villa-Lobos");
            oos.writeObject(d1);
            var d2 = new Documento("O Contador de Histórias: Memórias de Vida e Música", "Dave Grohl");
            oos.writeObject(d2);

        } catch (UnknownHostException e) {
            System.out.println("!!! Servidor desconhecido: " + e.getMessage());
        } catch (EOFException e) {
            System.out.println("!!! Nao ha mais dados de entrada: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("!!! E/S: " + e.getMessage());
        } finally {
            if (s != null) {
                try {
                    s.close();
                } catch (IOException e) {
                    System.out.println("!!! Encerramento do socket falhou: " + e.getMessage());
                }
            }
        }
    }
}
```

- Com base no exemplo de envio de dados via **objetos serializados**, altere sua implementação de Servidor e Cliente com threads para que os dados sejam enviados via **Objeto**, contendo o **nome** do usuário e a **mensagem**.

- Lembre que um arquivo é um aglomerado de bytes, independente do seu tipo (pdf, vídeo, texto, música, etc.)
- Sendo assim para garantir a integridade do arquivo nós enviaremos seu conteúdo como byte[].



TRANSFERÊNCIA DE ARQUIVOS



- Veremos a seguir um simples exemplo de transferência de arquivos.
- O servidor fica rodando aguardando conexões, ao receber uma nova conexão escolhe e envia para o cliente o arquivo escolhido.

TcpServidorPythonArquivo.py

```python
import socket
import os
from tkinter import filedialog, Tk

def Main():
    host = "0.0.0.0"
    port = 10000
    socketTCP = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    socketTCP.bind((host, port))
    socketTCP.listen(5)

    # ao receber a conexão de um novo cliente, escolhe e envia um arquivo para ele
    while True:
        conn, addr = socketTCP.accept()
        print("Conexão realizada por: {}".format(str(addr)))

        # abre tela para escolha de um arquivo
        root = Tk()
        root.withdraw()
        file_path = filedialog.askopenfilename(
            initialdir="/", title="Escolha um arquivo", filetypes=(("jpeg files", "*.jpg"), ("all files", "*.*")))
        print(file_path)

        # envia mensagem com o nome do arquivo que vai ser enviado
        conn.send(os.path.basename(file_path).encode())

        #abre o arquivo escolhido, e vai enviando para o cliente por partes
        with open( file_path , 'rb') as f:
            conn.sendfile(f, 0)

        conn.close()
        print('Arquivo enviado.')

if __name__ == '__main__':
    Main()
```


```
with open( file_path , 'rb') as f:

Abre o arquivo em binário, e realiza a
leitura, linha a linha, até que seu EOF
seja encontrado
O arquivo é automaticamente fechado!
```


```python
import socket
import os
from tkinter import filedialog

def Main():
    mySocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    servidorDestino = ('127.0.0.1', 10000)
    mySocket.connect(servidorDestino)

    # recebe mensagem com o nome do arquivo que vai receber
    nomeArquivo = mySocket.recv(1024)

    # solicita ao usuário onde deseja salvar o arquivo recebido
    fullpath = filedialog.askdirectory(initialdir=os.getcwd(), title='Salvar em:')

    # cria um novo arquivo binário, vazio, para receber o arquivo enviado
    file = open(fullpath+'/new_'+nomeArquivo.decode(), "wb")

    # Receba as partes do arquivo e monta o arquivo
    RecvData = mySocket.recv(1024)
    while RecvData:
        file.write(RecvData)
        RecvData = mySocket.recv(1024)

    # Feche o arquivo aberto no lado do servidor uma vez que a cópia seja concluída
    file.close()
    print("\n O arquivo foi copiado com sucesso \n")

    mySocket.close()


if __name__ == '__main__':
    Main()
```

TcpClientePythonArquivo.py

TcpServidorJavaArquivo.java

