**Muhammad Shahid Bhatti**

**SUBMITTED TO:**

**Muhibba Fatima (SP24-BSE-B-138)**

**SUBMITTED BY:**

**OBJECT ORIENTED PROGRAMMING**

**SUBJECT:**

**ASSIGNMENT**

**Explanation of the output:**

**Display Function:**

Here is the display function that I have made:

public void displayAllMessages(){

System.out.println("Displaying all messages : ");

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

System.out.println("Message for receiver" +(i+1)+ " :");

for(int j=0;j<messageCount[i];j++){

System.out.println(messages[i][j].toString());

System.out.println("-----------------------");

}

}

}

**Explanation:**

When the user selects to display all messages, then the program iterates through all receivers and print the messages in a structured format. Each message includes the sender, receiver, message content, id, status and timestamp. Receiver 1 got a message “Hello” with ID 0 which was seen at the given timestamp.

**Send Message Function:**

public void sendMessages(String[] contacts) {

Scanner sc = new Scanner(System.in);

System.out.println("Enter message content : ");

String messageContent = sc.nextLine();

System.out.println("Enter the receiver index (0 to " +(totalReceivers - 1) + "):");

int receiverIndex = sc.nextInt();

sc.nextLine();

if(receiverIndex < 0 || receiverIndex >= totalReceivers){

System.out.println("Invalid receiver index. Please provide a valid index.");

return;

}

if(contacts != null && receiverIndex < contacts.length){

addMessage(contacts, messageContent, false, receiverIndex);

System.out.println("Message sent to receiver " + (receiverIndex + 1) + ": " + messageContent);

} else {

System.out.println("Contact array is null or invalid index.");

}

}

if(receiverIndex < 0 || receiverIndex >= totalReceivers){

System.out.println("Invalid receiver index. Please provide a valid index.");

return;

}

if(contacts != null && receiverIndex < contacts.length){

addMessage(contacts, messageContent, false, receiverIndex);

System.out.println("Message sent to receiver " + (receiverIndex + 1) + ": " + messageContent);

} else {

System.out.println("Contact array is null or invalid index.");

}

}

**Explanation:**

This function prompts the user for two inputs: Message content and receiver index. The method checks if the receiver index is within the range and also checks that the contact array is not null. If the validation checks the method calls addMessage() function to store the message in appropriate receiver’s message array. In the example, the user chooses option 2 that takes the input “Hope you are doing well!” and then the message sent to receiver 6.

**Search message Function:**

public void searchMessage(){

Scanner sc = new Scanner(System.in);

System.out.println("Enter message id to search message : ");

String messageID = sc.nextLine();

boolean found = false;

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

for(int j=0;j<messageCount[i];j++){

if(messages[i][j].getMesageID().equals(messageID)){

System.out.println("Message found : \n" + messages[i][j].toString());

found = true;

break;

}

}

if(found) break;

}

if (!found){

System.out.println("Message not found");

}

}

**Explanation:**

We can search a message by ID in this function. If we enter an ID, it will iterate through all receivers and their respective messages. Then it checks if the ID of any message matches ID that we entered**.** If the ID matches, then it will print the details of that ID. Here we set a flag (found) to true that indicates that the message has been located. If found, then breaks out of the loop to stop searching further. If the message Id not matches, then it will print “No message found”. In the example, we entered the ID 9 and it displays the details of that ID.

**Delete Message Function:**

public void deleteMessage(){

Scanner sc = new Scanner(System.in);

System.out.println("Enter message id to delete message : ");

String messageID = sc.nextLine();

boolean found = false;

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

for(int j=0;j<messageCount[i];j++){

if(messages[i][j].getMesageID().equals(messageID)){

found = true;

for(int k=j;k<messageCount[i]-1;k++){

messages[i][k] = messages[i][k+1];

}

messages[i][--messageCount[i]] = null;

System.out.println("Message deleted successfully.");

break;

}

}

if(found) break;

}

if(!found) {

System.out.println("Message not found.");

}

}

**Explanation:**

We can delete a message by ID in this function. If we enter an ID, it will iterate through all receivers and their respective messages. Then it checks if the ID of any message matches ID that we entered. Message is deletedby shifting all the subsequent messages one index forward. The last message in the array is set to null. The total message count for the receiver is decremented to reflect deletion. If it is deleted, then the message “Message deleted successfully” is printed and it breaks out of the loop. Otherwise it will print “Message not found”. In example output, we entered the ID 12 and it got deleted.

