21..Write a Java program to implement simple chat using TCP/IP

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

public class ChatClient {
    public static void main(String[] args) {
        try {
            Socket socket = new Socket("localhost", 12345);
            System.out.println("Connected to server");

            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

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

            String clientMessage, serverMessage;

            while (true) {
                clientMessage = clientInput.readLine();
                out.println(clientMessage);

                serverMessage = in.readLine();
                if (serverMessage != null) {
                    System.out.println("Server: " + serverMessage);
                }

                if ("bye".equalsIgnoreCase(serverMessage) ||
"bye".equalsIgnoreCase(clientMessage)) {
                    System.out.println("Chat ended");
                    socket.close();
                    break;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}


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

public class ChatServer {
    public static void main(String[] args) {
        try {
            ServerSocket serverSocket = new ServerSocket(12345);
            System.out.println("Server is running and waiting for a client to
```

```
connect...");

            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected");

            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

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

            String clientMessage, serverMessage;

            while (true) {
                clientMessage = in.readLine();
                if (clientMessage != null) {
                    System.out.println("Client: " + clientMessage);
                }

                serverMessage = serverInput.readLine();
                out.println(serverMessage);

                if ("bye".equalsIgnoreCase(serverMessage) ||
"bye".equalsIgnoreCase(clientMessage)) {
                    System.out.println("Chat ended");
                    serverSocket.close();
                    break;
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

ALGORITHM:

Here's the algorithm for the simple chat program using TCP/IP with a client and server:

### ChatClient:

1. **Connect to Server:**
   - Create a `Socket` to connect to the server on "localhost" at port 12345.

2. **Setup Communication Streams:**
   - Create a `BufferedReader` (`in`) to read messages from the server.
   - Create a `PrintWriter` (`out`) to send messages to the server.
   - Create a `BufferedReader` (`clientInput`) to read messages from the user.

3. **Message Exchange Loop:**
   - Enter a loop that continues until the user or the server types "bye."
   - Read a message from the user (`clientMessage`).
   - Send the message to the server using `out.println(clientMessage)`.
   - Read a message from the server (`serverMessage`).
   - Print the server's message if it's not null.

4. **Chat Termination:**
   - If either the server or the client types "bye," print "Chat ended."
   - Close the socket (`socket`).
   - Exit the loop and close the program.

### ChatServer:

1. **Create Server Socket:**
   - Create a `ServerSocket` on port 12345.
   - Print "Server is running and waiting for a client to connect..."

2. **Accept Client Connection:**
   - Accept a client connection using `serverSocket.accept()`.
   - Print "Client connected."

3. **Setup Communication Streams:**
   - Create a `BufferedReader` (`in`) to read messages from the client.
   - Create a `PrintWriter` (`out`) to send messages to the client.
   - Create a `BufferedReader` (`serverInput`) to read messages from the server user.

4. **Message Exchange Loop:**
   - Enter a loop that continues until the user or the client types "bye."
   - Read a message from the client (`clientMessage`).
   - Print the client's message if it's not null.
   - Read a message from the server user (`serverMessage`).
   - Send the server's message to the client using `out.println(serverMessage)`.

5. **Chat Termination:**
   - If either the server or the client types "bye," print "Chat ended."
   - Close the server socket (`serverSocket`).
   - Exit the loop and close the program.

### Note:
- The `PrintWriter` with `autoFlush` set to `true` (`new PrintWriter(socket.getOutputStream(), true)`) automatically flushes the output stream after each println, ensuring that the message is sent immediately.
- Error handling is done by printing the stack trace for any `IOException` that might occur during communication or socket operations.

This algorithm describes the basic steps performed by the client and server to enable a simple chat over a TCP/IP connection.

