

ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY

COLLEGE OF ELECTRICAL AND MECHANICAL ENGINEERING

DEPARTMENT OF SOFTWARE ENGINEERING

**COURSE**:ADVANCED PROGRAMMING

**COURSE CODE**: SWEG3108

**SECTION:** B

GROUP 1

**Group members ID.NO**

1. Bethel Solomon ETS0318/14
2. Edom Mulugeta ETS0503/14
3. Ekhlas Abdulmelik ETS0511/14
4. Emnet Teshome ETS0529/14
5. Enkutatash Eshetu ETS0533/14
6. Ermias Ayele ETS0541/14

**SUBMITTED TO:** inst. Getnet

**SUBMISSION DATE** : Jun 07, 2024

# **Program set 1**

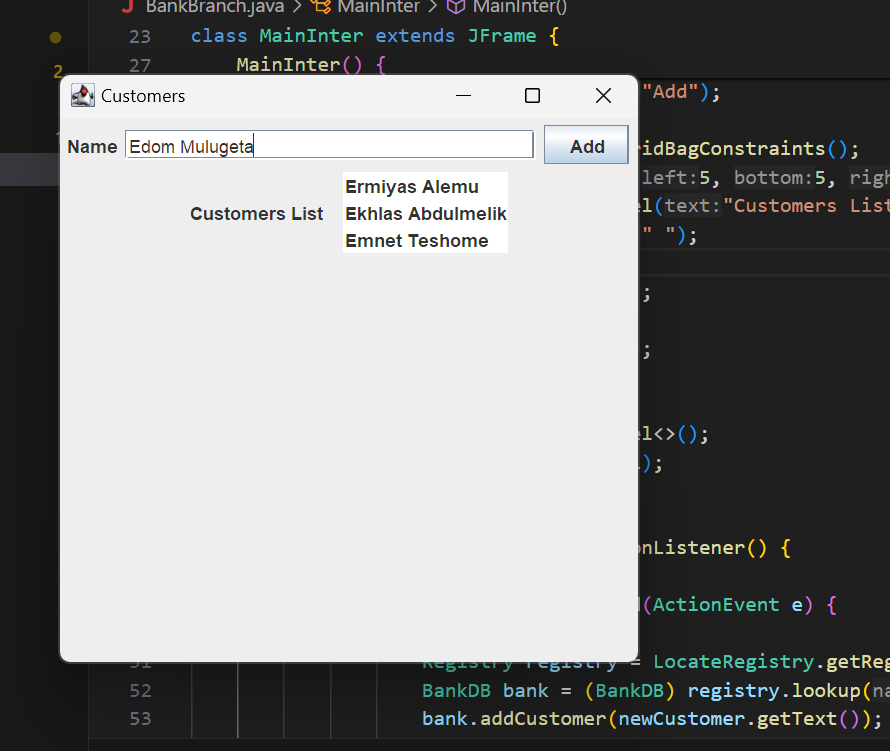
## Topic 1: RMI

### Program 1: Customer Registration in different branch

Branch register customers and the list of customer is stored in the server

|  |
| --- |
| // Interface import java.rmi.Remote; import java.rmi.RemoteException; import java.rmi.server.UnicastRemoteObject; import java.util.ArrayList; import java.util.List;  public interface BankDB extends Remote {  void addCustomer(String name) throws RemoteException;  List<String> seeCustomer() throws RemoteException; }  // Implementation class Bank extends UnicastRemoteObject implements BankDB {  private List<String> customerlist;  protected Bank() throws RemoteException {  super();  customerlist = new ArrayList<>();  }   @Override  public synchronized void addCustomer(String name) throws RemoteException {  customerlist.add(name);  }    @Override  public synchronized List<String> seeCustomer() throws RemoteException {  return new ArrayList<>(customerlist);   } }  // server import java.rmi.registry.LocateRegistry; import java.rmi.registry.Registry; public class BankServer {  public static void main(String[] args) throws Exception{  Bank bank = new Bank();  Registry registry = LocateRegistry.createRegistry(5000);  registry.bind("bank",bank);  System.out.println("Server Start");  }   }  // Add customer with GUI import java.awt.GridBagConstraints; import java.awt.Insets; import java.awt.event.ActionEvent; import java.awt.event.ActionListener; import java.rmi.registry.LocateRegistry; import java.rmi.registry.Registry; import java.util.List; import javax.swing.DefaultListModel; import javax.swing.JButton; import javax.swing.JFrame; import javax.swing.JLabel; import javax.swing.JList; import javax.swing.JPanel; import javax.swing.JTextField;  public class BankBranch {  public static void main(String[] args) throws Exception {  new MainInter();  } } class MainInter extends JFrame {  JList<String> listall;  DefaultListModel<String> listModel;  MainInter() {  JPanel mainPanel = new JPanel();  JLabel Name = new JLabel("Name");  JTextField newCustomer = new JTextField(30);  JButton add = new JButton("Add");  GridBagConstraints gbc = new GridBagConstraints();  gbc.insets = new Insets(5, 5, 5, 5);  mainPanel.add(Name, gbc);  mainPanel.add(newCustomer, gbc);  mainPanel.add(add, gbc);  listModel = new DefaultListModel<>();  listall = new JList<>(listModel);  mainPanel.add(listall, gbc);  add.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  try {  Registry registry = LocateRegistry.getRegistry("localhost", 5000);  BankDB bank = (BankDB) registry.lookup("bank");  bank.addCustomer(newCustomer.getText());  List<String> customers = bank.seeCustomer();  listModel.clear();  for (String customer : customers) {  listModel.addElement(customer);  }  newCustomer.setText("");  } catch (Exception error) {  error.printStackTrace();  }  }  });  add(mainPanel);  setTitle("Customers");  setSize(400, 400);  setVisible(true);  } } |

**Output:**



### Program 2: Online shopping cart

This code demonstrates how to implement a simple online shopping cart system using RMI. The server-side application provides remote methods for managing the shopping cart, while the client-side application interacts with these methods to perform operations such as adding, removing, and retrieving items from the shopping cart.

|  |
| --- |
| import java.rmi.Remote; import java.rmi.RemoteException;  public interface ShoppingCartService extends Remote {  void addItem(String itemName, double price) throws RemoteException;   void removeItem(String itemName) throws RemoteException;   double getTotal() throws RemoteException; }  // Implementation  import java.rmi.Naming; import java.util.\*; import java.rmi.RemoteException; import java.rmi.registry.LocateRegistry; import java.rmi.server.UnicastRemoteObject;  public class ShoppingCartServiceImpl extends UnicastRemoteObject implements ShoppingCartService {  private Map<String, Double> items;   protected ShoppingCartServiceImpl() throws RemoteException {  super();  items = new HashMap<>();  }   @Override  public void addItem(String itemName, double price) throws RemoteException {  items.put(itemName, price);  System.out.println(itemName + " added to cart.");  }   @Override  public void removeItem(String itemName) throws RemoteException {  if (items.containsKey(itemName)) {  items.remove(itemName);  System.out.println(itemName + " removed from cart.");  } else {  System.out.println("Item not found in cart.");  }  }   @Override  public double getTotal() throws RemoteException {  double total = 0.0;  for (Double price : items.values()) {  total += price;  }  return total;  }  public static void main(String[] args) {  try {  ShoppingCartService service = new ShoppingCartServiceImpl();  LocateRegistry.createRegistry(1099);  Naming.rebind("//localhost/ShoppingCartService", service);  System.out.println("Shopping Cart Service is running...");  } catch (Exception e) {  e.printStackTrace();  }  } }  // client import java.rmi.Naming;  public class ShoppingCartClient {  public static void main(String[] args) {  try {  ShoppingCartService service = (ShoppingCartService) Naming.lookup("//localhost/ShoppingCartService");   service.addItem("Laptop", 999.99);  service.addItem("Smartphone", 499.99);  service.addItem("Headphones", 149.99);   System.out.println("Total: $" + service.getTotal());   service.removeItem("Smartphone");  System.out.println("Total after removing Smartphone: $" + service.getTotal());  } catch (Exception e) {  e.printStackTrace();  }  } } |

**Output**

Shopping Cart Service is running...