```java
import java.io.*;
import java.net.*;

public class TcpServidorJavaArquivo {
    public static void main(String[] args) throws IOException {
        ServerSocket escuta = new ServerSocket(2022);
        while (true) {
            Socket cliente = escuta.accept();
            System.out.println("Conexão aceita: " + cliente);
            // transforma o arquivo em um byte array
            File meuArquivo = new File("teste.txt");
            byte[] arquivoEmBytes = new byte[(int) meuArquivo.length()];
            FileInputStream fis = new FileInputStream(meuArquivo);
            BufferedInputStream bis = new BufferedInputStream(fis);
            bis.read(arquivoEmBytes, 0, arquivoEmBytes.length);
            // cria o Output e envia o array de bytes do arquivo
            System.out.println("Enviando...");
            OutputStream os = cliente.getOutputStream();
            os.write(arquivoEmBytes, 0, arquivoEmBytes.length);
            os.flush();
            cliente.close();
        }
    }
}
```


```java
import java.io.*;
import java.net.*;

public class TcpClienteJavaArquivo {
    public static void main(String[] args) throws IOException {
        int filesize = 6022386;
        long start = System.currentTimeMillis();
        int bytesRead;
        int current = 0;

        Socket sock = new Socket("127.0.0.1", 2022);
        // recebendo o arquivo
        byte[] arquivoEmBytes = new byte[filesize];
        InputStream is = sock.getInputStream();
        FileOutputStream fos = new FileOutputStream("source-copy.txt");
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        bytesRead = is.read(arquivoEmBytes, 0, arquivoEmBytes.length);
        current = bytesRead;