OUTPUT:
server

Server is running and waiting for a client to connect...
Client connected
Client: hii
hii
Client: what are you doing
see the program

client

Connected to server
hii
Server: hii
what are you doing
Server: see the program
simply
==========
==========

22.22.  Write a Java program to implement simple chat using UDP

```
import java.io.*;
import java.net.*;
class ChatServer
{

public static DatagramSocket serversocket;
public static DatagramPacket dp;
public static BufferedReader dis;
public static InetAddress ia;
public static byte buf[] = new byte[1024];
public static int cport = 789,sport=790;
public static void main(String[] a) throws IOException
{
serversocket = new DatagramSocket(sport);
dp = new DatagramPacket(buf,buf.length);
dis = new BufferedReader
(new InputStreamReader(System.in));
ia = InetAddress.getLocalHost();
System.out.println("Server is Running...");
while(true)
{
serversocket.receive(dp);
String str = new String(dp.getData(), 0,
dp.getLength());
```

```
if(str.equals("STOP"))
{
System.out.println("Terminated...");
break;
}
System.out.println("Client: " + str);
String str1 = new String(dis.readLine());
buf = str1.getBytes();
serversocket.send(new DatagramPacket(buf, str1.length(), ia, cport));

}
}
}

import java.io.*;
import java.net.*;
class ChatClient
{
public static DatagramSocket clientsocket;
public static DatagramPacket dp;
public static BufferedReader dis;
public static InetAddress ia;
public static byte buf[] = new byte[1024];
public static int cport = 789, sport = 790;
public static void main(String[] a) throws IOException
{
clientsocket = new DatagramSocket(cport);
dp = new DatagramPacket(buf, buf.length);
dis = new BufferedReader(new
InputStreamReader(System.in));
ia = InetAddress.getLocalHost();
System.out.println("Client is Running... Type 'STOP'to Quit");
while(true)
{
String str = new String(dis.readLine());
buf = str.getBytes();
if(str.equals("STOP"))
{
System.out.println("Terminated...");
clientsocket.send(new
DatagramPacket(buf,str.length(), ia,
sport));
break;
}
clientsocket.send(new DatagramPacket(buf,
str.length(), ia, sport));
clientsocket.receive(dp);
String str2 = new String(dp.getData(), 0,
dp.getLength());
System.out.println("Server: " + str2);
```

```
}
}
}
```

Algorithm:

Certainly! Here's an algorithmic representation of the UDP chat program for both the server (`ChatServer.java`) and the client (`ChatClient.java`):

### ChatServer Algorithm:
1. **Initialize Server:**
   - Open a `DatagramSocket` on a specified port (`sport`).
   - Create a `DatagramPacket` to receive data.

2. **Setup Input:**
   - Create a `BufferedReader` (`dis`) to read messages from the server user.
   - Obtain the local host's `InetAddress` (`ia`).

3. **Main Loop:**
   - Enter an infinite loop to continuously receive messages from the client.
   - Receive a message from the client using `serversocket.receive(dp)`.
   - Convert the received bytes to a string (`str`).
   - Check if the received message is "STOP."
     - If true, print "Terminated..." and exit the loop.
   - Print the client's message.
   - Read a message from the server user (`str1`).
   - Convert the message to bytes (`buf = str1.getBytes()`).
   - Send the message to the client using `serversocket.send(new DatagramPacket(buf, str1.length(), ia, cport))`.

### ChatClient Algorithm:
1. **Initialize Client:**
   - Open a `DatagramSocket` on a specified port (`cport`).
   - Create a `DatagramPacket` to send and receive data.

2. **Setup Input:**
   - Create a `BufferedReader` (`dis`) to read messages from the client user.
   - Obtain the local host's `InetAddress` (`ia`).

3. **Main Loop:**
   - Enter an infinite loop to continuously send and receive messages.
   - Read a message from the client user (`str`).
   - Convert the message to bytes (`buf = str.getBytes()`).
   - Send the message to the server using `clientsocket.send(new DatagramPacket(buf, str.length(), ia, sport))`.
   - Check if the sent message is "STOP."
     - If true, print "Terminated..." and exit the loop.
   - Receive a message from the server using `clientsocket.receive(dp)`.
   - Convert the received bytes to a string (`str2`).
   - Print the server's message.

### Note:
- The server and client use UDP (User Datagram Protocol) for communication.
- The server listens on a specified port, and the client communicates with the server on that port.
- The communication is done using `DatagramPacket` and `DatagramSocket`.
- The loop continues until either the server or the client types "STOP," indicating the end of the chat.

Ensure that the server and client are running on the same machine or update the `InetAddress` accordingly for communication between different machines.