Total: $1649.97

Total after removing Smartphone: $1149.98

### Program 3: Student Management System

Develop a very simple student management system using java where the server maintains student records

**Interface:**

|  |
| --- |
| import java.rmi.Remote; import java.rmi.RemoteException;  public interface Student extends Remote {  void addStudent(String name, int age) throws RemoteException;   StudentRecord getStudent(int id) throws RemoteException;   void updateStudent(int id, String name, int age) throws RemoteException;   void deleteStudent(int id) throws RemoteException; } |

**System Server**

|  |
| --- |
| // StudentServer.java import java.rmi.RemoteException; import java.rmi.server.UnicastRemoteObject; import java.util.HashMap; import java.util.Map;  public class StudentServer extends UnicastRemoteObject implements Student {  private Map<Integer, StudentRecord> studentMap;  private int nextId; |

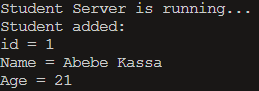
**System Client:**

|  |
| --- |
| // StudentClient.java import java.rmi.registry.LocateRegistry; import java.rmi.registry.Registry; import java.rmi.RemoteException;  public class StudentClient {  public static void main(String[] args) {  try {  Registry registry = LocateRegistry.getRegistry("localhost");  Student studentService = (Student) registry.lookup("rmi://localhost/Student");   // Test the student management system  studentService.addStudent("Alice", 20);  studentService.addStudent("Bob", 22);  studentService.addStudent("Charlie", 21);   System.out.println("Current Students:");  for (int i = 1; i <= 3; i++) {  System.out.println(studentService.getStudent(i));  }   studentService.updateStudent(2, "Bob Updated", 23);  studentService.deleteStudent(1);   System.out.println("Updated Students:");  for (int i = 1; i <= 3; i++) {  System.out.println(studentService.getStudent(i));  }  } catch (Exception e) {  e.printStackTrace();  }  } } |

**Student record:**

|  |
| --- |
| // StudentRecord.java import java.io.Serializable;  public class StudentRecord implements Serializable {  private int id;  private String name;  private int age;   public StudentRecord(int id, String name, int age) {  this.id = id;  this.name = name;  this.age = age;  }   public int getId() {  return id;  }   public String getName() {  return name;  }   public void setName(String name) {  this.name = name;  }   public int getAge() {  return age;  }   public void setAge(int age) {  this.age = age;  }   @Override  public String toString() {  return "StudentRecord{" +  "id = " + id +  ", name = '" + name + '\'' +  ", age = " + age +  '}';  } } |

**Output:**



### Program 4: File Uploader and Downloader

This is a simple program sets up an RMI server that allow clients to upload and download data then displays the output.

|  |
| --- |
| //interface  package RMI1;  import java.io.\*; import java.rmi.Remote; import java.rmi.RemoteException;  public interface FileServer extends Remote {  void uploadFile(String fileName, byte[] data) throws RemoteException;  byte[] downloadFile(String fileName) throws RemoteException; }  // implementation  package RMI1;  import java.io.\*; import java.rmi.RemoteException; import java.rmi.server.UnicastRemoteObject;  public class FileImpl extends UnicastRemoteObject implements FileServer {  private static final String FILE\_DIRECTORY = "files/";   protected FileImpl() throws RemoteException {  super();  File directory = new File(FILE\_DIRECTORY);  if (!directory.exists()) {  directory.mkdir();  }  }   @Override  public void uploadFile(String fileName, byte[] data) throws RemoteException {  try {  FileOutputStream fileOutputStream = new FileOutputStream(FILE\_DIRECTORY + fileName);  fileOutputStream.write(data);  fileOutputStream.close();  System.out.println("File " + fileName + " uploaded successfully.");  } catch (IOException e) {  throw new RemoteException("Error uploading file: " + e.getMessage());  }  }   @Override  public byte[] downloadFile(String fileName) throws RemoteException {  try {  File file = new File(FILE\_DIRECTORY + fileName);  byte[] buffer = new byte[(int) file.length()];  FileInputStream fileInputStream = new FileInputStream(file);  fileInputStream.read(buffer);  fileInputStream.close();  return buffer;  } catch (IOException e) {  throw new RemoteException("Error downloading file: " + e.getMessage());  }  } } // Server  package RMI1;  import java.rmi.registry.LocateRegistry; import java.rmi.registry.Registry;  public class FileServerMain {  public static void main(String[] args) {  try {  FileServer fileServer = new FileImpl();  Registry registry = LocateRegistry.createRegistry(1099);  registry.bind("FileServer", fileServer);  System.out.println("File server started.");  } catch (Exception e) {  System.err.println("Server exception: " + e.toString());  e.printStackTrace();  }  } }  // client package RMI1;   import java.rmi.registry.LocateRegistry; import java.rmi.registry.Registry;  public class FileClientMain {  public static void main(String[] args) {  try {  Registry registry = LocateRegistry.getRegistry("localhost");  FileServer fileServer = (FileServer) registry.lookup("FileServer");   // Upload file  String fileName = "test.txt";  byte[] data = "Hello, world!".getBytes();  fileServer.uploadFile(fileName, data);  System.out.println("File uploaded.");   // Download file  byte[] downloadedData = fileServer.downloadFile(fileName);  String receivedData = new String(downloadedData);  System.out.println("Downloaded data: " + receivedData);  } catch (Exception e) {  System.err.println("Client exception: " + e.toString());  e.printStackTrace();  }  } } |

**Output:**

|  |
| --- |
| File server started. File uploaded. Downloaded data: Hello, world! |

### Program 5: Random messages service

This is a Java-based RMI (Remote Method Invocation) application. The application consists of a client that can send and receive messages to/from a remote server. The server stores these messages in a file and can return a random message upon request.

### **1. Client**

The Client class connects to the RMI registry and interacts with the remote server to send or retrieve messages.

|  |
| --- |
| package RMI;  import java.rmi.registry.LocateRegistry; import java.rmi.registry.Registry; import java.util.Scanner;  public class Client {  public static void main(String[] args) {  try {  Registry reg = LocateRegistry.getRegistry("localhost", 1099);  RemoteLogic messenger = (RemoteLogic) reg.lookup("remoteMessageService");   Scanner scanner = new Scanner(System.in);  while (true) {  System.out.println("1. Send a message");  System.out.println("2. Get a random message");  System.out.println("3. Exit");  System.out.print("Choose an option: ");  int option = scanner.nextInt();  scanner.nextLine();    if (option == 1) {  System.out.print("Enter your message: ");  String message = scanner.nextLine();  messenger.sendMessage(message);  System.out.println("Message sent.");  } else if (option == 2) {  String randomMessage = messenger.getMessage();  System.out.println("Random message: " + randomMessage);  } else if (option == 3) {  break;  } else {  System.out.println("Invalid option. Please try again.");  }  }  scanner.close();  } catch (Exception e) {  e.printStackTrace();  }  } } |

### **2. Remote Interface**

The Remote Logic interface defines the remote methods that can be invoked by the client.

|  |
| --- |
| package RMI;  import java.rmi.Remote; import java.rmi.RemoteException;  public interface RemoteLogic extends Remote {  void sendMessage(String message) throws RemoteException;  String getMessage() throws RemoteException; } |

### **3. Server Implementation**

The RemoteLogicImp class implements the RemoteLogic interface and handles the logic for storing and retrieving messages.

|  |
| --- |
| package RMI;  import java.io.\*; import java.rmi.RemoteException; import java.rmi.server.UnicastRemoteObject; import java.util.\*;  public class RemoteLogicImp extends UnicastRemoteObject implements RemoteLogic {  private File file;  private ArrayList<String> messages;   public RemoteLogicImp() throws RemoteException {  super();  this.file = new File("RMI/repo.txt");  this.messages = loadMessagesFromFile();  }   private ArrayList<String> loadMessagesFromFile() {  ArrayList<String> messages = new ArrayList<>();  try (BufferedReader reader = new BufferedReader(new FileReader(file))) {  String line;  while ((line = reader.readLine()) != null) {  messages.add(line);  }  } catch (IOException e) {  e.printStackTrace();  }  return messages;  }   private void saveMessagesToFile() {  try (BufferedWriter writer = new BufferedWriter(new FileWriter(file))) {  for (String message : messages) {  writer.write(message);  writer.newLine();  }  } catch (IOException e) {  e.printStackTrace();  }  }   @Override  public synchronized void sendMessage(String message) {  messages.add(message);  saveMessagesToFile();  System.out.println("Your message is recorded!");  }   @Override  public synchronized String getMessage() {  if (messages.isEmpty()) {  return "No messages available";  }  int randomized = new Random().nextInt(messages.size());  return messages.get(randomized);  } } |

### **4. Server**

The RemoteServer class creates and binds the remote object to the RMI registry.

|  |
| --- |
| package RMI;  import java.rmi.registry.LocateRegistry; import java.rmi.registry.Registry;  public class RemoteServer {  public static void main(String[] args) {  try {  RemoteLogicImp rli = new RemoteLogicImp();   Registry regs = LocateRegistry.createRegistry(1099);  regs.rebind("remoteMessageService", rli);   System.out.println("Server ready for use");  } catch (Exception e) {  e.printStackTrace();  System.out.println("Server problem");  }  } } |

## **Output**

### **Client Output**

|  |
| --- |
| 1. Send a message 2. Get a random message 3. Exit Choose an option: 1 Enter your message: Hello, RMI! Message sent.  1. Send a message 2. Get a random message 3. Exit Choose an option: 2 Random message: Hello, RMI!  1. Send a message 2. Get a random message 3. Exit Choose an option: 3 |

**Server Output**

|  |
| --- |
| Server ready for use Your message is recorded! |

### Program 6: calculator

The calculator project demonstrates a simple Remote Method Invocation (RMI) application in Java. It consists of a client-server architecture where the server provides arithmetic operations as services, and the client consumes these services remotely.