        do {
            bytesRead = is.read(arquivoEmBytes, current, (arquivoEmBytes.length - current));
            if (bytesRead >= 0) {
                current += bytesRead;
            }
        } while (bytesRead > -1);
        bos.write(arquivoEmBytes, 0, current);
        long end = System.currentTimeMillis();
        System.out.println(end - start);
        bos.close();
        sock.close();
    }
}
```

TcpClienteJavaArquivo.java

FRAMEWORKS PARA GUI DESENVOLVIMENTO DESKTOP COM PYTHON

- **Tkinter** - **https://docs.python.org/3/library/tkinter.html** | **https://www.devmedia.com.br/tkinter-interfaces-graficas-em-python/33956**
  - Faz parte da biblioteca do Python e é instalada junto com o pacote padrão. Sua grande vantagem é a facilidade de uso e grande quantidade de recursos. Por ser nativo da linguagem Python, é muito prático e tudo o que é necessário para começar a usá-lo é importá-lo.
  - Além disso, com o Tkinter, é possível desenvolver programas de maneira muito rápida e simples.
  - Ele é multiplataforma, ou seja, permite o desenvolvimento para Windows, Macs e maioria dos sistemas Unix.
  - Por fim, como foi lançado em 1990, é uma ferramenta estável e que conta com uma grande variedade de extensões.

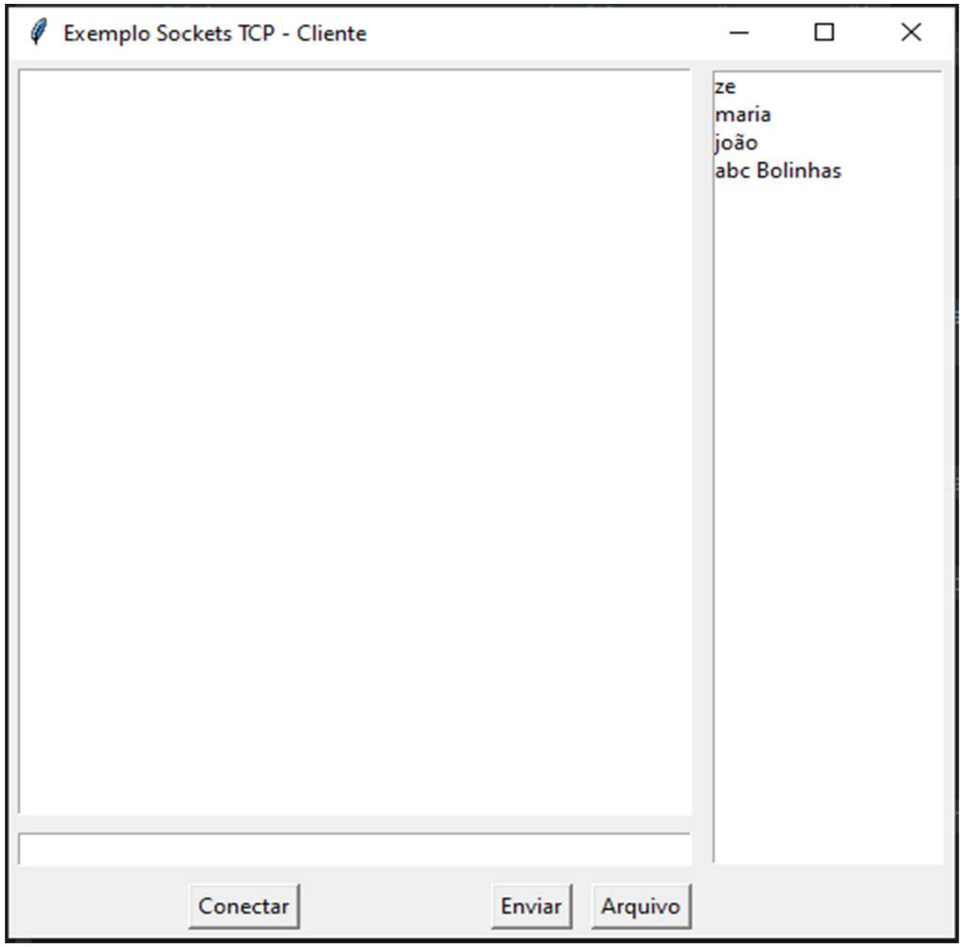
- **Kivy - https://kivy.org/#home | https://pythonacademy.com.br/blog/desenvolva-aplicativos-para-android-ios-com-python-e-kivy**
  - Kivy é a biblioteca mais completa para o desenvolvimento de aplicações multiplataforma em Python. Com ela, você pode desenvolver não apenas para desktop, mas também para mobile.
  - Por ser multiplataforma e compatível com desenvolvimento mobile e desktop, com apenas um código é possível gerar 4 versões executáveis, para rodar em iOS, Android, Windows e OSX.
  - Ela possui código aberto e segue o padrão NUI – Natural User Interface, que é muito parecido com as interfaces que costumamos usar no dia a dia. Possui versões próprias para botões, rótulos de textos, formulários e assim por diante, o que garante consistência e portabilidade para o seu aplicativo.
  - Assim como o Python, a Kivy possui sua própria filosofia que pode ser resumida em algumas palavras-chave: moderna, rápida, flexível, focada, comunidade e gratuita. Ela também tem a sua própria linguagem para organização e estruturação, a Kivy Language. Com a Kylanguage, você pode organizar a árvore de widgets utilizados e determinar propriedades e funções entre eles, separando a lógica da programação da interface do usuário.

- **wxPython**
  - É um conjunto de ferramentas para GUI com plataforma cruzada. Ele permite o desenvolvimento de programas com uma interface gráfica robusta e altamente funcional de maneira simples e fácil. Ele é uma plataforma cruzada, o que significa que o mesmo programa vai rodar bem sem precisar de modificações em diferentes plataformas, como Windows, Mac OS X e Linux.
  - Escrito em código aberto, o wxPython deve ser implementado como um módulo de expansão da biblioteca nativa do Python e engloba a popular plataforma cruzada wxWidgets, que é escrita em C++.

- Além dos listados acima, temos muitos outros: PyGTK, PySide, PyQt, Qt etc.

```python
from tkinter import *
from tkinter import messagebox

class TelaAplicacao(Frame):

    def __init__(self):
        super().__init__()

        self.master.title("Exemplo Sockets TCP - Cliente")
        self.pack(fill=BOTH, expand=True)

        self.columnconfigure(0, weight=1)
        self.rowconfigure(0, weight=1)

        self.textMsgRecebida = Text(self)
        self.textMsgRecebida.grid(row=0, column=0, columnspan=3, rowspan=1, padx=5, pady=5, sticky=E+W+S+N)

        self.lbConectados = Listbox(self)
        self.lbConectados.insert(1, 'ze')
        self.lbConectados.insert(2, 'maria')
        self.lbConectados.insert(3, 'joão')
        self.lbConectados.insert(4, 'abc Bolinhas')
        self.lbConectados.grid(row=0, column=4, columnspan=1, rowspan=2, padx=5, pady=5, sticky=E+W+S+N)

        self.entryMsgEnviar = Entry(self)
        self.entryMsgEnviar.grid(row=1, column=0, columnspan=3, rowspan=1, padx=5, pady=5, sticky=E+W+S+N)