OUTPUT:

```
C:\Users\SURIYA\Desktop\JAVA SEM\22>java ChatServer
Server is Running...
Client: hlo
hlo
Client: how are you
fine
Terminated...

C:\Users\SURIYA\Desktop\JAVA SEM\22>java ChatClient
Client is Running... Type 'STOP'to Quit
hlo
Server: hlo
how are you
Server: fine
STOP
Terminated...
```

==========

23(a) Write a Java program to create Tabbed Pane with Colors and Fruits as tab and to demonstrate JComboBox and JCheckbox components.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class FruitShopApp {
    private JFrame frame;
    private JTabbedPane tabbedPane;
    private JPanel colorsPanel, fruitsPanel;
    private JComboBox<String> colorComboBox, fruitComboBox;
    private JCheckBox organicCheckbox, homeDeliveryCheckbox;

    public FruitShopApp() {
```

```
        frame = new JFrame("Fruit Shop App");
        frame.setSize(400, 300);
        frame.setLayout(new BorderLayout());

        // Create a JTabbedPane
        tabbedPane = new JTabbedPane();

        // Colors Tab
        colorsPanel = createColorsPanel();
        tabbedPane.addTab("Colors", colorsPanel);

        // Fruits Tab
        fruitsPanel = createFruitsPanel();
        tabbedPane.addTab("Fruits", fruitsPanel);

        // Add the tabbedPane to the frame
        frame.add(tabbedPane, BorderLayout.CENTER);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

    private JPanel createColorsPanel() {
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());

        // Color JComboBox
        colorComboBox = new JComboBox<>(new String[]{"Red", "Blue", "Green",
"Yellow"});
        panel.add(new JLabel("Select Color:"));
        panel.add(colorComboBox);

        // Organic Checkbox
        organicCheckbox = new JCheckBox("Organic");
        panel.add(organicCheckbox);

        // Order Button
        JButton orderButton = new JButton("Place Order");
        panel.add(orderButton);

        // ActionListener for the orderButton
        orderButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String selectedColor = (String) colorComboBox.getSelectedItem();
                boolean isOrganic = organicCheckbox.isSelected();

                // Perform order logic (not implemented in this example)
                System.out.println("Order placed for Color: " + selectedColor + ",
Organic: " + isOrganic);
```

```
            }
        });

        return panel;
    }

    private JPanel createFruitsPanel() {
        JPanel panel = new JPanel();
        panel.setLayout(new FlowLayout());

        // Fruit JComboBox
        fruitComboBox = new JComboBox<>(new String[]{"Apple", "Banana", "Orange",
"Grapes"});
        panel.add(new JLabel("Select Fruit:"));
        panel.add(fruitComboBox);

        // Home Delivery Checkbox
        homeDeliveryCheckbox = new JCheckBox("Home Delivery");
        panel.add(homeDeliveryCheckbox);

        // Order Button
        JButton orderButton = new JButton("Place Order");
        panel.add(orderButton);

        // ActionListener for the orderButton
        orderButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String selectedFruit = (String) fruitComboBox.getSelectedItem();
                boolean isHomeDelivery = homeDeliveryCheckbox.isSelected();

                // Perform order logic (not implemented in this example)
                System.out.println("Order placed for Fruit: " + selectedFruit + ",
Home Delivery: " + isHomeDelivery);
            }
        });