**Calculator Interface**

import java.rmi.Remote;

import java.rmi.RemoteException;

public interface Calculator extends Remote {

int add(int a, int b) throws RemoteException;

int subtract(int a, int b) throws RemoteException;

int multiply(int a, int b) throws RemoteException;

int divide(int a, int b) throws RemoteException;

}

**Calculator Implementation**

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

public class CalculatorImpl extends UnicastRemoteObject implements Calculator {

protected CalculatorImpl() throws RemoteException {

super();

}

public int add(int a, int b) throws RemoteException {

return a + b;

}

public int subtract(int a, int b) throws RemoteException {

return a - b;

}

public int multiply(int a, int b) throws RemoteException {

return a \* b;

}

public int divide(int a, int b) throws RemoteException {

if (b == 0) throw new ArithmeticException("Division by zero");

return a / b;

}

}

**Calculator Server**

import java.rmi.Naming;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

public class CalculatorServer {

public static void main(String[] args) {

try {

CalculatorImpl calc = new CalculatorImpl();

Registry registry = LocateRegistry.createRegistry(5000);

registry.bind("cal", calc);

System.out.println("Calculator Service is running...");

} catch (Exception e) {

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

e.printStackTrace();

}

}

}

Output:



**Calculator Client**

import java.rmi.Naming;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

public class CalculatorClient {

public static void main(String[] args) {

try {

Registry registry = LocateRegistry.getRegistry(5000);

Calculator calc = (Calculator) registry.lookup("cal");

System.out.println("Addition: " + calc.add(5, 3));

System.out.println("Subtraction: " + calc.subtract(5, 3));

System.out.println("Multiplication: " + calc.multiply(5, 3));

System.out.println("Division: " + calc.divide(5, 3));

} catch (Exception e) {

System.out.println("Client error: " + e.getMessage());

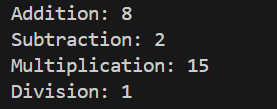
e.printStackTrace();

}

}

}

Output:



# Topic 2: UDP

### Program 1: Real-Time Temperature Retrieval from Server upon Client Request

The client request to get the current information from the server and the server send the temperature at that moment

|  |
| --- |
| package Temp; import java.net.DatagramPacket; import java.net.DatagramSocket; import java.net.InetAddress; import java.util.Random;  public class UDPServer {  public static void main(String[] args) {  try {  DatagramSocket socket = new DatagramSocket(9876);  byte[] receiveBuffer = new byte[1024];  Random random = new Random();  while (true) {  DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);  socket.receive(receivePacket);    String receivedData = new String(receivePacket.getData(), 0, receivePacket.getLength());  System.out.println("Received request: " + receivedData);    if (receivedData.equals("GET\_TEMPERATURE")) {    int temperature = 15 + random.nextInt(15);   String responseData = "Current temperature: " + temperature + "°C";  byte[] sendBuffer = responseData.getBytes();    InetAddress clientAddress = receivePacket.getAddress();  int clientPort = receivePacket.getPort();  DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length, clientAddress, clientPort);    socket.send(sendPacket);  }  }  } catch (Exception e) {  e.printStackTrace();  }  } } package Temp;  import java.net.DatagramPacket; import java.net.DatagramSocket; import java.net.InetAddress;  public class UDPClient {  public static void main(String[] args) {  try {  DatagramSocket socket = new DatagramSocket();  String message = "GET\_TEMPERATURE";  byte[] sendBuffer = message.getBytes();    InetAddress serverAddress = InetAddress.getByName("localhost");  DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length, serverAddress, 9876);  socket.send(sendPacket);    byte[] receiveBuffer = new byte[1024];  DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);  socket.receive(receivePacket);    String receivedData = new String(receivePacket.getData(), 0, receivePacket.getLength());  System.out.println("Received response: " + receivedData);    socket.close();  } catch (Exception e) {  e.printStackTrace();  }  } } |

**Output**

|  |
| --- |
| Server : Server start Client : Received response: Current temperature: 27°C |

### Program 2: Word count

This Java code implements a UDP-based word count service, where the server listens on port 9876 for text strings from clients, counts the words, and sends the word count back. The client sends a text string to the server and receives the word count as a response. The server runs indefinitely, handling multiple client requests sequentially.

|  |
| --- |
| //server  import java.net.DatagramPacket; import java.net.DatagramSocket; import java.net.InetAddress;  public class UDPWordCountServer {  public static void main(String[] args) {  DatagramSocket socket = null;  try {  socket = new DatagramSocket(9876);   byte[] receiveBuffer = new byte[1024];   System.out.println("UDP Word Count Server is running...");   while (true) {  DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);  socket.receive(receivePacket);   String receivedText = new String(receivePacket.getData(), 0, receivePacket.getLength());  int wordCount = countWords(receivedText);   String response = "Word count: " + wordCount;  byte[] sendBuffer = response.getBytes();  InetAddress clientAddress = receivePacket.getAddress();  int clientPort = receivePacket.getPort();  DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length, clientAddress,  clientPort);  socket.send(sendPacket);   }  } catch (Exception e) {  e.printStackTrace();  } finally {  if (socket != null && !socket.isClosed()) {  socket.close();  }  }  }  private static int countWords(String text) {  if (text == null || text.isEmpty()) {  return 0;  }  String[] words = text.trim().split("\\s+");  return words.length;  } }  // client  import java.net.DatagramPacket; import java.net.DatagramSocket; import java.net.InetAddress; import java.util.Scanner;  public class UDPWordCountClient {  public static void main(String[] args) {  DatagramSocket socket = null;  try {  socket = new DatagramSocket();  InetAddress serverAddress = InetAddress.getByName("localhost");    Scanner scanner = new Scanner(System.in);  System.out.print("Enter a string: ");  String text = scanner.nextLine();   byte[] sendBuffer = text.getBytes();  DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length, serverAddress, 9876);  socket.send(sendPacket); // Send text to server   byte[] receiveBuffer = new byte[1024];  DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);  socket.receive(receivePacket); // Receive word count from server   String response = new String(receivePacket.getData(), 0, receivePacket.getLength());  System.out.println("Response from server: " + response);  } catch (Exception e) {  e.printStackTrace();  } finally {  if (socket != null && !socket.isClosed()) {  socket.close();  }  }  } } |

**Output**

UDP Word Count Server is running...

Enter a string: final project schedule program

Response from server: Word count: 4

### Program 3: UDP File Transfer Application

Develop a file transfer application using that allows users to send and receive files between clients over a UDP connection.

**Sender:**

|  |
| --- |
| import java.io.File; import java.io.FileInputStream; import java.io.IOException; import java.net.DatagramPacket; import java.net.DatagramSocket; import java.net.InetAddress;  public class UdpFileSender {  public static void main(String[] args) {  String filePath = "path/to/file.txt"; // Specify the path to the file to be sent  String host = "localhost";  int port = 12345;   try (DatagramSocket socket = new DatagramSocket()) {  File file = new File(filePath);  FileInputStream fileInputStream = new FileInputStream(file);  byte[] buffer = new byte[1024];  DatagramPacket packet;  InetAddress address = InetAddress.getByName(host);   while (fileInputStream.read(buffer) != -1) {  packet = new DatagramPacket(buffer, buffer.length, address, port);  socket.send(packet);  }   System.out.println("File sent successfully.");  fileInputStream.close();  } catch (IOException e) {  e.printStackTrace();  }  } } |

**Receiver:**

|  |
| --- |
| import java.io.FileOutputStream; import java.io.IOException; import java.net.DatagramPacket; import java.net.DatagramSocket;  public class UdpFileReceiver {  public static void main(String[] args) {  int port = 12345;   try (DatagramSocket socket = new DatagramSocket(port)) {  byte[] buffer = new byte[1024];  DatagramPacket packet = new DatagramPacket(buffer, buffer.length);   FileOutputStream fileOutputStream = new FileOutputStream("received\_file.txt");   while (true) {  socket.receive(packet);  fileOutputStream.write(packet.getData(), 0, packet.getLength());  }   } catch (IOException e) {  e.printStackTrace();  }  } } |

**Output:**



### Program 4: Basic Reservation system for a cinema

This program allows customers to check the available movies and seat and then reserve seats.

