DPA – Lab 2

import *java.io.Serializable*;

*public* *class* Request *implements* Serializable {

*private* *static* *final* long serialVersionUID = 1L;

*public* *enum* Type {

        UPLOAD, SEARCH, DISCONNECT

    }

*private* Type type;

*private* ImageData imageData;

*private* String keyword;

*public* Request(Type type) {

        this.type = type;

    }

*public* Type getType() {

        return type;

    }

*public* ImageData getImageData() {

        return imageData;

    }

*public* void setImageData(ImageData imageData) {

        this.imageData = imageData;

    }

*public* String getKeyword() {

        return keyword;

    }

*public* void setKeyword(String keyword) {

        this.keyword = keyword;

    }

}

import *java.io.Serializable*;

import *java.util.Arrays*;

*public* *class* ImageData *implements* Serializable {

*private* *static* *final* long serialVersionUID = 1L;

*private* String fileName;

*private* byte[] imageData;

*private* String[] keywords;

*public* ImageData(String fileName, byte[] imageData, String[] keywords) {

        this.fileName = fileName;

        this.imageData = imageData;

        this.keywords = keywords;

    }

*public* String getFileName() {

        return fileName;

    }

*public* byte[] getImageData() {

        return imageData;

    }

*public* String[] getKeywords() {

        return keywords;

    }

    @Override

*public* String toString() {

        return "ImageData{" +

                "fileName='" + fileName + '\'' +

                ", keywords=" + Arrays.toString(keywords) +

                ", imageSize=" + (imageData != null ? imageData.length : 0) + " bytes}";

    }

}

import *java.io.\**;

import *java.net.\**;

import *java.nio.file.\**;

import *javax.swing.\**;

import *java.awt.\**;

import *java.awt.event.\**;

import *java.util.List*;

*public* *class* Client {

*private* *static* *final* String SERVER\_ADDRESS = "localhost";

*private* *static* *final* int SERVER\_PORT = 9000;

*private* Socket socket;

*private* ObjectOutputStream out;

*private* ObjectInputStream in;

*private* JFrame frame;

*private* JPanel resultsPanel;

*public* *static* void main(String[] args) {

        SwingUtilities.invokeLater(() -> {

            try {

                new Client().createAndShowGUI();

            } catch (Exception e) {

                JOptionPane.showMessageDialog(null, "Error starting client: " + e.getMessage(),

                                             "Error", JOptionPane.ERROR\_MESSAGE);

                e.printStackTrace();

            }

        });

    }

*public* Client() *throws* IOException {

        connectToServer();

    }

*private* void connectToServer() *throws* IOException {

        socket = new Socket(SERVER\_ADDRESS, SERVER\_PORT);

        out = new ObjectOutputStream(socket.getOutputStream());

        in = new ObjectInputStream(socket.getInputStream());

        System.out.println("Connected to server at " + SERVER\_ADDRESS + ":" + SERVER\_PORT);

    }

*private* void createAndShowGUI() {

        frame = new JFrame("Image Sharing");

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        frame.addWindowListener(new WindowAdapter() {

            @Override

*public* void windowClosing(WindowEvent e) {

                disconnectFromServer();

            }

        });

        JPanel mainPanel = new JPanel(new BorderLayout(10, 10));

        mainPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

*final* File[] selectedImage = new File[1];

        JPanel uploadPanel = new JPanel();

        uploadPanel.setLayout(new BoxLayout(uploadPanel, BoxLayout.Y\_AXIS));

        uploadPanel.setBorder(BorderFactory.createTitledBorder("Upload Image"));

        JPanel selectImagePanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

        JLabel fileLabel = new JLabel("Image File:");

        JButton selectImageBtn = new JButton("Select Image");

        JLabel selectedFileLabel = new JLabel("No file selected");

        selectedFileLabel.setForeground(Color.GRAY);

        selectImagePanel.add(fileLabel);

        selectImagePanel.add(selectImageBtn);

        selectImagePanel.add(selectedFileLabel);

        JPanel keywordsPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

        JLabel keywordsLabel = new JLabel("Keywords:");

        JTextField keywordsField = new JTextField(30);

        keywordsPanel.add(keywordsLabel);

        keywordsPanel.add(keywordsField);