        return panel;
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new FruitShopApp();
            }
        });
    }
}
```

Algorithm:

Sure, here's the algorithmic representation for the Java program to create a `JTabbedPane` with "Colors" and "Fruits" tabs, demonstrating `JComboBox` and `JCheckBox` components for a fruit shop:

### Algorithm:

1. **Initialize GUI Components:**
   - Create a `JFrame` for the main window.
   - Set the layout of the frame to `BorderLayout`.
   - Create a `JTabbedPane` (`tabbedPane`) to hold the tabs.

2. **Create "Colors" Tab:**
   - Create a panel (`colorsPanel`) for the "Colors" tab.
   - Add a `JComboBox` (`colorComboBox`) with color options.
   - Add a `JCheckBox` (`organicCheckbox`) for organic selection.
   - Add a "Place Order" button.
   - Attach an `ActionListener` to the button to handle order placement.
     - Get the selected color from the `colorComboBox`.
     - Check if the `organicCheckbox` is selected.
     - Print or handle the order details (not fully implemented in the example).

3. **Create "Fruits" Tab:**
   - Create a panel (`fruitsPanel`) for the "Fruits" tab.
   - Add a `JComboBox` (`fruitComboBox`) with fruit options.
   - Add a `JCheckBox` (`homeDeliveryCheckbox`) for home delivery selection.
   - Add a "Place Order" button.
   - Attach an `ActionListener` to the button to handle order placement.
     - Get the selected fruit from the `fruitComboBox`.
     - Check if the `homeDeliveryCheckbox` is selected.
     - Print or handle the order details (not fully implemented in the example).

4. **Add Tabs to the TabbedPane:**
   - Add the "Colors" and "Fruits" panels as tabs to the `tabbedPane`.

5. **Add TabbedPane to Frame:**
   - Add the `tabbedPane` to the `BorderLayout.CENTER` of the frame.

6. **Display the Frame:**
   - Set the default close operation of the frame to `EXIT_ON_CLOSE`.
   - Set the frame visibility to `true`.

### Note:
- The program uses Swing components for GUI creation.
- The order placement logic in the `ActionListener` is a placeholder and should be adapted to your specific requirements.
- The program is structured to handle user interaction with a tabbed interface, allowing them to select options and place orders in a fruit shop scenario.

This algorithm provides a high-level overview of the steps involved in creating the Java program. The specific implementation details are provided in the Java code.

OUTPUT:

```
Order placed for Color: Blue, Organic: false
Order placed for Fruit: Orange, Home Delivery: true
Order placed for Fruit: Orange, Home Delivery: true
```

==========

23.b). (b) Write a Java program to find minimum of n numbers.

```
import java.util.Scanner;

public class MinimumNumberFinder {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter the number of elements (n): ");
        int n = scanner.nextInt();

        if (n <= 0) {
            System.out.println("Please enter a valid number of elements (n > 0).");
            return;
        }

        System.out.println("Enter " + n + " numbers:");

        int minValue = Integer.MAX_VALUE;

        for (int i = 0; i < n; i++) {
            System.out.print("Enter number " + (i + 1) + ": ");
            int currentNumber = scanner.nextInt();

            if (currentNumber < minValue) {
                minValue = currentNumber;
            }
        }

        System.out.println("Minimum value: " + minValue);

        scanner.close();
    }
}
```

ALGORITHM:

Certainly! Here's an algorithmic representation of the Java program to find the minimum of `n` numbers:

### Algorithm:

1. **Initialize Variables:**
   - Initialize a variable `n` to store the number of elements.
   - Initialize a variable `minValue` to store the minimum value (set to `Integer.MAX_VALUE` initially).
   - Create a `Scanner` object for user input.

2. **Prompt for User Input:**
   - Print a message to prompt the user to enter the number of elements (`n`).
   - Read and store the value of `n` from the user.

3. **Input Validation:**
   - Check if `n` is greater than 0.
     - If not, print an error message and exit.

4. **Number Entry Loop:**
   - Print a message to prompt the user to enter `n` numbers.
   - Use a loop to iterate `n` times.
     - Prompt the user to enter the current number.
     - Read and store the current number in a variable `currentNumber`.

5. **Minimum Value Search:**
   - Check if `currentNumber` is less than `minValue`.
     - If true, update `minValue` with the value of `currentNumber`.

6. **Display Result:**
   - Print the minimum value (`minValue`) found.

7. **Closing Scanner:**
   - Close the `Scanner` object.

### Note:
- The algorithm is structured to find the minimum value among `n` numbers entered by the user.
- The `Integer.MAX_VALUE` is used as an initial value for `minValue` to ensure any entered value will be smaller.
- The program uses a loop for user input, allowing the user to enter multiple numbers.
- Input validation is included to handle cases where the user enters an invalid number of elements.

This algorithm provides a step-by-step guide for the logic implemented in the Java program to find the minimum of `n` numbers.

OUTPUT:

```
Enter the number of elements (n): 5
Enter 5 numbers:
Enter number 1: 1
Enter number 2: 2
```

```
Enter number 3: 3
Enter number 4: 4
Enter number 5: 5
Minimum value: 1

================================================================================
============================
24.a)Write a Java program to implement User interface screen for Blood Donor using
AWT controls