        self.buttonConectar = Button(self, text="Conectar")
        self.buttonConectar.grid(row=2, column=0, padx=5, pady=5 )
        self.buttonConectar["command"] = self.conectar

        self.buttonEnviar = Button(self, text="Enviar")
        self.buttonEnviar.grid(row=2, column=1, padx=5, pady=5 )
        self.buttonEnviar["command"] = self.enviarMensagem

        self.buttonEnviarArquivo = Button(self, text="Arquivo")
        self.buttonEnviarArquivo.grid(row=2, column=2, padx=5, pady=5 )
        self.buttonEnviarArquivo["command"] = self.enviarArquivo


    def conectar(self):
        messagebox.showinfo("Conectar", "implemente as rotinas para conectar")

    def enviarMensagem(self):
        messagebox.showerror("Enviar Mensagem", "implemente as rotinas para enviar mensagem")

        teste = self.entryMsgEnviar.get()
        self.entryMsgEnviar.delete(0, END)
        self.textMsgRecebida.insert(END, "\n"+teste)

    def enviarArquivo(self):
        messagebox.showwarning("Enviar Arquivo", "implemente as rotinas para enviar arquivo")
```

```python
def main():

    root = Tk()
    root.geometry("500x500")
    app = TelaAplicacao()
    root.mainloop()