|  |
| --- |
| // server  package UDP;  import java.net.DatagramPacket; import java.net.DatagramSocket; import java.util.HashMap; import java.util.Map;  public class ReservationServer { private static final int PORT = 9876; private static final int TOTAL\_SEATS = 20; private static final Map<String, Map<Integer, Boolean>> movies = new HashMap<>();  public static void main(String[] args) { initializeMovies();  DatagramSocket socket = null; try { socket = new DatagramSocket(PORT); byte[] receiveData = new byte[1024]; byte[] sendData = new byte[1024];  while (true) { DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length); System.out.println("Server is waiting for a packet..."); socket.receive(receivePacket);  String message = new String(receivePacket.getData(), 0, receivePacket.getLength()); System.out.println("Received: " + message);  String response = handleRequest(message); |

|  |
| --- |
| sendData = response.getBytes();  DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, receivePacket.getAddress(), receivePacket.getPort()); socket.send(sendPacket); } } catch (Exception e) { e.printStackTrace(); } finally { if (socket != null && !socket.isClosed()) { socket.close(); } } } private static void initializeMovies() { // Initializing movies with available seats movies.put("home", new HashMap<>()); movies.put("closed", new HashMap<>()); movies.put("Movie1", new HashMap<>()); movies.put("Movie2", new HashMap<>()); movies.put("begin", new HashMap<>());  for (int i = 1; i <= TOTAL\_SEATS; i++) { movies.get("home").put(i, true); movies.get("closed").put(i, true); movies.get("Movie1").put(i, true); movies.get("Movie2").put(i, true); movies.get("begin").put(i, true); } } private static String handleRequest(String message) { String[] parts = message.split(" "); if (parts.length < 1) { return "Invalid command."; }  String command = parts[0].toUpperCase(); switch (command) { case "CHECK": |

|  |
| --- |
| if (parts.length < 2) { return "Invalid command. Usage: CHECK <movie\_name>"; } return checkAvailability(parts[1].toLowerCase()); case "RESERVE": if (parts.length < 3) { |

|  |
| --- |
| return "Invalid command. Usage: RESERVE <movie\_name> <seat\_number>"; } int seatNumber; try { seatNumber = Integer.parseInt(parts[2]); } catch (NumberFormatException e) { return "Invalid seat number. It must be an integer."; } return reserveSeat(parts[1].toLowerCase(), seatNumber); default: return "Unknown command. Available commands: CHECK <movie\_name>, RESERVE <movie\_name> <seat\_number>"; } }  private static String checkAvailability(String movieName) { if (!movies.containsKey(movieName)) { return "Movie not found. Available movies: " + movies.keySet(); }  StringBuilder response = new StringBuilder("Seat Availability for ").append(movieName).append(":\n"); for (Map.Entry<Integer, Boolean> entry : movies.get(movieName).entrySet()) { response.append("Seat ").append(entry.getKey()).append(": ").append(entry.getValue() ? "Available" : "Reserved").append("\n"); } return response.toString(); }  private static String reserveSeat(String movieName, int seatNumber) { if (!movies.containsKey(movieName)) { return "Movie not found. Available movies: " + movies.keySet(); }  Map<Integer, Boolean> seats = movies.get(movieName); if (seatNumber < 1 || seatNumber > TOTAL\_SEATS) { return "Invalid seat number. Please choose a seat between 1 and " + TOTAL\_SEATS; } if (seats.get(seatNumber)) { seats.put(seatNumber, false); return "Seat " + seatNumber + " for " + movieName + " has been successfully reserved."; } else { return "Seat " + seatNumber + " for " + movieName + " is already reserved."; } } } |

|  |
| --- |
| // Client package UDP;  import java.net.DatagramPacket; import java.net.DatagramSocket; import java.util.HashMap; import java.util.Map;  public class ReservationServer { private static final int PORT = 9876; private static final int TOTAL\_SEATS = 20; private static final Map<String, Map<Integer, Boolean>> movies = new HashMap<>();  public static void main(String[] args) { initializeMovies(); DatagramSocket socket = null; try { socket = new DatagramSocket(PORT); byte[] receiveData = new byte[1024]; byte[] sendData = new byte[1024];  while (true) { DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length); System.out.println("Server is waiting for a packet..."); socket.receive(receivePacket);  String message = new String(receivePacket.getData(), 0, receivePacket.getLength()); System.out.println("Received: " + message);  String response = handleRequest(message); sendData = response.getBytes();  DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length, receivePacket.getAddress(), receivePacket.getPort()); socket.send(sendPacket); } } catch (Exception e) { e.printStackTrace(); } finally { if (socket != null && !socket.isClosed()) { socket.close(); } } } |

|  |
| --- |
| private static void initializeMovies() { // Initialize movies with available seats movies.put("home", new HashMap<>()); movies.put("closed", new HashMap<>()); movies.put("Movie1", new HashMap<>()); movies.put("Movie2", new HashMap<>()); movies.put("begin", new HashMap<>());  for (int i = 1; i <= TOTAL\_SEATS; i++) { movies.get("home").put(i, true); movies.get("closed").put(i, true); movies.get("Movie1").put(i, true); movies.get("Movie2").put(i, true); |

|  |
| --- |
| movies.get("begin").put(i, true); } }  private static String handleRequest(String message) { String[] parts = message.split(" "); if (parts.length < 1) { return "Invalid command."; }  String command = parts[0].toUpperCase(); switch (command) { case "CHECK": if (parts.length < 2) { return "Invalid command. Usage: CHECK <movie\_name>"; } return checkAvailability(parts[1].toLowerCase()); case "RESERVE": if (parts.length < 3) { return "Invalid command. Usage: RESERVE <movie\_name> <seat\_number>"; } int seatNumber; try { seatNumber = Integer.parseInt(parts[2]); } catch (NumberFormatException e) { return "Invalid seat number. It must be an integer."; } return reserveSeat(parts[1].toLowerCase(), seatNumber); default: return "Unknown command. Available commands: CHECK <movie\_name>, RESERVE <movie\_name> <seat\_number>"; } }  private static String checkAvailability(String movieName) { if (!movies.containsKey(movieName)) { return "Movie not found. Available movies: " + movies.keySet();} |

|  |
| --- |
| StringBuilder response = new StringBuilder("Seat Availability for ").append(movieName).append(":\n"); for (Map.Entry<Integer, Boolean> entry : movies.get(movieName).entrySet()) { response.append("Seat").append(entry.getKey()).append(":").append(entry.getValue() ? "Available" : "Reserved").append("\n"); } return response.toString(); }  private static String reserveSeat(String movieName, int seatNumber) { if (!movies.containsKey(movieName)) { return "Movie not found. Available movies: " + movies.keySet(); }  Map<Integer, Boolean> seats = movies.get(movieName); if (seatNumber < 1 || seatNumber > TOTAL\_SEATS) { return "Invalid seat number. Please choose a seat between 1 and " + TOTAL\_SEATS; }  **if (seats.get(seatNumber)) { seats.put(seatNumber, false); return "Seat " + seatNumber + " for " + movieName + " has been successfully reserved."; } else { return "Seat " + seatNumber + " for " + movieName + " is already reserved."; } } }** |

Output:

|  |
| --- |
| // server Server is waiting for a packet... Received: CHECK home  Enter command (CHECK <movie\_name> / RESERVE <movie\_name> <seat\_number> / EXIT): CHECK home Response from server: Seat Availability for home: Seat 1: Available Seat 2: Available Seat 3: Available Seat 4: Available Seat 5: Available Seat 6: Available Seat 7: Available Seat 8: Available Seat 9: Available Seat 10: Available Seat 11: Available Seat 12: Available Seat 13: Available Seat 14: Available Seat 15: Available Seat 16: Available Seat 17: Available Seat 18: Available Seat 19: Available Seat 20: Available  Enter command (CHECK <movie\_name> / RESERVE <movie\_name> <seat\_number> / EXIT): RESERVE home 16 Response from server: Seat 16 for home has been successfully reserved. Enter command (CHECK <movie\_name> / RESERVE <movie\_name> <seat\_number> / EXIT): RESERVE home16 Response from server: Invalid command. Usage: RESERVE <movie\_name> <seat\_number> Enter command (CHECK <movie\_name> / RESERVE <movie\_name> <seat\_number> / EXIT): RESERVE home 16 Response from server: Seat 16 for home is already reserved. Enter command (CHECK <movie\_name> / RESERVE <movie\_name> <seat\_number> / EXIT): |

### Program 5: File transfer

The main function of this project is to facilitate the transfer of a file from a client to a server using UDP (User Datagram Protocol). This simple file transfer system consists of a FileTransferServer that receives and saves the file, and a FileTransferClient that reads and sends the file.