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

public class BloodDonorUI extends Frame {
    private Label nameLabel, ageLabel, bloodGroupLabel;
    private TextField nameTextField, ageTextField;
    private Choice bloodGroupChoice;
    private Button submitButton;

    public BloodDonorUI() {
        // Set frame properties
        setTitle("Blood Donor Registration");
        setSize(400, 300);
        setLayout(new GridLayout(4, 2));
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent windowEvent) {
                System.exit(0);
            }
        });

        // Initialize components
        nameLabel = new Label("Name:");
        ageLabel = new Label("Age:");
        bloodGroupLabel = new Label("Blood Group:");

        nameTextField = new TextField();
        ageTextField = new TextField();

        // Populate blood group choices
        bloodGroupChoice = new Choice();
        bloodGroupChoice.add("A+");
        bloodGroupChoice.add("A-");
        bloodGroupChoice.add("B+");
        bloodGroupChoice.add("B-");
        bloodGroupChoice.add("O+");
        bloodGroupChoice.add("O-");
        bloodGroupChoice.add("AB+");
        bloodGroupChoice.add("AB-");

        submitButton = new Button("Submit");
```

```
        submitButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                submitButtonClicked();
            }
        });

        // Add components to the frame
        add(nameLabel);
        add(nameTextField);
        add(ageLabel);
        add(ageTextField);
        add(bloodGroupLabel);
        add(bloodGroupChoice);
        add(new Label()); // Empty space
        add(submitButton);

        // Set frame visibility
        setVisible(true);
    }

    private void submitButtonClicked() {
        String name = nameTextField.getText();
        String age = ageTextField.getText();
        String bloodGroup = bloodGroupChoice.getSelectedItem();

        // Perform data validation and processing (not implemented in this example)
        System.out.println("Donor Information:");
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Blood Group: " + bloodGroup);

        // You can add further logic here for data processing or storage.
    }

    public static void main(String[] args) {
        new BloodDonorUI();
    }
}
```

Alogorithm:

Certainly! Below is an algorithmic representation of the Java program to implement a user interface screen for a Blood Donor using AWT controls:

### Algorithm:

1. **Initialize Frame:**
   - Create a new `Frame` object to represent the main window.
   - Set the title, size, layout, and window-closing event.

2. **Initialize UI Components:**
   - Create `Label` objects for "Name:", "Age:", and "Blood Group:".
   - Create `TextField` objects for name and age.
   - Create a `Choice` object for the blood group dropdown.
   - Create a `Button` for submission.

3. **Configure Frame Layout:**
   - Set the layout of the frame to a `GridLayout` (4 rows, 2 columns).

4. **Add Components to Frame:**
   - Add labels, text fields, choice, and button to the frame.

5. **Handle Window-Closing Event:**
   - Add a window-closing event listener to the frame to exit the application when the user closes the window.

6. **Create ActionListener for Submission Button:**
   - Add an `ActionListener` to the submit button.
   - Implement the `actionPerformed` method to handle button clicks.
     - Get the text from name and age text fields.
     - Get the selected blood group from the choice dropdown.
     - Perform data validation and processing (for example, print information to the console in this example).

7. **Display the Frame:**
   - Set the frame visibility to true.

8. **Submit Button Click Handling:**
   - Implement the `submitButtonClicked` method to process the information entered when the submit button is clicked.

### Note:
- The program uses AWT controls for basic UI components.
- The `main` method creates an instance of the `BloodDonorUI` class to display the UI.
- In a real-world application, you would typically replace the console print statements in the `submitButtonClicked` method with your desired logic for data validation, storage, or transmission.

This algorithm provides a high-level overview of the steps involved in creating the Java program to implement a user interface for a Blood Donor using AWT controls.

OUTPUT:

Donor Information:
Name: s
Age: 21
Blood Group: O+
==========
24.b)Write a Java program to reverse the digits of the given numbers.