if __name__ == '__main__':
    main()
```



AVALIAÇÃO 02 – PARCIAL DE NOTA
LABORATÓRIOS PRÁTICOS (PROCESSUAL E CONTINUA)
5 PARCIAIS DE NOTA

- Utilizando os conceitos estudados sobre **Sockets TCP**, implemente uma solução de troca de mensagens atendendo os seguintes requisitos:
  - **Cliente** deve possuir **interface gráfica** contendo:
    - Área para **mensagens recebidas**
    - Área com os **clientes conectados**
      - Atualizada automaticamente toda vez que um cliente conectar ou desconectar
    - Área de **envio de texto** e/ou **arquivo**
    - Ao receber um arquivo o mesmo deve ser salvo em disco com o nome e extensão original, dando ao usuário somente a opção de escolher em qual diretório deseja salvar
    - Opções para **conectar, desconectar, enviar texto, enviar arquivo** etc
    - O cliente pode enviar mensagens/arquivos para **todos**, ou então **escolher na lista de conectados** para qual usuário quer enviar uma **mensagem privada**

  - **Servidor** deve manter **LOG** com os seguintes dados:
    - Data; hora; IP remetente; Nome remetente; IP[s] destinatário[s]; Nome[s] destinatário[s]; Ação
    - Ações do LOG:
      - login
      - logoff
      - msg:mensagem
      - arq:nomeArquivo
    - Exemplo:
      - 06/04/2022; 12:54; 192.168.10.50; luciano; 200.10.10.10; servidor; login
      - 06/04/2022; 13:03; 192.168.10.50; luciano; 172.16.10.87-200.20.32.12-190.200.232.9; zé-maria-joão; msg:olá, tudo bem?
      - 06/04/2022; 14:10; 172.16.10.87; zé; 192.168.10.50; luciano; arq:abcBolinhas.txt