**FileTransferServer.java**

import java.io.\*;

import java.net.\*;

public class FileTransferServer {

private static final int PORT = 5002;

private static final String OUTPUT\_FILENAME = "received\_message.txt";

public static void main(String[] args) {

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

System.out.println("Server is running...");

byte[] receiveBuffer = new byte[65535];

DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);

// Receive the file data

serverSocket.receive(receivePacket);

// Save the received file data to disk

try (FileOutputStream fos = new FileOutputStream(OUTPUT\_FILENAME)) {

fos.write(receivePacket.getData(), 0, receivePacket.getLength());

System.out.println("File received and saved as " + OUTPUT\_FILENAME);

}

} catch (IOException e) {

e.printStackTrace();

}}}

**Output:**



**FileTransferClient.java**

import java.io.\*;

import java.net.\*;

public class FileTransferClient {

public static void main(String[] args) {

String hostname = "localhost";

int port = 5002;

String filename = "message.txt";

try {

File file = new File(filename);

byte[] fileData = new byte[(int) file.length()];

// Read the file data

try (FileInputStream fis = new FileInputStream(file)) {

fis.read(fileData);

}

// Send the file data to the server

try (DatagramSocket clientSocket = new DatagramSocket()) {

DatagramPacket sendPacket = new DatagramPacket(fileData, fileData.length, InetAddress.getByName(hostname), port);

clientSocket.send(sendPacket);

System.out.println("File sent: " + filename);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

Output:



### Program 5: Echo server

The server listens on a specified port for incoming messages and echoes them back to the client. The client sends messages to the server and receives responses.

**Server:**

package UDP;

import java.io.\*;

import java.net.\*;

public class Server {

public static void main(String[] args) {

final int PORT = 12345;

try (DatagramSocket socket = new DatagramSocket(PORT)) {

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

while (true) {

byte[] receiveBuffer = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);

socket.receive(receivePacket);

String receivedMessage = new String(receivePacket.getData(), 0, receivePacket.getLength());

System.out.println("Client: " + receivedMessage);

InetAddress clientAddress = receivePacket.getAddress();

int clientPort = receivePacket.getPort();

byte[] sendBuffer = receivedMessage.getBytes();

DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length, clientAddress, clientPort);

socket.send(sendPacket);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Client:**

package UDP;

import java.io.\*;

import java.net.DatagramPacket;

import java.net.DatagramSocket;

import java.net.InetAddress;

public class Client {

public static void main(String[] args) {

final String SERVER\_ADDRESS = "localhost";

final int SERVER\_PORT = 12345;

try (DatagramSocket socket = new DatagramSocket()) {

BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));

while (true) {

System.out.print("Enter message to send (type 'exit' to quit): ");

String message = userInput.readLine();

byte[] sendBuffer = message.getBytes();

InetAddress serverAddress = InetAddress.getByName(SERVER\_ADDRESS);

DatagramPacket sendPacket = new DatagramPacket(sendBuffer, sendBuffer.length, serverAddress, SERVER\_PORT);

socket.send(sendPacket);

if (message.equalsIgnoreCase("exit")) {

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

break;

}

byte[] receiveBuffer = new byte[1024];

DatagramPacket receivePacket = new DatagramPacket(receiveBuffer, receiveBuffer.length);

socket.receive(receivePacket);

String receivedMessage = new String(receivePacket.getData(), 0, receivePacket.getLength());

System.out.println("Server: " + receivedMessage);

}

} catch (IOException e) {

e.printStackTrace();

}

}

}

**Output:**

Server started. Listening on port 12345

Enter message to send (type 'exit' to quit): Hello, Server!

Client: Hello, Server!

Server: Hello, Server!

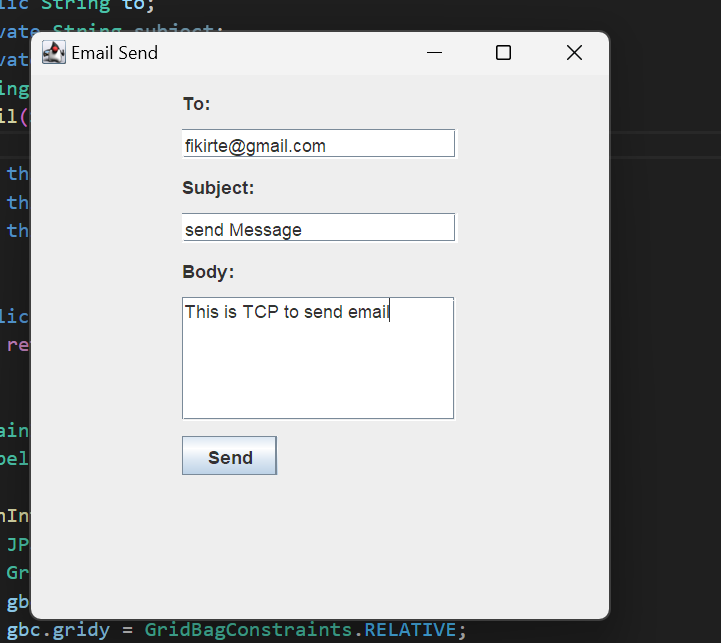
# Topic 3: TCP

### Program 1: send email project

The client send the receiver address and the body of the email then the server send the data to the receiver using TCP and thread

|  |
| --- |
| import java.io.ObjectInputStream; import java.io.ObjectOutputStream; import java.io.Serializable; import java.net.ServerSocket; import java.net.Socket; import javax.swing.JButton; import javax.swing.JFrame; import javax.swing.JLabel; import javax.swing.JPanel; import javax.swing.JScrollPane; import javax.swing.JTextArea; import javax.swing.JTextField; import java.awt.FlowLayout; import java.awt.GridBagConstraints; import java.awt.GridBagLayout; import java.awt.Insets; import java.awt.event.ActionEvent; import java.awt.event.ActionListener; public class TCP {  public static void main(String[] args) throws Exception{  ServerThread serverThread = new ServerThread();  serverThread.start();   Thread.sleep(1000);  new MainInterface();  }  } class ServerThread extends Thread{  public void run(){  try{  ServerSocket serverSocket = new ServerSocket(5000);  Socket socket = serverSocket.accept();   ObjectOutputStream output = new ObjectOutputStream(socket.getOutputStream());  ObjectInputStream input = new ObjectInputStream(socket.getInputStream());   Gmail gmail = (Gmail) input.readObject();  String success = gmail.sendEmail();   output.writeObject(success);    serverSocket.close();  }catch(Exception e){  e.printStackTrace();  }  } }  class Gmail implements Serializable{    private String to;  private String subject;  private String body;  String host = "127.0.0.1";  Gmail(String to,String subject,String body){    this.to = to;  this.subject = subject;  this.body = body;  }   public String sendEmail(){  return "Message Sucessfully send";  } } class MainInterface extends JFrame {  JLabel resultLabel;   MainInterface() {  JPanel mainpanel = new JPanel(new GridBagLayout());  GridBagConstraints gbc = new GridBagConstraints();  gbc.gridx = 0;  gbc.gridy = GridBagConstraints.RELATIVE;  gbc.anchor = GridBagConstraints.WEST;  gbc.insets = new Insets(5, 5, 5, 5);   JLabel toLabel = new JLabel("To:");  JLabel subjectLabel = new JLabel("Subject:");  JLabel bodyLabel = new JLabel("Body:");   JTextField toField = new JTextField(20);  JTextField subjectField = new JTextField(20);  JTextArea bodyField = new JTextArea(5, 20);  JScrollPane bodyScrollPane = new JScrollPane(bodyField);   JButton send = new JButton("Send");  resultLabel = new JLabel("");    mainpanel.add(toLabel, gbc);  mainpanel.add(toField, gbc);  mainpanel.add(subjectLabel, gbc);  mainpanel.add(subjectField, gbc);  mainpanel.add(bodyLabel, gbc);  mainpanel.add(bodyScrollPane, gbc);    mainpanel.add(send, gbc);  mainpanel.add(resultLabel, gbc);     // Button event  send.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  String to = toField.getText();  String subject = subjectField.getText();  String body = bodyField.getText();  toField.setText("");  subjectField.setText("");  bodyField.setText("");   Gmail gmail = new Gmail(to, subject, body);   try {  Socket client = new Socket("localhost", 5000);  ObjectOutputStream output =   new ObjectOutputStream(client.getOutputStream());  ObjectInputStream input =   new ObjectInputStream(client.getInputStream());  output.writeObject(gmail);  String result = (String) input.readObject();  resultLabel.setText(result);   client.close();  } catch (Exception error) {  error.printStackTrace();  }  }  });   JPanel panel = new JPanel(new FlowLayout(FlowLayout.CENTER));  panel.add(mainpanel);  add(panel);  setTitle("Email Send");  setSize(400, 400);  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  setLocationRelativeTo(null); // Centers the frame  setVisible(true);  } } |

## Output:



## 

### Program 2: e-commerce checkout process

This implementation demonstrates a simple TCP-based e-commerce checkout process where clients can place orders and the server processes these orders, simulating payment handling.