```java
import java.util.Scanner;

public class ReverseDigits {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Prompt user for input
        System.out.print("Enter a number: ");
        int number = scanner.nextInt();

        // Reverse the digits
        int reversedNumber = reverseDigits(number);

        // Display the result
        System.out.println("Original Number: " + number);
        System.out.println("Reversed Number: " + reversedNumber);

        scanner.close();
    }

    // Method to reverse the digits of a number
    private static int reverseDigits(int num) {
        int reversedNum = 0;

        while (num != 0) {
            int digit = num % 10;
            reversedNum = reversedNum * 10 + digit;
            num /= 10;
        }

        return reversedNum;
    }
}
```

ALGORITHM:

Certainly! Here's the algorithmic representation of the Java program to reverse the digits of a given number:

### Algorithm:

1. **Initialize Variables:**
   - Create a `Scanner` object for user input.
   - Declare variables `number` and `reversedNumber` to store the original and reversed numbers.

2. **Prompt for User Input:**
   - Print a message to prompt the user to enter a number.

   - Read and store the entered number in the variable `number`.

3. **Reverse Digits:**
   - Call the `reverseDigits` method with the entered number.
   - In the `reverseDigits` method:
     - Initialize `reversedNum` to 0.
     - Use a while loop to extract digits from the right of the number:
       - Calculate the last digit using the modulo operation.
       - Multiply `reversedNum` by 10 and add the last digit.
       - Divide the number by 10 to remove the last digit.
     - Continue until the number becomes 0.
     - Return the `reversedNum`.

4. **Display Result:**
   - Print the original and reversed numbers.

5. **Closing Scanner:**
   - Close the `Scanner` object to free up resources.

### Note:
- The `reverseDigits` method encapsulates the logic for reversing the digits of a number.
- The program uses a simple mathematical approach to reverse the digits.

This algorithm provides a step-by-step guide for the logic implemented in the Java program to reverse the digits of a given number.

OUTPUT:

```
Enter a number: 32
Original Number: 32
Reversed Number: 23
```

==========

25.(a) Write a Java program to store information into file and read information from file using CharacterStream.

```
import java.io.*;

public class CopyFile {

public static void main(String[] args) throws IOException {

FileReader in = null;

FileWriter out = null;

try {
 int c;
```

```
in = new FileReader("input.txt");

out = new FileWriter("output.txt");

while ((c = in.read()) != -1) {
out.write(c);
System.out.println((char)c);

}
System.out.println("success");

}
catch(Exception e){
System.out.println(e);
}
finally {

if (in != null) { in.close(); }

if (out!= null) { out.close();}

}
}
}
```

==========

25.b)  Write a Java program to display personal details using parameter.

```
public class DisplayPersonalDetails {
    public static void main(String[] args) {
        // Call the displayDetails method with personal information
        displayDetails("John Doe", 25, "Male", "john@example.com", "123-456-7890");
    }

    // Method to display personal details using parameters
    private static void displayDetails(String name, int age, String gender, String
email, String phoneNumber) {
        System.out.println("Personal Details:");
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Gender: " + gender);
        System.out.println("Email: " + email);
        System.out.println("Phone Number: " + phoneNumber);
    }
}
```

OUTPUT:

```
Personal Details:
Name: John Doe
```

```
Age: 25
Gender: Male
Email: john@example.com
Phone Number: 123-456-7890
================================================================================
Applet parameter
import java.applet.Applet;
import java.awt.Graphics;

public class DisplayPersonalDetailsApplet extends Applet {
    private String name, age, gender, email, phoneNumber;

    public void init() {
        // Get parameters from the HTML file
        name = getParameter("name");
        age = getParameter("age");
        gender = getParameter("gender");
        email = getParameter("email");
        phoneNumber = getParameter("phoneNumber");
    }

    public void paint(Graphics g) {
        // Display personal details on the applet window
        g.drawString("Personal Details:", 20, 20);
        g.drawString("Name: " + name, 20, 40);
        g.drawString("Age: " + age, 20, 60);
        g.drawString("Gender: " + gender, 20, 80);
        g.drawString("Email: " + email, 20, 100);
        g.drawString("Phone Number: " + phoneNumber, 20, 120);
    }
}

<!DOCTYPE html>
<html>
<head>
    <title>Display Personal Details Applet</title>
</head>
<body>
    <applet code="DisplayPersonalDetailsApplet.class" width="400" height="200">
        <param name="name" value="John Doe">
        <param name="age" value="25">
        <param name="gender" value="Male">
        <param name="email" value="john@example.com">
        <param name="phoneNumber" value="123-456-7890">
    </applet>

</body>
</html>
```