  - **Servidor** deve possuir uma interface gráfica que permita visualizar em tempo real os **clientes conectados** e as entradas no **arquivo de LOG**.

- Threads
- Serialização de objetos
- Utilize uma espécie de chave para indicar o tipo de conteúdo/ação que esta ocorrendo, por exemplo:
  - **N**, novo cliente
  - **NN**, nome novo cliente
  - **M**, mensagem de texto
  - **NA**, nome do arquivo
  - **A**, arquivo
  - **S**, sair/desconectar
  - **etc**

AVALIAÇÃO 02 – PARCIAL DE NOTA
LABORATÓRIOS PRÁTICOS (PROCESSUAL E CONTINUA)
5 PARCIAIS DE NOTA

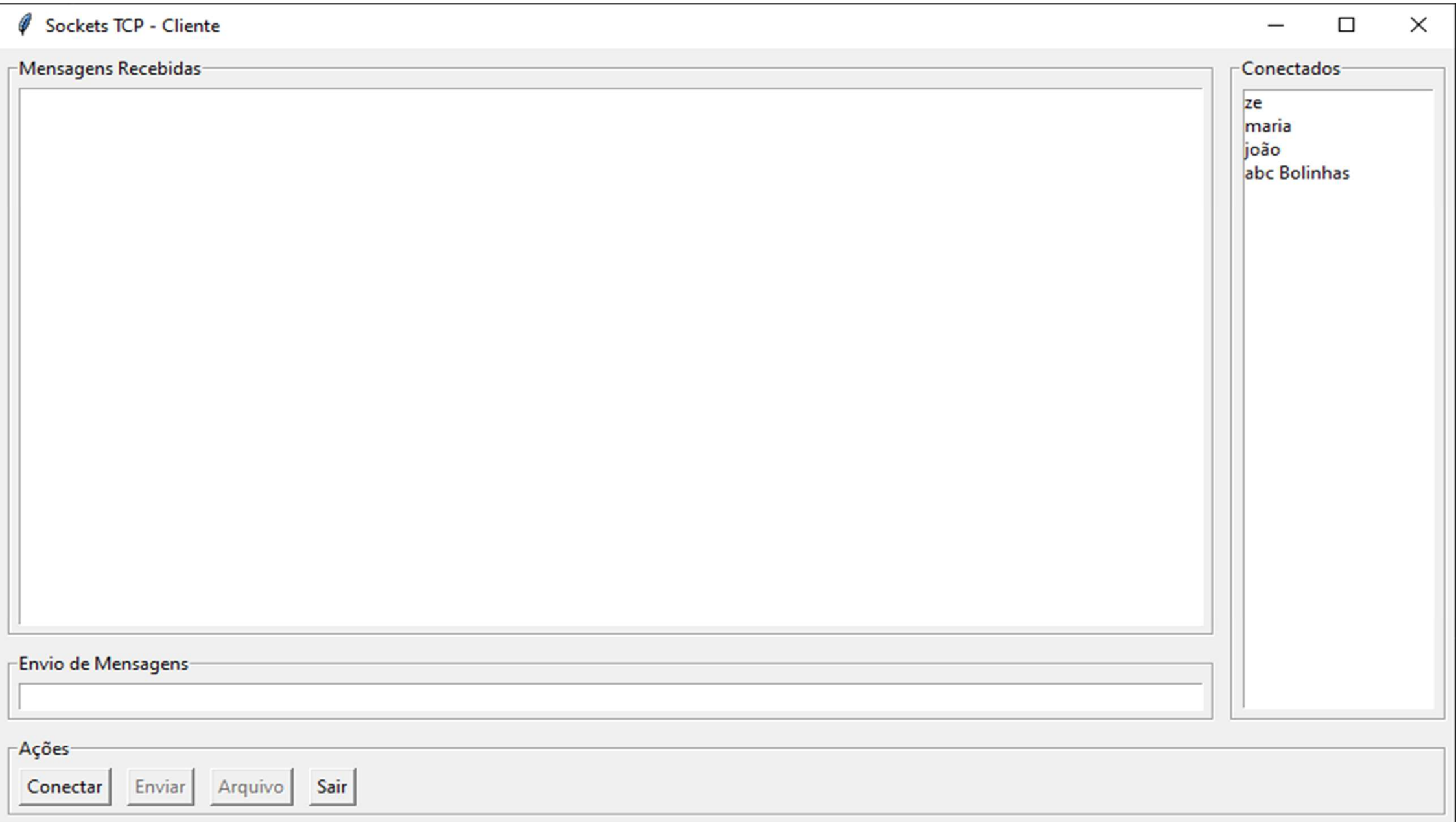
- A linguagem de programação: **LIVRE**!

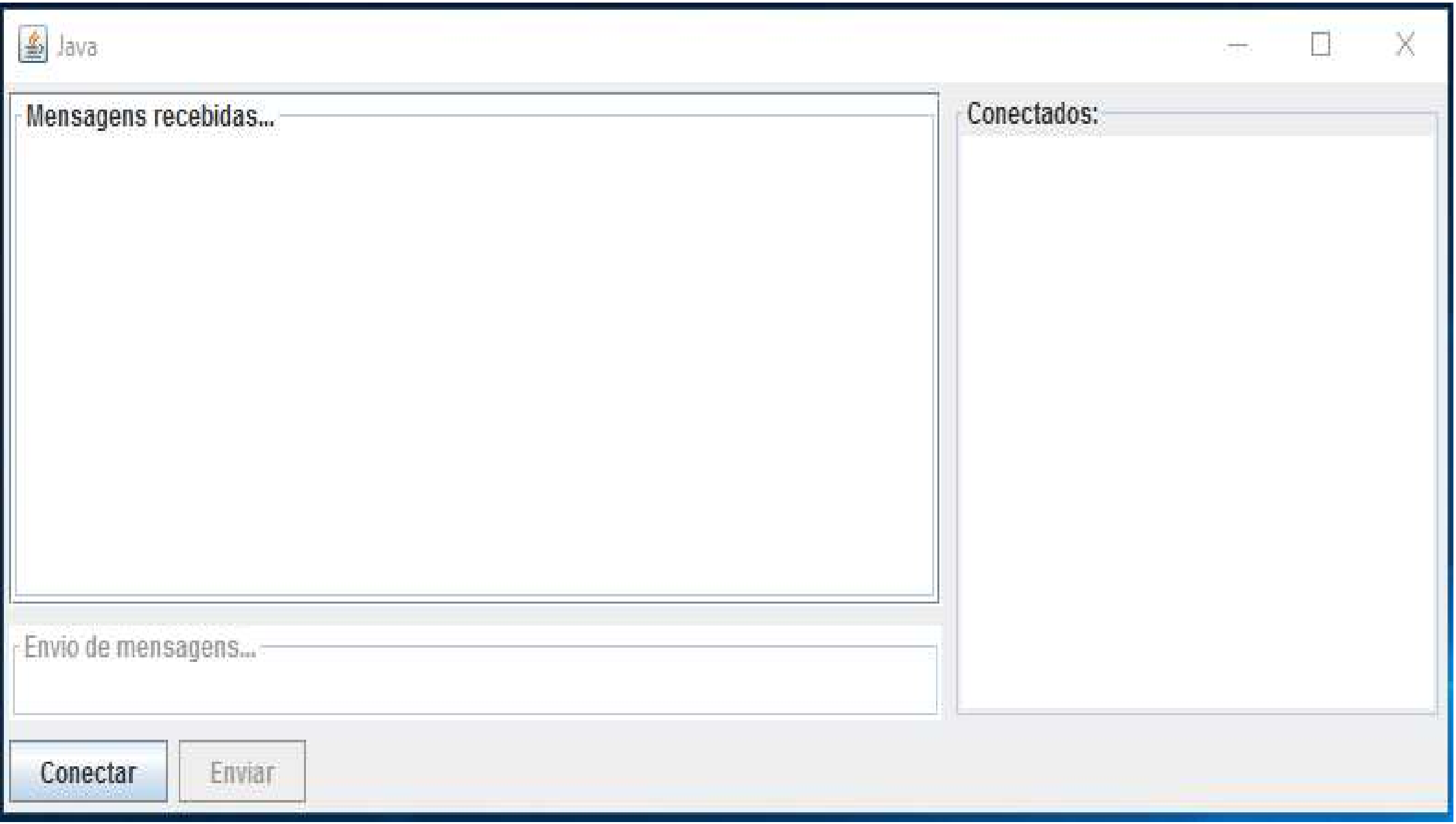
- Data de entrega:
  - **24/04/2022 23:59**

- Data de entrega como recuperação:
  - **26/04/2022 18:40**

- Deve ser postado na atividade correspondente os seguintes arquivos:
  - Vídeo demonstrando o funcionamento, devendo aparecer a tela do servidor e pelo menos três clientes conectados. Todas as funcionalidades devem ser apresentadas no vídeo!
  - Todos os fontes desenvolvidos!
  - Arquivo descrevendo as tecnologias e versões utilizadas para o desenvolvimento.

- Na aula do dia 19/04/2022 somente postarei um material complementar sobre sockets multicast, ficando a aula livre para que as **DUPLAS** trabalhem no desenvolvimento da aplicação.


Sockets TCP - Cliente
Mensagens Recebidas
Conectados
ze
maria
joão
abc Bolinhas
Envio de Mensagens
Ações
Conectar
Enviar
Arquivo
Sair


Java
Mensagens recebidas...
Conectados:
Envio de mensagens...
Conectar
Enviar