|  |
| --- |
| // server import java.io.\*; import java.net.\*;  public class ECommerceServer {  public static void main(String[] args) {  try {  ServerSocket serverSocket = new ServerSocket(12345);  System.out.println("Server started. Waiting for clients...");   while (true) {  Socket clientSocket = serverSocket.accept();  System.out.println("Client connected: " + clientSocket);   BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));  PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);   // Read order details from client  String orderDetails = in.readLine();  System.out.println("Received order details from client: " + orderDetails);   // Simulate processing order and payment  boolean paymentSuccess = processPayment(orderDetails);   // Send response to client  if (paymentSuccess) {  out.println("Order placed successfully. Thank you for shopping with us!");  } else {  out.println("Payment failed. Please try again.");  }  clientSocket.close();  }  } catch (IOException e) {  e.printStackTrace();  }  }   private static boolean processPayment(String orderDetails) {    return true;   } }  //client package TCP; import java.io.BufferedReader; import java.io.IOException; import java.io.InputStreamReader; import java.io.PrintWriter; import java.net.Socket;  public class ECommerceClient {  public static void main(String[] args) {  try {  Socket socket = new Socket("localhost", 12345);   BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));  BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  PrintWriter out = new PrintWriter(socket.getOutputStream(), true);     System.out.println("Enter order details:");  String orderDetails = userInput.readLine();   out.println(orderDetails);   String response = in.readLine();  System.out.println("Server response: " + response);   socket.close();  } catch (IOException e) {  e.printStackTrace();  }  } } |

**Output**

|  |
| --- |
| **Server started. Waiting for clients... Enter order details: jeans Server response: Order placed successfully. Thank you for shopping with us!** |

### Program 3: TCP Chat Application

Develop a multi-client chat application using java where users can send messages to each other over TCP connections.

Client Side

|  |
| --- |
| import java.io.IOException; import java.io.PrintWriter; import java.net.Socket; import java.util.Scanner;  public class ChatClient {  public static void main(String[] args) {  try {  Socket socket = new Socket("localhost", 5000); // Connect to the server on localhost and port 5000   PrintWriter writer = new PrintWriter(socket.getOutputStream());  Thread readerThread = new Thread(new ClientReader(socket));  readerThread.start();   Scanner scanner = new Scanner(System.in);  while (true) {  String message = scanner.nextLine();  writer.println(message);  writer.flush();  }  } catch (IOException e) {  e.printStackTrace();  }  }   static class ClientReader implements Runnable {   private Socket socket;  private Scanner input;   public ClientReader(Socket socket) {  this.socket = socket;  }   @Override  public void run() {  try {  input = new Scanner(socket.getInputStream());  while (input.hasNextLine()) {  String message = input.nextLine();  System.out.println("Received from server: " + message);  }  } catch (IOException e) {  e.printStackTrace();  }  }  } } |

**Server Side:**

|  |
| --- |
| import java.io.IOException; import java.io.PrintWriter; import java.net.ServerSocket; import java.net.Socket; import java.util.Scanner; import java.util.concurrent.CopyOnWriteArrayList;  public class ChatServer {  private static CopyOnWriteArrayList<PrintWriter> clientOutputStreams = new CopyOnWriteArrayList<>();   public static void main(String[] args) {  try {  ServerSocket serverSocket = new ServerSocket(5000); // Open server socket on port 5000  System.out.println("Server started. Waiting for clients...");   while (true) {  Socket clientSocket = serverSocket.accept(); // Accept incoming connection from a client  System.out.println("Client connected: " + clientSocket);   PrintWriter writer = new PrintWriter(clientSocket.getOutputStream());  clientOutputStreams.add(writer);   Thread t = new Thread(new ClientHandler(clientSocket));  t.start();  }  } catch (IOException e) {  e.printStackTrace();  }  }   static class ClientHandler implements Runnable {  private Socket clientSocket;  private Scanner input;   public ClientHandler(Socket socket) {  this.clientSocket = socket;  }   @Override  public void run() {  try {  input = new Scanner(clientSocket.getInputStream());  while (input.hasNextLine()) {  String message = input.nextLine();  System.out.println("Received from client: " + message);  broadcastMessage(message);  }  } catch (IOException e) {  e.printStackTrace();  }  }  }   private static void broadcastMessage(String message) {  for (PrintWriter writer : clientOutputStreams) {  writer.println(message);  writer.flush();  }  } } |

**Output:**



### Program 4: Contact Manager

This program uses a TCP server to enable the user to add, list and search contact information like name and phone number.

|  |
| --- |
| //server package TCP; import java.io.\*; import java.net.ServerSocket; import java.net.Socket; import java.util.ArrayList; import java.util.List;  class Contact {  private String name;  private int phone;   public Contact(String name, int phone) {  this.name = name;  this.phone = phone;  }   public String getName() {  return name;  }  public int getPhone() {  return phone;  }  @Override  public String toString() {  return name + ": " + phone;  } } public class ContactServer {  private static final List<Contact> contacts = new ArrayList<>();  private static final int MAX\_PHONE\_LENGTH = 15;  private static final int MAX\_NAME\_LENGTH = 30;   public static synchronized void addContact(String name, int phone) {  if (!isValidName(name)) {  return;  }  for (Contact contact : contacts) {  if (contact.getName().equalsIgnoreCase(name) && contact.getPhone() == phone) {  System.err.println("Warning: A contact with the same name and phone number already exists.");  return;  }  }  contacts.add(new Contact(name, phone));  }  public static synchronized String listContacts() {  StringBuilder list = new StringBuilder();  for (Contact contact : contacts) {  list.append(contact).append("\n");  }   list.append("\n");  return list.toString();  }  public static synchronized String searchContact(String name) {  StringBuilder result = new StringBuilder();  boolean found = false;  for (Contact contact : contacts) {  if (contact.getName().equalsIgnoreCase(name)) {  result.append("Found contact: ").append(contact).append("\n");  found = true;  }  }  if (!found) {  result.append("No contact found with the name: ").append(name).append("\n");  }  return result.toString();  }  private static boolean isValidName(String name) {  if (name.isEmpty()) {  System.err.println("Error: Name is not entered.");  return false;  }  if (name.length() > MAX\_NAME\_LENGTH) {  System.err.println("Error: Name exceeds maximum length!");  return false;  }  return true;  }  private static boolean isValidPhone(int phone) {  if (phone < 0 || phone > 999999999) {  System.err.println("Error: Invalid phone number.");  return false;  }  return true;  }  public static void main(String[] args) {  try (ServerSocket serverSocket = new ServerSocket(6789)) {  System.out.println("Server has started.....");   while (true) {  Socket clientSocket = serverSocket.accept();  System.out.println("Client connected: " + clientSocket.getInetAddress());   new Thread(new ClientHandler(clientSocket)).start();  }  } catch (IOException e) {  e.printStackTrace();  }  }   static class ClientHandler implements Runnable {  private final Socket clientSocket;  private BufferedReader inFromClient;  private DataOutputStream outToClient;   public ClientHandler(Socket clientSocket) {  this.clientSocket = clientSocket;  try {  inFromClient = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));  outToClient = new DataOutputStream(clientSocket.getOutputStream());  } catch (IOException e) {  e.printStackTrace();  }  }   @Override  public void run() {  try {  while (true) {  String clientMessage = inFromClient.readLine();  if (clientMessage == null) break;  processMessage(clientMessage);  }  } catch (IOException e) {  e.printStackTrace();  } finally {  try {  clientSocket.close();  if (inFromClient != null) {  inFromClient.close();  }  if (outToClient != null) {  outToClient.close();  }  } catch (IOException e) {  e.printStackTrace();  }  }  }   private void processMessage(String message) {  try {  String[] parts = message.split(":");  String command = parts[0].trim().toLowerCase();  switch (command) {  case "add":  if (parts.length == 3) {  String name = parts[1].trim();  int phone;  try {  phone = Integer.parseInt(parts[2].trim());  if (isValidPhone(phone)) {  addContact(name, phone);  outToClient.writeBytes("Contact added successfully.\n");  } else { outToClient.writeBytes("Invalid phone number. Please provide a valid integer phone number.\n");  }  } catch (NumberFormatException e) { outToClient.writeBytes("Invalid phone number format. Please provide a valid integer phone number.\n");  }  } else {  outToClient.writeBytes("Invalid command format. Please use 'add:name:phone'.\n");  }  break;  case "list":  outToClient.writeBytes(listContacts());  break;  case "search":  if (parts.length == 2) {  outToClient.writeBytes(searchContact(parts[1].trim()));  } else {  outToClient.writeBytes("Invalid command format. Please use 'search:name'.\n");  }  break;  default:  outToClient.writeBytes("Invalid command. Available commands: add, list, search.\n");  }  } catch (IOException e) {  e.printStackTrace();  } finally {  try {  if (clientSocket != null) {  clientSocket.close();  }  if (inFromClient != null) {  inFromClient.close();  }  if (outToClient != null) {  outToClient.close();  }  } catch (IOException e) {  e.printStackTrace();  }  }  }   } } // client package TCP; import java.io.\*; import java.net.Socket; import java.util.Scanner;  public class ContactClient {  public static void main(String[] args) throws IOException {  Scanner scanner = new Scanner(System.in);  Socket clientSocket = new Socket("localhost", 6789);  DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());  BufferedReader inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));   // Display available commands upon connection  System.out.println("Connected to server.");  System.out.println("Available commands: add, list, search.(add:<name>:<phone number>, list<enter>)");  System.out.println("Type 'quit' to exit.");   String userInput;  while (!(userInput = scanner.nextLine()).equalsIgnoreCase("quit")) {  outToServer.writeBytes(userInput + '\n');  String serverResponse = inFromServer.readLine();  System.out.println("Server: " + serverResponse);  }   clientSocket.close();  scanner.close();  } } |

|  |
| --- |
| Output: Server has started..... Client connected: /127.0.0.1 Connected to server. Available commands: add, list, search.(add:<name>:<phone number>, list<enter>) Type 'quit' to exit. add:hello:937743 Server: Contact added successfully. |

# 

### Program 5: File content sharing

The application consists of a client that requests a file from a server. The server reads the file's contents and sends it back to the client, which then displays it.