**Displaying seen Messages Function:**

**Displaying Unseen Messages Function:**

**Explanation:**

This function allows users to display seen messages. The method iterates through all receivers using a nested loop. The outer loop iterates through all receivers and inner loop iterates through each message for that particular receiver. Then for all messages, it checks if the status of the message is true. Those messages whose status are shown as seen are printed which include the information as sender, receiver, content, id, status, timestamp. In example output, we select the option 5 that will display seen messages. The messages with IDs 0,2,3,5,7,8,11,13,14,16,18,19 are displayed because their status is seen.

public void displaySeenMessages(){

System.out.println("Seen messages : ");

boolean hasSeen = false;

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

for(int j=0;j<messageCount[i];j++){

if(messages[i][j].getStatus()){

System.out.println(messages[i][j].toString());

System.out.println("-----------------------");

hasSeen = true;

}

}

}

if(!hasSeen) {

System.out.println("No seen messages");

}

}

public void displayUnseenMessages(){

System.out.println("Unseen messages : ");

boolean hasUnseen = false;

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

for(int j=0;j<messageCount[i];j++){

if(!messages[i][j].getStatus()) {

System.out.println(messages[i][j].toString());

System.out.println("-----------------------");

messages[i][j].setStatus(true);

hasUnseen = true;

}

}

}

if(!hasUnseen) {

System.out.println("No unseen messages");

}

}

**Explanation:**

This function allows users to display unseen messages. The method iterates through all receivers using a nested loop. The outer loop iterates through all receivers and inner loop iterates through each message for that particular receiver. Then for all messages, it checks if the status of the message is false. Those messages whose status are shown as unseen are printed which include the information as sender, receiver, content, id, status, timestamp. In example output, we select the option 6 that will display unseen messages. The messages with IDs 1,4,6,9,10,15,17,20 are displayed because their status is unseen.

**Shuffle Messages Function:**

public void shuffleMessages(){

int totalMessages = 0;

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

totalMessages += messageCount[i];

}

Message[] allMessages = new Message[totalMessages];

int index = 0;

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

for(int j=0;j<messageCount[i];j++){

allMessages[index++] = messages[i][j];

}

}

Random rand = new Random();

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

int randomIndex = rand.nextInt(totalMessages);

Message temp = allMessages[i];

allMessages[i] = allMessages[randomIndex];

allMessages[randomIndex] = temp;

}

System.out.println("Messages in random order : ");

for (Message message : allMessages){

System.out.println(message.toString());

System.out.println("-----------------------");

}

}

**Explanation:**

Its functionality includes: Gathering all messages from all receivers, then randomize the order of the messages and display all the messages in a random order.

Its process includes: First of all, it iterates through each receiver and sums up the number of messages for all receivers then it creates an array to hold all messages for all receivers and copy all messages into allMessages. Then it shuffles the messages and display them. Example output shows all the messages in a random order.

**Display Message between sender and a particular receiver Function:**

public void displayMessageBetweenSenderAndReceiver(String receiverNumber){

boolean found = false;

System.out.println("Message between" +SENDER\_NUMBER+ " and " +receiverNumber+ ":");

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

for(int j=0;j<messageCount[i];j++){

if (messages[i][j].getReceiverNumber().equals(receiverNumber) && messages[i][j].getSenderNumber().equals(SENDER\_NUMBER)) {

System.out.println(messages[i][j].getMessageContent());

found = true;

}

}

}

if (!found){

System.out.println("No messages found between these numbers");

}

}

**Explanation:**

It displays messages between a sender and a specific receiver. The user inputs a receiver number, then it iterates through all the messages in the system. It then checks if the message is from sender number and if the receiver number matches the provided number. If matches, then it prints the message content. If not matches, then it will display “No message found”. Example output prints the message between the sender and the specific receiver number “03005430006”.

**Exiting:**

while (true){

System.out.println("Enter 1 to display all messages, 2 to send message to a particular receiver, 3 to search a message, 4 to delete a message, 5 to display seen messages, 6 to display unseen messages, 7 to shuffle messages, 8 to display message between sender and any receiver number, 9 to exit : ");

int choice = sc.nextInt();

sc.nextLine();

switch (choice){

case 1:

app.displayAllMessages();

break;

case 2:

app.sendMessages(contacts);

break;

case 3:

app.searchMessage();

break;

case 4:

app.deleteMessage();

break;

case 5:

app.displaySeenMessages();

break;

case 6:

app.displayUnseenMessages();

break;

case 7:

app.shuffleMessages();

break;

case 8:

System.out.println("Enter receiver number : ");

String receiverNumber = sc.nextLine();

app.displayMessageBetweenSenderAndReceiver(receiverNumber);

break;

case 9:

System.out.println("Exiting...");

sc.close();

return;

default:

System.out.println("Invalid choice. Please try again");

}

app.sendMessages(contacts);

break;

case 3:

app.searchMessage();

break;

case 4:

app.deleteMessage();

break;

case 5:

app.displaySeenMessages();

break;

case 6:

app.displayUnseenMessages();

break;

case 7:

app.shuffleMessages();

break;

case 8:

System.out.println("Enter receiver number : ");

String receiverNumber = sc.nextLine();

app.displayMessageBetweenSenderAndReceiver(receiverNumber);

break;

case 9:

System.out.println("Exiting...");

sc.close();

return;

default:

System.out.println("Invalid choice. Please try again");

}

**Explanation:**

This loop continuously displays options to interact with the system until the user decides to exit. The menu offers choices for action. The user inputs a number according to their desired action. Example output shows the user chooses 9 and exit the loop.

**“UML Diagram”**

Here is the UML diagram of our code:

It outlines the relationship between Message, MessaginApp and Main classes along with their attributes and methods.