|  |  |
| --- | --- |
| Name: Moutasem Mohamed Saleh Mohamed | 23CSCI07I |
| ID: 219140 | Assignment one |
| AI – A2 | 2023-2024 |

A close-up of a logo

Description automatically generated

**Distributed Systems**

As per the provided instructions, herein is the peer-to-peer conversation application designed to facilitate communication among four nodes within the network. A coordinator entity has been incorporated to enable the transmission of both public and private messages among the nodes.

# Diagram:

# GUI for Node:

# A screenshot of a computer Description automatically generated

After pressing Private button:

# A screenshot of a message Description automatically generated

Coordinator GUI:

A screenshot of a computer

Description automatically generated

Code Implementation for Coordination:

1. **Imports**:
   * It imports necessary classes for networking (**java.io.\***, **java.net.\***) and collections (**java.util.ArrayList**, **java.util.List**).
2. **Coordinator Class**:
   * This class represents the coordinator in a peer-to-peer communication system.
   * It manages connections with multiple nodes and facilitates message exchange.
3. **Fields**:
   * **nodeConnections**: A list to store connections with nodes.
   * **c\_GUI**: An instance of **CoordinatorGUI** for displaying messages.
4. **NodeConnection Class**:
   * This is an inner class representing a connection with a node.
   * It holds information about a node's socket, name, input and output streams, and connection status.
5. **Constructor**:
   * The constructor initializes the fields of the **NodeConnection** class.
6. **Main Method**:
   * The **main** method is the entry point of the program.
   * It initializes the **CoordinatorGUI**, starts listening for incoming connections on port 2000, and handles incoming connections.
   * For each incoming connection, a **NodeConnection** instance is created, added to **nodeConnections**, and a new thread is started to handle message reception from the node.
7. **receiveMessages Method**:
   * This method is responsible for receiving messages from a node.
   * It continuously reads messages from the node's input stream and processes them.
   * Messages are printed to the console and appended to the GUI.
   * If a node leaves or joins, appropriate actions are taken.
8. **broadcastMessage Method**:
   * This method broadcasts a message to all connected nodes.
   * It iterates through **nodeConnections** and sends the message to each connected node.
   * If sending fails, the node is removed from the list of connections.
9. **removeNode Method**:
   * This method removes a disconnected node from the list of connections.
   * It removes the node from **nodeConnections** and updates the GUI.
10. **sendRingNodesList Method**:
    * This method sends a list of connected nodes to a newly joined node.
    * It sends a message containing information about connected nodes to the specified node.
    * If sending fails, an error message is printed to the console and appended to the GUI.

Top of Form

Code Implementation for Node: ( All 4 nodes share same concept but has different portName and different privatePortNumber):

1. **Imports**:
   * The code imports classes for handling networking, I/O operations, logging, and GUI components.
2. **Node11 Class**:
   * This class represents a node in a peer-to-peer communication system.
   * It manages connections with other nodes, sends and receives messages, and provides a graphical user interface (GUI) for interaction.
3. **Fields**:
   * **PRIVATE\_MESSAGE\_PORT**: A constant defining the port for private messaging.
   * **privateMessageSocket**: A **DatagramSocket** for handling private messages.
   * **nodeName**: A string representing the name of the node.
   * **RecieverNodeName**: A string representing the name of the receiver node.
   * **serverHostname** and **serverPort**: Variables defining the hostname and port of the coordinator server.
   * **socket**, **input**, **output**: Objects for handling socket connections and I/O streams.
4. **Constructor**:
   * Initializes the socket connection with the coordinator server and sets up input and output streams.
5. **Utility Methods**:
   * **appendTextArea**: Appends text to the text area of the GUI.
   * **encryptMessage** and **decryptMessage**: Methods for encrypting and decrypting messages using a simple character shifting technique.
6. **Message Handling Methods**:
   * **receiveMessages**: Receives messages from the coordinator server and appends them to the GUI text area.
   * **handlePrivateMessages**: Handles incoming private messages by decrypting them and displaying them in the GUI.
   * **sendPrivateMessage**: Sends a private message to another node after encrypting it.
7. **GUI Initialization Method**:
   * **initComponents**: Initializes the graphical components of the GUI.
8. **Action Listeners**:
   * **jTextField1ActionPerformed**, **jButton4ActionPerformed**, **jButton1ActionPerformed**, **jButton3ActionPerformed**, **jButton2ActionPerformed**: Event handling methods for button clicks and text field actions.
9. **Main Method**:
   * Entry point of the program.
   * Sets the look and feel of the GUI.
   * Creates an instance of **Node11** and makes it visible.
10. **Variables Declaration**:
    * Declarations of GUI components and other variables.