### **1. Client**

The Client class connects to the server using a socket, sends a filename to the server, and receives the file's contents.

|  |
| --- |
| package TCP;  import java.io.\*; import java.net.\*;  public class Client {  public static void main(String[] args) {  try (Socket socket = new Socket("localhost", 12345)) {  BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));  PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);  BufferedReader userInput = new BufferedReader(new InputStreamReader(System.in));   System.out.print("Enter filename to read: ");  String filename = userInput.readLine();   writer.println(filename);   String line;  while ((line = reader.readLine()) != null) {  System.out.println(line);  }  } catch (IOException e) {  e.printStackTrace();  }  } } |

### **2. Server**

The FileServer class listens for client connections on a specified port, reads the requested filename from the client, and sends back the file's contents.

|  |
| --- |
| package TCP;  import java.io.\*; import java.net.\*;  public class FileServer {  public static void main(String[] args) {  try (ServerSocket serverSocket = new ServerSocket(12345)) {  System.out.println("Server started. Listening on port " + 12345);   while (true) {  Socket clientSocket = serverSocket.accept();  System.out.println("Client connected: " + clientSocket);   try (  BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));  PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true)  ) {  String request = reader.readLine();  String filename = request;   try (BufferedReader fileReader = new BufferedReader(new FileReader(filename))) {  String line;  while ((line = fileReader.readLine()) != null) {  writer.println(line);  }  } catch (FileNotFoundException e) {  writer.println("File not found.");  }  }  }  } catch (IOException e) {  e.printStackTrace();  }  } } |

## **Output**

### **Client Output**

|  |
| --- |
| Enter filename to read: sample.txt This is the first line of the sample file. This is the second line of the sample file. |

**Server Output**

|  |
| --- |
| Server started. Listening on port 12345 Client connected: Socket[addr=/127.0.0.1,port=54321,localport=12345] |

### Program 6: Time Server

The main job of this project is to implement a simple time server-client system using Java's socket programming. The project consists of three main classes: TimeServer, TimeClient, and TimeServerClient.

**TimeServerClient.java**

public class TimeServerClient {

public static void main(String[] args) {

if (args.length > 0 && args[0].equalsIgnoreCase("server")) {

TimeServer.main(args);

} else if (args.length > 0 && args[0].equalsIgnoreCase("client")) {

TimeClient.main(args);

} else {

System.out.println("Usage: java TimeServerClient <server|client>");

}

}

}

**TimeServer.java**

import java.io.\*;

import java.net.\*;

import java.util.Date;

public class TimeServer {

public static void main(String[] args) {

try (ServerSocket serverSocket = new ServerSocket(12346)) {

System.out.println("Time Server is running...");

while (true) {

try (Socket clientSocket = serverSocket.accept();

PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true)) {

String currentTime = new Date().toString();

out.println(currentTime);

System.out.println("Sent time to client: " + currentTime);

} catch (IOException e) {

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

}

}

} catch (IOException e) {

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

e.printStackTrace();

}

}

}

Output:



**TimeClient.java**

import java.io.\*;

import java.net.\*;

public class TimeClient {

public static void main(String[] args) {

try (Socket socket = new Socket("localhost", 12346);

BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()))) {

System.out.println("Connected to the server");

String serverTime = in.readLine();

System.out.println("Current server time: " + serverTime);

} catch (UnknownHostException e) {

System.err.println("Don't know about host: localhost");

} catch (IOException e) {

System.err.println("Couldn't get I/O for the connection to: localhost");

e.printStackTrace();

}

}

}

Output:



# Topic 4: Threads

### Program 1: Thread counter

This Java code demonstrates a multi-threaded application where two threads increment a shared counter concurrently. It highlights the use of synchronization to ensure thread-safe operations on a shared resource.

|  |
| --- |
| public class ThreadedCounter {  public static void main(String[] args) {  Counter counter = new Counter();  Thread thread1 = new Thread(new CounterIncrementer(counter));  Thread thread2 = new Thread(new CounterIncrementer(counter));  thread1.start();  thread2.start();  try {  thread1.join();  thread2.join();  } catch (InterruptedException e) {  e.printStackTrace();  }  System.out.println("Final Counter Value: " + counter.getValue());  } }  class Counter {  private int value = 0;   public synchronized void increment() {  value++;  }   public int getValue() {  return value;  } } class CounterIncrementer implements Runnable {  private final Counter counter;   public CounterIncrementer(Counter counter) {  this.counter = counter;  }   @Override  public void run() {  for (int i = 0; i < 1000; i++) {  counter.increment();  }  } } |

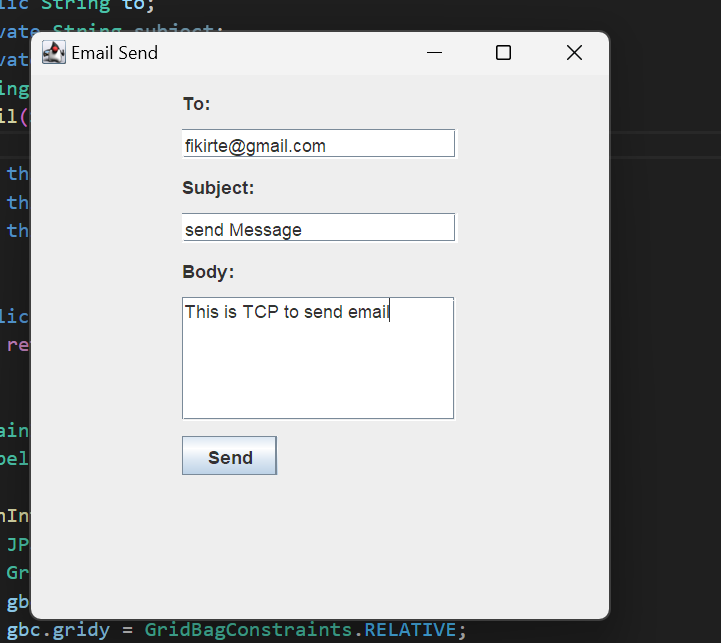
|  |
| --- |
| output Final Counter Value: 2000 |

### Program 2: send email project

The client send the receiver address and the body of the email then the server send the data to the receiver using TCP and thread