        JPanel uploadButtonPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

        JButton uploadBtn = new JButton("Upload Image");

        uploadButtonPanel.add(uploadBtn);

        uploadPanel.add(selectImagePanel);

        uploadPanel.add(keywordsPanel);

        uploadPanel.add(uploadButtonPanel);

        JPanel searchPanel = new JPanel();

        searchPanel.setLayout(new BoxLayout(searchPanel, BoxLayout.Y\_AXIS));

        searchPanel.setBorder(BorderFactory.createTitledBorder("Search Images"));

        JPanel searchInputPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

        JLabel searchLabel = new JLabel("Search by Keyword:");

        JTextField searchField = new JTextField(30);

        JButton searchBtn = new JButton("Search");

        searchInputPanel.add(searchLabel);

        searchInputPanel.add(searchField);

        searchInputPanel.add(searchBtn);

        searchPanel.add(searchInputPanel);

        JPanel controlPanel = new JPanel(new BorderLayout(0, 10));

        controlPanel.add(uploadPanel, BorderLayout.NORTH);

        controlPanel.add(searchPanel, BorderLayout.CENTER);

        JPanel resultsContainer = new JPanel(new BorderLayout());

        resultsContainer.setBorder(BorderFactory.createTitledBorder("Search Results"));

        resultsPanel = new JPanel();

        resultsPanel.setLayout(new BoxLayout(resultsPanel, BoxLayout.Y\_AXIS));

        JScrollPane scrollPane = new JScrollPane(resultsPanel);

        scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL\_SCROLLBAR\_ALWAYS);

        scrollPane.setPreferredSize(new Dimension(750, 400));

        resultsContainer.add(scrollPane, BorderLayout.CENTER);

        mainPanel.add(controlPanel, BorderLayout.NORTH);

        mainPanel.add(resultsContainer, BorderLayout.CENTER);

        selectImageBtn.addActionListener(e -> {

            JFileChooser fileChooser = new JFileChooser();

            fileChooser.setFileFilter(new javax.swing.filechooser.FileFilter() {

*public* boolean accept(File f) {

                    return f.isDirectory() || f.getName().toLowerCase().endsWith(".jpg") ||

                           f.getName().toLowerCase().endsWith(".jpeg") ||

                           f.getName().toLowerCase().endsWith(".png");

                }

*public* String getDescription() {

                    return "Image Files (\*.jpg, \*.jpeg, \*.png)";

                }

            });

            int result = fileChooser.showOpenDialog(frame);

            if (result == JFileChooser.APPROVE\_OPTION) {

                selectedImage[0] = fileChooser.getSelectedFile();

                selectedFileLabel.setText(selectedImage[0].getName());

                selectedFileLabel.setForeground(Color.BLACK);

            }

        });

        uploadBtn.addActionListener(e -> {

            if (selectedImage[0] == null) {

                JOptionPane.showMessageDialog(frame, "Please select an image first");

                return;

            }

            String keywordsText = keywordsField.getText().trim();

            if (keywordsText.isEmpty()) {

                JOptionPane.showMessageDialog(frame, "Please enter at least one keyword");

                return;

            }

            String[] keywords = keywordsText.split(",");

            for (int i = 0; i < keywords.length; i++) {

                keywords[i] = keywords[i].trim();

            }

            try {

                byte[] imageData = Files.readAllBytes(selectedImage[0].toPath());

                ImageData img = new ImageData(selectedImage[0].getName(), imageData, keywords);

                Request request = new Request(Request.Type.UPLOAD);

                request.setImageData(img);

                out.writeObject(request);

                out.flush();

                Response response = (Response) in.readObject();

                if (response.getStatus() == Response.Status.SUCCESS) {

                    JOptionPane.showMessageDialog(frame, "Image uploaded successfully");

                    keywordsField.setText("");

                    selectedFileLabel.setText("No file selected");

                    selectedFileLabel.setForeground(Color.GRAY);

                    selectedImage[0] = null;

                } else {

                    JOptionPane.showMessageDialog(frame, "Upload failed: " + response.getMessage());

                }

            } catch (IOException | ClassNotFoundException ex) {

                JOptionPane.showMessageDialog(frame, "Error uploading image: " + ex.getMessage());

                ex.printStackTrace();

            }

        });

        searchBtn.addActionListener(e -> {

            String keyword = searchField.getText().trim();

            if (keyword.isEmpty()) {

                JOptionPane.showMessageDialog(frame, "Please enter a search keyword");

                return;

            }

            try {

                Request request = new Request(Request.Type.SEARCH);

                request.setKeyword(keyword);

                out.writeObject(request);

                out.flush();

                Response response = (Response) in.readObject();

                if (response.getStatus() == Response.Status.SUCCESS) {

                    displaySearchResults((List<ImageData>) response.getResults());

                } else {

                    JOptionPane.showMessageDialog(frame, "Search failed: " + response.getMessage());

                }

            } catch (IOException | ClassNotFoundException ex) {

                JOptionPane.showMessageDialog(frame, "Error searching: " + ex.getMessage());

                ex.printStackTrace();

            }

        });

*// Finalize and display frame*

        frame.getContentPane().add(mainPanel);

        frame.pack();

        frame.setSize(800, 700);

        frame.setLocationRelativeTo(null);

        frame.setVisible(true);

    }

    @SuppressWarnings("unchecked")

*private* void displaySearchResults(List<ImageData> results) {

        SwingUtilities.invokeLater(() -> {

            resultsPanel.removeAll();

            if (results.isEmpty()) {

                JPanel noResultsPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));

                JLabel noResults = new JLabel("No images found for this keyword");

                noResults.setFont(new Font("Arial", Font.ITALIC, 14));

                noResultsPanel.add(noResults);

                resultsPanel.add(noResultsPanel);

            } else {

                JLabel countLabel = new JLabel("Found " + results.size() + " images");

                countLabel.setFont(new Font("Arial", Font.BOLD, 14));

                countLabel.setBorder(BorderFactory.createEmptyBorder(5, 5, 10, 5));

                resultsPanel.add(countLabel);

                for (ImageData img : results) {

                    JPanel imagePanel = new JPanel(new BorderLayout(5, 5));

                    imagePanel.setBorder(BorderFactory.createCompoundBorder(

                        BorderFactory.createEmptyBorder(5, 5, 15, 5),

                        BorderFactory.createLineBorder(Color.LIGHT\_GRAY, 1)

                    ));

                    try {

                        ImageIcon icon = new ImageIcon(img.getImageData());

                        int maxWidth = 500;

                        int maxHeight = 800;

                        int origWidth = icon.getIconWidth();

                        int origHeight = icon.getIconHeight();

                        int width = origWidth;

                        int height = origHeight;

                        if (width > maxWidth) {

                            width = maxWidth;

                            height = (int)((double)maxWidth / origWidth \* origHeight);

                        }

                        if (height > maxHeight) {

                            height = maxHeight;

                            width = (int)((double)maxHeight / origHeight \* width);

                        }

                        Image scaledImage = icon.getImage().getScaledInstance(width, height, Image.SCALE\_SMOOTH);

                        JLabel imageLabel = new JLabel(new ImageIcon(scaledImage));

                        imageLabel.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));

                        imageLabel.setHorizontalAlignment(JLabel.CENTER);

                        JPanel infoPanel = new JPanel();

                        infoPanel.setLayout(new BoxLayout(infoPanel, BoxLayout.Y\_AXIS));

                        infoPanel.setBorder(BorderFactory.createEmptyBorder(5, 10, 10, 10));

                        JLabel titleLabel = new JLabel("File: " + img.getFileName());

                        titleLabel.setAlignmentX(Component.LEFT\_ALIGNMENT);

                        JLabel keywordsLabel = new JLabel("Keywords: " + String.join(", ", img.getKeywords()));

                        keywordsLabel.setAlignmentX(Component.LEFT\_ALIGNMENT);

                        infoPanel.add(titleLabel);

                        infoPanel.add(Box.createRigidArea(new Dimension(0, 5)));

                        infoPanel.add(keywordsLabel);

                        infoPanel.add(Box.createRigidArea(new Dimension(0, 5)));

                        imagePanel.add(imageLabel, BorderLayout.CENTER);

                        imagePanel.add(infoPanel, BorderLayout.SOUTH);

                        resultsPanel.add(imagePanel);

                    } catch (Exception e) {

                        JLabel errorLabel = new JLabel("Error displaying image: " + e.getMessage());

                        imagePanel.add(errorLabel, BorderLayout.CENTER);

                        resultsPanel.add(imagePanel);

                        System.err.println("Error displaying image: " + e.getMessage());

                        e.printStackTrace();

                    }

                }

            }

            resultsPanel.revalidate();

            resultsPanel.repaint();

        });

    }

*private* void disconnectFromServer() {

        try {

            if (socket != null && !socket.isClosed()) {

                Request disconnectRequest = new Request(Request.Type.DISCONNECT);

                out.writeObject(disconnectRequest);

                out.flush();

                socket.close();

                System.out.println("Disconnected from server");

            }

        } catch (IOException e) {

            System.err.println("Error disconnecting: " + e.getMessage());

            e.printStackTrace();

        }

    }

}

import *java.io.Serializable*;

import *java.util.List*;

*public* *class* Response *implements* Serializable {

*private* *static* *final* long serialVersionUID = 1L;

*public* *enum* Status {

        SUCCESS, ERROR

    }

*private* Status status;

*private* String message;

*private* List<ImageData> results;

*public* Response(Status status, String message) {

        this.status = status;

        this.message = message;

    }

*public* Response(Status status, List<ImageData> results) {

        this.status = status;

        this.results = results;

    }

*public* Status getStatus() {

        return status;

    }

*public* String getMessage() {

        return message;

    }

*public* List<ImageData> getResults() {

        return results;

    }

}

import *java.io.\**;

import *java.net.\**;

import *java.util.\**;

import *java.util.concurrent.\**;

*public* *class* Server {

*private* *static* *final* int PORT = 9000;

*private* *final* Map<String, List<ImageData>> imageDatabase = new ConcurrentHashMap<>();

*public* *static* void main(String[] args) {

        new Server().startServer();

    }

*public* void startServer() {

        System.out.println("Server starting on port " + PORT);

        try (ServerSocket serverSocket = new ServerSocket(PORT)) {

            while (true) {

                Socket clientSocket = serverSocket.accept();

                System.out.println("New client connected: " + clientSocket.getInetAddress());

                new Thread(new ClientHandler(clientSocket)).start();

            }

        } catch (IOException e) {

            System.err.println("Server error: " + e.getMessage());

        }

    }

*private* *class* ClientHandler *implements* Runnable {

*private* *final* Socket clientSocket;

*public* ClientHandler(Socket socket) {

            this.clientSocket = socket;

        }

        @Override

*public* void run() {

            try (

                ObjectInputStream in = new ObjectInputStream(clientSocket.getInputStream());

                ObjectOutputStream out = new ObjectOutputStream(clientSocket.getOutputStream())

            ) {

                while (true) {

                    Request request = (Request) in.readObject();

                    if (request.getType() == Request.Type.UPLOAD) {

                        handleUpload(request, out);

                    } else if (request.getType() == Request.Type.SEARCH) {

                        handleSearch(request, out);

                    } else if (request.getType() == Request.Type.DISCONNECT) {

                        break;

                    }

                }

            } catch (EOFException e) {

*// Client disconnected*

                System.out.println("Client disconnected");

            } catch (IOException | ClassNotFoundException e) {

                System.err.println("Error handling client: " + e.getMessage());

            } finally {

                try {

                    clientSocket.close();

                } catch (IOException e) {

                    System.err.println("Error closing socket: " + e.getMessage());

                }

            }

        }

*private* void handleUpload(Request request, ObjectOutputStream out) *throws* IOException {

            ImageData imageData = request.getImageData();

            for (String keyword : imageData.getKeywords()) {

                keyword = keyword.toLowerCase();

                imageDatabase.computeIfAbsent(keyword, k -> new CopyOnWriteArrayList<>())

                             .add(imageData);

            }

            Response response = new Response(Response.Status.SUCCESS, "Image uploaded successfully");

            out.writeObject(response);

            out.flush();

            System.out.println("Image uploaded with keywords: " + String.join(", ", imageData.getKeywords()));

        }

*private* void handleSearch(Request request, ObjectOutputStream out) *throws* IOException {

            String keyword = request.getKeyword().toLowerCase();

            List<ImageData> results = imageDatabase.getOrDefault(keyword, Collections.emptyList());

            Response response = new Response(Response.Status.SUCCESS, results);

            out.writeObject(response);

            out.flush();

            System.out.println("Search performed for keyword '" + keyword +

                               "', found " + results.size() + " results");

        }

    }

}