|  |
| --- |
| import java.io.ObjectInputStream; import java.io.ObjectOutputStream; import java.io.Serializable; import java.net.ServerSocket; import java.net.Socket;  import javax.swing.JButton; import javax.swing.JFrame; import javax.swing.JLabel; import javax.swing.JPanel; import javax.swing.JScrollPane; import javax.swing.JTextArea; import javax.swing.JTextField;  import java.awt.FlowLayout; import java.awt.GridBagConstraints; import java.awt.GridBagLayout; import java.awt.Insets; import java.awt.event.ActionEvent; import java.awt.event.ActionListener;   public class TCP {  public static void main(String[] args) throws Exception{  ServerThread serverThread = new ServerThread();  serverThread.start();   Thread.sleep(1000);  new MainInterface();  }  } class ServerThread extends Thread{  public void run(){  try{  ServerSocket serverSocket = new ServerSocket(5000);  Socket socket = serverSocket.accept();   ObjectOutputStream output =  new ObjectOutputStream(socket.getOutputStream());  ObjectInputStream input = new ObjectInputStream(socket.getInputStream());   Gmail gmail = (Gmail) input.readObject();  String success = gmail.sendEmail();   output.writeObject(success);    serverSocket.close();  }catch(Exception e){  e.printStackTrace();  }  } }  class Gmail implements Serializable{    private String to;  private String subject;  private String body;  String host = "127.0.0.1";  Gmail(String to,String subject,String body){    this.to = to;  this.subject = subject;  this.body = body;  }   public String sendEmail(){  return "Message Sucessfully send";  } } class MainInterface extends JFrame {  JLabel resultLabel;   MainInterface() {  JPanel mainpanel = new JPanel(new GridBagLayout());  GridBagConstraints gbc = new GridBagConstraints();  gbc.gridx = 0;  gbc.gridy = GridBagConstraints.RELATIVE;  gbc.anchor = GridBagConstraints.WEST;  gbc.insets = new Insets(5, 5, 5, 5);   JLabel toLabel = new JLabel("To:");  JLabel subjectLabel = new JLabel("Subject:");  JLabel bodyLabel = new JLabel("Body:");   JTextField toField = new JTextField(20);  JTextField subjectField = new JTextField(20);  JTextArea bodyField = new JTextArea(5, 20);  JScrollPane bodyScrollPane = new JScrollPane(bodyField);   JButton send = new JButton("Send");  resultLabel = new JLabel("");    mainpanel.add(toLabel, gbc);  mainpanel.add(toField, gbc);  mainpanel.add(subjectLabel, gbc);  mainpanel.add(subjectField, gbc);  mainpanel.add(bodyLabel, gbc);  mainpanel.add(bodyScrollPane, gbc);    mainpanel.add(send, gbc);  mainpanel.add(resultLabel, gbc);     // Button event  send.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  String to = toField.getText();  String subject = subjectField.getText();  String body = bodyField.getText();  toField.setText("");  subjectField.setText("");  bodyField.setText("");   Gmail gmail = new Gmail(to, subject, body);   try {  Socket client = new Socket("localhost", 5000);  ObjectOutputStream output = new ObjectOutputStream(client.getOutputStream());  ObjectInputStream input = new ObjectInputStream(client.getInputStream());   output.writeObject(gmail);  String result = (String) input.readObject();  resultLabel.setText(result);   client.close();  } catch (Exception error) {  error.printStackTrace();  }  }  });   JPanel panel = new JPanel(new FlowLayout(FlowLayout.CENTER));  panel.add(mainpanel);  add(panel);  setTitle("Email Send");  setSize(400, 400);  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  setLocationRelativeTo(null); // Centers the frame  setVisible(true);   } } |

**Output**



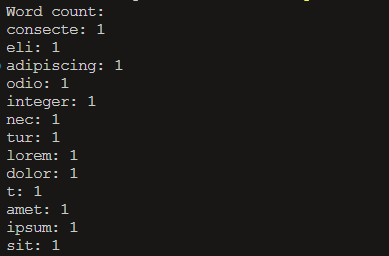
## 

### Program 3: Word count multi-Threaded

A java program for a concurrent text processing tool that performs word counting using multiple threads:

|  |
| --- |
| import java.util.HashMap; import java.util.Map;  public class TextProcessor {   public static void main(String[] args) {  String text = "Lorem ipsum dolor sit amet, consectetur adipiscing elit. Integer nec odio."; // Randomly generated sentence to test the program  int numThreads = 4;  Map<String, Integer> wordCount = new HashMap<>();  Thread[] threads = new Thread[numThreads];   int chunkSize = text.length() / numThreads;  for (int i = 0; i < numThreads; i++) {  int start = i \* chunkSize;  int end = (i == numThreads - 1) ? text.length() : (i + 1) \* chunkSize;  threads[i] = new Thread(new WordCounter(text.substring(start, end), wordCount));  threads[i].start();  }   for (Thread thread : threads) {  try {  thread.join();  } catch (InterruptedException e) {  e.printStackTrace();  }  }   System.out.println("Word count:");  for (Map.Entry<String, Integer> entry : wordCount.entrySet()) {  System.out.println(entry.getKey() + ": " + entry.getValue());  }  }   static class WordCounter implements Runnable {  private String text;  private Map<String, Integer> wordCount;   public WordCounter(String text, Map<String, Integer> wordCount) {  this.text = text;  this.wordCount = wordCount;  }   @Override  public void run() {  String[] words = text.split("\\s+");   for (String word : words) {  word = word.replaceAll("[^a-zA-Z]", "").toLowerCase();  if (!word.isEmpty()) {  wordCount.put(word, wordCount.getOrDefault(word, 0) + 1);  }  }  }  } } |

**Output:**



### Program 4: Factorial Calculator

This program calculates the factorial of a given number using multiple threads. Each thread will calculate a part of the factorial, and the main thread will merge the results.

|  |
| --- |
| import java.math.BigInteger; public class FactorialCalculator {   static class FactorialThread extends Thread {  private int start;  private int end;  private BigInteger result;   public FactorialThread(int start, int end) {  this.start = start;  this.end = end;  }   public void run() {  result = BigInteger.ONE;  for (int i = start; i <= end; i++) {  result = result.multiply(BigInteger.valueOf(i));  }  }   public BigInteger getResult() {  return result;  }  }   public static BigInteger calculateFactorial(int n, int numThreads) throws InterruptedException {  FactorialThread[] threads = new FactorialThread[numThreads];  int numbersPerThread = n / numThreads;  int remainingNumbers = n % numThreads;   for (int i = 0, start = 1; i < numThreads; i++) {  int end = start + numbersPerThread - 1;  if (i == numThreads - 1) {  end += remainingNumbers; // distribute remaining numbers to the last thread  }  threads[i] = new FactorialThread(start, end);  threads[i].start();  start = end + 1;  }   for (FactorialThread thread : threads) {  thread.join();  }  BigInteger result = BigInteger.ONE;  for (FactorialThread thread : threads) {  result = result.multiply(thread.getResult());  }  return result;  }   public static void main(String[] args) {  int number = 10;  int numThreads = 4;  try {  BigInteger factorial = calculateFactorial(number, numThreads);  System.out.println("Factorial of " + number + " is: " + factorial);  } catch (InterruptedException e) {  e.printStackTrace();  }  } } |

|  |
| --- |
| Output: Factorial of 10 is: 3628800 |

### 

### Program 5: File Swap

This document describes a Java-based application that uses multithreading to swap the contents of two files. The application consists of a main class that initiates two threads, each responsible for reading from one file and writing to the other.

### **Main Class: FileSwap**

The FileSwap class is the entry point of the application. It creates and starts two threads to perform the file swapping operation.

|  |
| --- |
| package Thread;  import java.io.\*;  public class FileSwap {  public static void main(String[] args) {  String filePath1 = "file1.txt";  String filePath2 = "file2.txt";   Thread swapThread1 = new Thread(new FileSwapTask(filePath1, filePath2));  Thread swapThread2 = new Thread(new FileSwapTask(filePath2, filePath1));  swapThread1.start();  swapThread2.start();   try {  swapThread1.join();  swapThread2.join();  System.out.println("File contents swapped successfully.");  } catch (InterruptedException e) {  e.printStackTrace();  }  }   static class FileSwapTask implements Runnable {  private String filePath1;  private String filePath2;   public FileSwapTask(String filePath1, String filePath2) {  this.filePath1 = filePath1;  this.filePath2 = filePath2;  }   @Override  public void run() {  try {  StringBuilder content = new StringBuilder();  try (BufferedReader reader = new BufferedReader(new FileReader(filePath1))) {  String line;  while ((line = reader.readLine()) != null) {  content.append(line).append("\n");  }  }   try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath2))) {  writer.write(content.toString());  }   } catch (IOException e) {  e.printStackTrace();  }  }  } } |

## **Output**

### **Example Files**

**file1.txt**:

|  |
| --- |
| This is file 1. It has some content. |

**file2.txt:**

|  |
| --- |
| **This is file 2. It also has content.** |

**Example Output**

|  |
| --- |
| File contents swapped successfully. |

### **Resulting Files**

* **file1.txt** (after swap):

|  |
| --- |
| This is file 2. It also has content. |

* **file2.txt** (after swap):

|  |
| --- |
| This is file 1. It has some content. |

### Program 6: Thread count

The main job of this project is to demonstrate the creation and execution of multiple threads in Java. It illustrates how to define a thread by extending the Thread class, start the threads, and manage their execution.

**BasicThreadExample.java**

class SimpleThread extends Thread {

private String threadName;

SimpleThread(String name) {

threadName = name;

}

public void run() {

for (int i = 0; i < 5; i++) {

System.out.println(threadName + " iteration: " + i);

try {

Thread.sleep(500); // Sleep for 500 milliseconds

} catch (InterruptedException e) {

System.out.println(threadName + " interrupted.");

}

}

System.out.println(threadName + " exiting.");

}

}

public class BasicThreadExample {

public static void main(String[] args) {

SimpleThread thread1 = new SimpleThread("Thread 1");

SimpleThread thread2 = new SimpleThread("Thread 2");

thread1.start();

thread2.start();

}

}

Output:

