**Chat Application**



An

ObjectOriented Programming through Java Course Project Report

in partial fulfilment of the degree

**Bachelor of Technology**

in

**Computer Science & Engineering**

**By**

M.D Shah Nawaz 2203A51L87

M.D Abdul Rizwan 2203A51L36

M.D Abdul 2203A51L34

T. Maruthi Ganesh 2203A51L60

**Under the guidance of**

**Padala Sravan**

**Submitted to**





**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**CERTIFICATE**

This is to certify that the **Object Oriented Programming through Java Course Project** Report entitled **“Chat Application”** is a record of bonafide work carried out by the student(s) **MD. Shah Nawaz**, **MD. Abdul Rizwan**, **MD. Abdul, T. Maruthi Ganesh** bearing

Roll No(s) **2203A51L87**, **2203A51L36, 2203A51L34, 2203A51L60** during the academic year 202223 in partial fulfillment of the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering** by the SR University.

**Lab Incharge Head of the Department**

**Table of Contents**

Chapter No. Title Page No.

1. Introduction

1.1. ABSTRACT 1

1.2. Overview 3

1.3. Problem Statement 3

1.4. Objective 4

1. System Analysis

2.1. Existing System 5

2.2. Proposed System 5

2.3. Advantages 6

1. Literature Survey 7
2. System Requirements 8
3. Elements used in the Project 10

5.1. Java AWT 19

5.2. Java Swing 22

1. Design

6.1 Architecture/Workflow 27

6.2 Screens 29

1. Implementation

7.1 Code 32

1. Test Cases 53
2. Conclusion 55
3. Reference 56

**ABSTRACT**

The "Chat Application" is a Java Swingbased graphical user interface (GUI) program designed to facilitate realtime textbased communication. This project provides users with the ability to engage in personal and group chats in a userfriendly environment. The application offers an initial login screen for authentication and then allows users to select between personal and group chats.

At the outset, the application provides a secure login screen where users are required to input their username and password. The application verifies this information against predefined values to grant access, after Login process, users can choose between two chat options: "Personal Chat" and "Group Chat", The "Personal Chat" option enables oneonone conversations, while "Group Chat" allows users to specify the number of friends they want to include in a group chat. This choice facilitates both private and group communication. Messages are sent in realtime, displayed instantly in the user's chat window, and archived for future reference.

**INTRODUCTION**

In this project, the primary goal was to develop a robust and userfriendly Chat Application that addresses the evolving communication needs of modern organizations and individuals. The project aimed to create a versatile platform for realtime communication, encompassing personal and group chats, with an emphasis on security and ease of use.

A Chat Application is a versatile and indispensable software tool designed for realtime communication. In an increasingly connected world, this application provides a means for individuals and groups to engage in conversations, share information, and collaborate efficiently. Its core features include personal and group chat functionality, realtime updates, secure data transmission, and userfriendly interfaces. Chat applications are not only valuable for casual conversations but also play a vital role in modern business operations. They enable organizations to reduce email, enhance collaboration among employees, and facilitate quick decisionmaking. Additionally, in the context of remote work and the digital workplace, chat applications become central hubs for communication, making it possible for teams to remain connected and productive, regardless of their physical locations. Whether used in corporate environments, educational institutions, healthcare facilities, or customer support centers, chat applications have become essential tools for streamlining communication and fostering connectivity in today's fastpaced world. One of the significant advantages of the Chat Application is that it's not just for large corporations. It's a gamechanger for small businesses, educational institutions, healthcare providers, government agencies, and nonprofit organizations too. Everyone can benefit from enhanced communication, improved collaboration, and, ultimately, greater productivity.

That’s what the Chat Application is all about – making r worklife easier, more productive, and more connected. It's not just a tool; it's a gamechanger for the way we work and communicate.

The reason why many chat applications continue to enter the market with similar messaging concepts is that realtime communication is a fundamental and everevolving aspect of our interconnected world. These applications aim to address specific user needs, preferences, and niches, even within the messaging space. r project can connect with this concept by offering a unique selling point or value proposition. Whether it's focusing on enhanced security, tailored features for specific user groups, or a simplified and intuitive user experience, r application can stand out by addressing specific communication needs, more interactive User interface and preferences, making it a valuable addition to the diverse landscape of chat applications.

**Overview:**

The provided code represents the foundation for a simple chat application in Java using the Swing framework. The project aims to create a basic chat application with a graphical user interface. Users can log in and choose between personal and group chat options. The application features chat windows for individual and group conversations, including message input fields, message sending, and message display.

This project provides users with the ability to engage in personal and group chats in a userfriendly environment. This application begins with a secure login screen, where users must enter their username and password. After successful authentication, users can choose between "Personal Chat" and "Group Chat" options. "Personal Chat" facilitates oneonone conversations, while "Group Chat" lets users create group chats with a specified number of friends. At the outset, the application provides a secure login screen where users are required to input their username and password. The application verifies this information against predefined values to grant access, after Login process, users can choose between two chat options: "Personal Chat" and "Group Chat", The "Personal Chat" option enables oneonone conversations, while "Group Chat" allows users to specify the number of friends they want to include in a group chat. This choice facilitates both private and group communication.

This code serves as a starting point for a simple chat application and can be extended and enhanced to meet more advanced requirements.

**Problem statement:**

"Develop a chat application that facilitates realtime communication between users. The application should allow users to exchange text messages, multimedia content, and create personal or group chat sessions. The primary goal is to provide a seamless, secure, and userfriendly platform for individuals to connect and communicate, addressing the need for efficient and reliable digital communication in today's interconnected world. The project should encompass clientserver architecture, user authentication, data transmission, realtime message delivery, and intuitive user interface design, ensuring the privacy and security of user interactions.

Creating a chat application as a project provides an excellent opportunity to develop a comprehensive understanding of software development and the intricacies of building realtime communication systems. Chat applications are not only relevant and widely used in today's digital world, but they also encompass various complex aspects that make them an engaging and valuable project choice.

Firstly, a chat application project allows to explore the world of network programming, emphasizing clientserver architecture, data transmission, and handling. will learn how to establish and maintain connections between users, manage data flow, and ensure secure communication.

**Objective**

The objective of this project is to develop a Java Swingbased chat application that provides a userfriendly interface for both personal and group conversations. The application's primary goal is to enable users to engage in textbased communication with their friends and colleagues.

The project aims to achieve the following key objectives:

1. **User Authentication**: Implement a basic login system to verify the identity of users, allowing access only to authenticated users. This enhances the security and privacy of the chat application.

2. **Personal and Group Chat**: Provide users with the flexibility to engage in oneonone personal chats as well as group chats. Personal chats facilitate private conversations, while group chats are ideal for communication with multiple participants.

3. **RealTime Messaging**: Enable realtime messaging, allowing users to send and receive messages instantly. The chat interface provides a seamless and responsive communication platform.

4. **UserFriendly Interface**: Create an intuitive and visually appealing graphical user interface (GUI) that enhances the overall user experience. This includes profile pictures, text input fields, and easily readable message display.

5. **Customization**: Make it possible for users to customize their profiles, including profile pictures, user names, and other preferences, to personalize their chat experience.

6. **Message Management**: Implement features for composing and sending messages, displaying messages in a conversation history, and handling message input and output efficiently.

7.  **Convenience**: Ensure that the application is easy to use and understand, even for users with limited technical expertise.

The project seeks to provide a chat application that can be used for various purposes, such as personal communication, collaboration in group projects, and staying connected with friends and colleagues. It serves as a versatile and userfriendly tool for textbased communication in both personal and professional contexts.

**System Analysis**

**Existing System**

There are several existing chat systems and applications that serve as valuable benchmarks. Popular platforms like WhatsApp, Facebook Messenger, Snapchat and Instagram are widely used in the real world. These offers seamless personal and group chats, with endtoend encryption for privacy. While these platforms are wellestablished, my project can aim to offer a similar level of utility by focusing on userfriendly interfaces, realtime messaging making it a useful and competitive communication tool in its own right.

The reason why many chat applications continue to enter the market with similar messaging concepts is that realtime communication is a fundamental and everevolving aspect of our interconnected world. These applications aim to address specific user needs, preferences, and niches, even within the messaging space. Project can connect with this concept by offering a unique selling point or value proposition. Whether it's focusing on enhanced security, tailored features for specific user groups, or a simplified and intuitive user experience, r application can stand out by addressing specific communication needs, more interactive User interface and preferences, making it a valuable addition to the diverse landscape of chat applications.

**PROPOSED SYSTEM**

The motivation for working on a project, even if similar applications exist, can stem from several reasons. It's essential to have a clear understanding of r project's unique value proposition and goals. I have designed a unique user interface(UI) based on me which makes it more interactive and more attractive and The application offers an initial login screen for authentication and then allows users to select between personal and group chats.

At the outset, the application provides a secure login screen where users are required to input their username and password. The application verifies this information against predefined values to grant access, after Login process, users can choose between two chat options: "Personal Chat" and "Group Chat", The "Personal Chat" option enables oneonone conversations, while "Group Chat" allows users to specify the number of friends they want to include in a group chat. This choice facilitates both private and group communication. Messages are sent in realtime, displayed instantly in the user's chat window, and archived for future reference.

**Advantages**

The most common and useful advantages of chat applications are:

1. **RealTime Communication**: Chat applications provide instant communication, allowing users to exchange messages in realtime, making it highly efficient for quick conversations.

2**. CostEfficiency**: Most chat applications offer free messaging, which is costeffective compared to traditional SMS or phone calls.

3. Group Chat: Group chat functionality is common, facilitating collaboration among multiple participants in both personal and professional contexts.

4. Global Reach: Chat applications connect people worldwide, making it possible to communicate with individuals regardless of geographical location.

5. Privacy and Security: Many chat apps provide endtoend encryption to protect the privacy and security of messages, ensuring that only the intended recipients can read them.

6. Archiving: Chat apps often archive conversations, making it easy to review past messages and search for specific information or references.

7. Business and Productivity: Chat apps are widely used in the business environment for collaboration, document sharing, and project management, enhancing productivity and teamwork.

8. Customization: Users can customize their profiles, chat settings, and notification preferences to tailor their chat experience.

it provides a platform for realtime communication, facilitating instantaneous exchanges of messages. This feature is invaluable for both personal and professional interactions, enabling users to stay connected with friends, family, colleagues, and clients.

Moreover, chat applications are highly costeffective. Most of these apps offer free messaging, reducing the need for traditional SMS and phone calls, which can be costly. The ability to share multimedia, including images, videos, documents, and voice recordings, enhances the richness of communication, making it an ideal tool for conveying information and ideas. Group chat functionality further enhances the versatility of chat applications, allowing multiple users to collaborate simultaneously. This is especially valuable for businesses and project teams, as it fosters collaboration and efficient document sharing.

Additionally, the emphasis on privacy and security, often through endtoend encryption, ensures that conversations remain private and secure, providing peace of mind for users. Users can also archive their conversations and search for specific information easily, making it a valuable tool for referencing past messages.

Overall, chat applications offer a userfriendly, efficient, and costeffective means of communication, and they have a wide range of applications, from personal conversations to professional collaboration, making them a powerful tool for modern communication needs.

**3. Literature Survey**

A literature survey of chat applications reveals a diverse and evolving landscape, reflecting the evergrowing demand for realtime communication and collaboration. Chat applications have become integral in various domains, from personal communication to business and healthcare. Researchers and developers have explored a wide range of topics within many field.

In summary, the literature surrounding chat applications is rich and diverse, spanning technical, social, and practical aspects. As technology and user expectations continue to evolve, research in this field remains dynamic and relevant, addressing the everchanging landscape of realtime communication.

In today's digital landscape, the competition in the realm of chat applications is fierce. Popular messaging platforms like WhatsApp, Facebook Messenger, and Slack have dominated the market for years. They offer a wide range of features, security, and reliability, making it challenging for new entrants to compete effectively. Nevertheless, new chat applications keep emerging due to the dynamic nature of technology and users' evolving needs.

Existing chat applications serve a multitude of purposes, from personal communication to professional collaboration. Users rely on these platforms to send text messages, make voice and video calls, share multimedia content, and engage in group conversations. They also value the convenience and mobility that chat apps offer, allowing them to stay connected with friends, family, and colleagues from anywhere in the world.

The need for new chat applications arises from several factors. Firstly, users' expectations change over time, leading to the demand for innovative features and enhanced user experiences. This can include improved security measures, integration with other services, or novel communication methods. Secondly, users may have specific privacy concerns or regional preferences, driving the creation of chat apps tailored to those needs. Additionally, niche markets, such as business communication tools or specialized interest groups, require tailored solutions to address their unique requirements.

Moreover, people's thoughts on chat applications have evolved. While convenience and connectivity remain paramount, users now prioritize data security and privacy more than ever. Recent controversies surrounding data privacy have led to increased scrutiny of chat apps' security practices, giving rise to privacycentric alternatives. Users also seek applications that facilitate productive remote work and collaboration, as the global workforce increasingly embraces telecommuting.

In conclusion, the literature survey highlights the dynamic landscape of chat applications, with fierce competition driven by changing user expectations and needs. The emergence of new applications is a testament to the everevolving nature of technology and the continuous pursuit of improved user experiences, security, and privacy. As users' thoughts and needs evolve, chat applications will continue to adapt and innovate to meet the demands of an everchanging digital world.

**SYSTEM REQUIREMENTS**

##### HARDWARE REQUIREMENTS:

* SYSTEM : Pentium IV 2.4 GHz
* HARD DISK : 40 GB
* RAM : 256 MB

##### SOFTWARE REQUIREMENTS:

* Operating system : Windows XP Professional
* Front End : JAVA(swing)

**FEASIBILITY STUDY**

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

* ECONOMICAL FEASIBILITY
* TECHNICAL FEASIBILITY
* SOCIAL FEASIBILITY

**ECONOMICAL FEASIBILITY**

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

* The chat application project is designed to be cost-effective by leveraging freely available technologies wherever possible.
* Customized components will be purchased judiciously to ensure the project remains within the budget.

### **TECHNICAL FEASIBILITY**

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

* The chat application is designed to have modest technical requirements, minimizing the demands on available resources.
* User-friendliness and effective training strategies will be implemented to ensure user acceptance and confidence in the system, promoting constructive feedback from users.

**SOCIAL FEASIBILITY**

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

Ensuring the chat application's success involves assessing its acceptance among users. This entails:

* Implementing effective training methods to ensure users can efficiently use the system.
* Fostering an environment where users do not perceive the system as a threat but rather as a necessary tool.
* Raising users' confidence in the system to the extent that they feel comfortable offering constructive criticism, as they are the ultimate end-users. Their feedback is valuable for continual improvement.

**Definitions of Elements Used in the Project**

**SOFTWARE ENVIRONMENT**

## Java Technology

Java technology is both a programming language and a platform.

### **The Java programming language is a high level language that can be characterized by all of the following buzzwords:**

* + - Simple
    - Architecture neutral
    - Object oriented
    - Portable
    - Distributed
    - High performance
    - Interpreted
    - Multithreaded
    - Robust
    - Dynamic
    - Secure

With most programming languages, either compile or interpret a program so that can run it on r computer. The Java programming language is unusual in that a program is both compiled and interpreted. With the compiler, first translate a program into an intermediate language called Java byte codes —the platform independent codes interpreted by the interpreter on the Java platform. The interpreter parses and runs each Java byte code instruction on the computer. Compilation happens just once; interpretation occurs each time the program is executed. The following figure illustrates how this works.



Every Java interpreter, whether it’s a development tool or a Web browser that can run applets, is an implementation of the Java VM. Java byte codes help make “write once, run anywhere” possible. can compile r program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the Java VM. That means that as long as a computer has a Java VM, the same program written in the Java programming language can run on Windows 2000, a Solaris workstation, or on an iMac.



### The Java Platform

A platform is the hardware or software environment in which a program runs. We’ve already mentioned some of the most popular platforms like Windows 2000, Linux, Solaris, and MacOS. Most platforms can be described as a combination of the operating system and hardware. The Java platform differs from most other platforms in that it’s a softwareonly platform that runs on top of other hardwarebased platforms.

The Java platform has two components:

* The Java Virtual Machine (Java VM)
* The Java Application Programming Interface (Java API)

’ve already been introduced to the Java VM. It’s the base for the Java platform and is ported onto various hardwarebased platforms.

The Java API is a large collection of readymade software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries of related classes and interfaces; these libraries are known as packages. The next section, What Can Java Technology Do? Highlights what functionality some of the packages in the Java API provide.

The following figure depicts a program that’s running on the Java platform. As the figure shows, the Java API and the virtual machine insulate the program from the hardware.



Native code is code that after compile it, the compiled code runs on a specific hardware platform. As a platformindependent environment, the Java platform can be a bit slower than native code. However, smart compilers, welltuned interpreters, and justintime byte code compilers can bring performance close to that of native code without threatening portability.

## What Can Java Technology Do?

The most common types of programs written in the Java programming language are applets and applications. If ’ve surfed the Web, ’re probably already familiar with applets. An applet is a program that adheres to certain conventions that allow it to run within a Javaenabled browser.

However, the Java programming language is not just for writing cute, entertaining applets for the Web. The generalpurpose, highlevel Java programming language is also a powerful software platform. Using the generous API, can write many types of programs.

An application is a standalone program that runs directly on the Java platform. A special kind of application known as a server serves and supports clients on a network. Examples of servers are Web servers, proxy servers, mail servers, and print servers. Another specialized program is a servlet. A servlet can almost be thought of as an applet that runs on the server side. Java Servlets are a popular choice for building interactive web applications, replacing the use of CGI scripts. Servlets are similar to applets in that they are runtime extensions of applications. Instead of working in browsers, though, servlets run within Java Web servers, configuring or tailoring the server.

How does the API support all these kinds of programs? It does so with packages of software components that provides a wide range of functionality. Every full implementation of the Java platform gives the following features:

* **The essentials**: Objects, strings, threads, numbers, input and output, data structures, system properties, date and time, and so on.
* **Applets**: The set of conventions used by applets.
* **Networking**: URLs, TCP (Transmission Control Protocol), UDP (User Data gram Protocol) sockets, and IP (Internet Protocol) addresses.
* **Internationalization**: Help for writing programs that can be localized for users worldwide. Programs can automatically adapt to specific locales and be displayed in the appropriate language.
* **Security**: Both low level and high level, including electronic signatures, public and private key management, access control, and certificates.
* **Software components**: Known as JavaBeansTM, can plug into existing component architectures.
* **Object serialization**: Allows lightweight persistence and communication via Remote Method Invocation (RMI).
* **Java Database Connectivity (JDBCTM)**: Provides uniform access to a wide range of relational databases.

The Java platform also has APIs for 2D and 3D graphics, accessibility, servers, collaboration, telephony, speech, animation, and more. The following figure depicts what is included in the Java 2 SDK.



## How Will Java Technology Change My Life?

We can’t promise fame, fortune, or even a job if learn the Java programming language. Still, it is likely to make r programs better and requires less effort than other languages. We believe that Java technology will help do the following:

* **Get started quickly**: Although the Java programming language is a powerful object oriented language, it’s easy to learn, especially for programmers already familiar with C or C++.
* **Write less code**: Comparisons of program metrics (class counts, method counts, and so on) suggest that a program written in the Java programming language can be four times smaller than the same program in C++.
* **Write better code**: The Java programming language encourages good coding practices, and its garbage collection helps avoid memory leaks. Its object orientation, its JavaBeans component architecture, and its wideranging, easily extendible API let reuse other people’s tested code and introduce fewer bugs.
* **Develop programs more quickly**: r development time may be as much as twice as fast versus writing the same program in C++. Why? write fewer lines of code and it is a simpler programming language than C++.
* **Avoid platform dependencies with 100% Pure Java**: can keep r program portable by avoiding the use of libraries written in other languages. The 100% Pure JavaTM Product Certification Program has a repository of historical process manuals, white papers, brochures, and similar materials online.
* **Write once, run anywhere**: Because 100% Pure Java programs are compiled into machine independent byte codes, they run consistently on any Java platform.
* **Distribute software more easily**: can upgrade applets easily from a central server. Applets take advantage of the feature of allowing new classes to be loaded “on the fly,” without recompiling the entire program.

### **ODBC**

Microsoft Open Database Connectivity (ODBC) is a standard programming interface for application developers and database systems providers. Before ODBC became a de facto standard for Windows programs to interface with database systems, programmers had to use proprietary languages for each database they wanted to connect to. Now, ODBC has made the choice of the database system almost irrelevant from a coding perspective, which is as it should be. Application developers have much more important things to worry about than the syntax that is needed to port their program from one database to another when business needs suddenly change.

Through the ODBC Administrator in Control Panel, can specify the particular database that is associated with a data source that an ODBC application program is written to use. Think of an ODBC data source as a door with a name on it. Each door will lead to a particular database. For example, the data source named Sales Figures might be a SQL Server database, whereas the Accounts Payable data source could refer to an Access database. The physical database referred to by a data source can reside anywhere on the LAN.

The ODBC system files are not installed on r system by Windows 95. Rather, they are installed when setup a separate database application, such as SQL Server Client or Visual Basic 4.0. When the ODBC icon is installed in Control Panel, it uses a file called ODBCINST.DLL. It is also possible to administer r ODBC data sources through a standalone program called ODBCADM.EXE. There is a 16bit and a 32bit version of this program and each maintains a separate list of ODBC data sources.

From a programming perspective, the beauty of ODBC is that the application can be written to use the same set of function calls to interface with any data source, regardless of the database vendor. The source code of the application doesn’t change whether it talks to Oracle or SQL Server. We only mention these two as an example. There are ODBC drivers available for several dozen popular database systems. Even Excel spreadsheets and plain text files can be turned into data sources. The operating system uses the Registry information written by ODBC Administrator to determine which lowlevel ODBC drivers are needed to talk to the data source (such as the interface to Oracle or SQL Server). The loading of the ODBC drivers is transparent to the ODBC application program. In a client/server environment, the ODBC API even handles many of the network issues for the application programmer.

The advantages of this scheme are so numerous that are probably thinking there must be some catch. The only disadvantage of ODBC is that it isn’t as efficient as talking directly to the native database interface. ODBC has had many detractors make the charge that it is too slow. Microsoft has always claimed that the critical factor in performance is the quality of the driver software that is used. In our humble opinion, this is true. The availability of good ODBC drivers has improved a great deal recently. And anyway, the criticism about performance is somewhat analogous to those who said that compilers would never match the speed of pure assembly language. Maybe not, but the compiler (or ODBC) gives the opportunity to write cleaner programs, which means finish sooner. Meanwhile, computers get faster every year.

**JDBC**

In an effort to set an independent database standard API for Java; Sun Microsystems developed Java Database Connectivity, or JDBC. JDBC offers a generic SQL database access mechanism that provides a consistent interface to a variety of RDBMSs. This consistent interface is achieved through the use of “plugin” database connectivity modules, or drivers. If a database vendor wishes to have JDBC support, he or she must provide the driver for each platform that the database and Java run on.

To gain a wider acceptance of JDBC, Sun based JDBC’s framework on ODBC. As discovered earlier in this chapter, ODBC has widespread support on a variety of platforms. Basing JDBC on ODBC will allow vendors to bring JDBC drivers to market much faster than developing a completely new connectivity solution.

JDBC was announced in March of 1996. It was released for a 90 day public review that ended June 8, 1996. Because of user input, the final JDBC v1.0 specification was released soon after.

The remainder of this section will cover enough information about JDBC for to know what it is about and how to use it effectively. This is by no means a complete overview of JDBC. That would fill an entire book.

### **JDBC Goals**

Few software packages are designed without goals in mind. JDBC is one that, because of its many goals, drove the development of the API. These goals, in conjunction with early reviewer feedback, have finalized the JDBC class library into a solid framework for building database applications in Java.

The goals that were set for JDBC are important. They will give some insight as to why certain classes and functionalities behave the way they do. The eight design goals for JDBC are as follows:

1. **SQL Level API**

The designers felt that their main goal was to define a SQL interface for Java. Although not the lowest database interface level possible, it is at a low enough level for higherlevel tools and APIs to be created. Conversely, it is at a high enough level for application programmers to use it confidently. Attaining this goal allows for future tool vendors to “generate” JDBC code and to hide many of JDBC’s complexities from the end user.

1. **SQL Conformance**

SQL syntax varies as move from database vendor to database vendor. In an effort to support a wide variety of vendors, JDBC will allow any query statement to be passed through it to the underlying database driver. This allows the connectivity module to handle nonstandard functionality in a manner that is suitable for its users.

1. **JDBC must be implemental on top of common database interfaces**   
    The JDBC SQL API must “sit” on top of other common SQL level APIs. This goal allows JDBC to use existing ODBC level drivers by the use of a software interface. This interface would translate JDBC calls to ODBC and vice versa.
2. **Provide a Java interface that is consistent with the rest of the Java system**

Because of Java’s acceptance in the user community thus far, the designers feel that they should not stray from the current design of the core Java system.

1. **Keep it simple**

This goal probably appears in all software design goal listings. JDBC is no exception. Sun felt that the design of JDBC should be very simple, allowing for only one method of completing a task per mechanism. Allowing duplicate functionality only serves to confuse the users of the API.

1. **Use strong, static typing wherever possible**

Strong typing allows for more error checking to be done at compile time; also, less error appear at runtime.

1. **Keep the common cases simple**

Because more often than not, the usual SQL calls used by the programmer are simple SELECT’s, INSERT’s, DELETE’s and UPDATE’s, these queries should be simple to perform with JDBC. However, more complex SQL statements should also be possible.

Finally we decided to proceed the implementation using Java Networking.

And for dynamically updating the cache table we go for MS Access database.

Java ha two things: a programming language and a platform.

Java is a highlevel programming language that is all of the following

Simple Architectureneutral

Objectoriente d Portable

Distributed Highperformance

Interpreted multithreaded

Robust Dynamic

Secure

Java is also unusual in that each Java program is both compiled and interpreted. With a compile translate a Java program into an intermediate language called Java byte codes the platform independent code instruction is passed and run on the computer.

Compilation happens just once; interpretation occurs each time the program is executed. The figure illustrates how this works.

**Java Program**

**Compilers**

**Interpreter**

**My Program**

can think of Java byte codes as the machine code instructions for the Java Virtual Machine (Java VM). Every Java interpreter, whether it’s a Java development tool or a Web browser that can run Java applets, is an implementation of the Java VM. The Java VM can also be implemented in hardware.

**Java AWT**

(Abstract Window Toolkit) is an API to develop Graphical User Interface (GUI) or windows-based applications in Java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

The java.awt [package](https://www.javatpoint.com/package) provides [classes](https://www.javatpoint.com/object-and-class-in-java) for AWT API such as [TextField](https://www.javatpoint.com/java-awt-textfield), [Label](https://www.javatpoint.com/java-awt-label), [TextArea](https://www.javatpoint.com/java-awt-textarea), RadioButton, [CheckBox](https://www.javatpoint.com/java-awt-checkbox), [Choice](https://www.javatpoint.com/java-awt-choice), [List](https://www.javatpoint.com/java-awt-list) etc.

The AWT tutorial will help the user to understand Java GUI programming in simple and easy steps.

## Why AWT is platform independent?

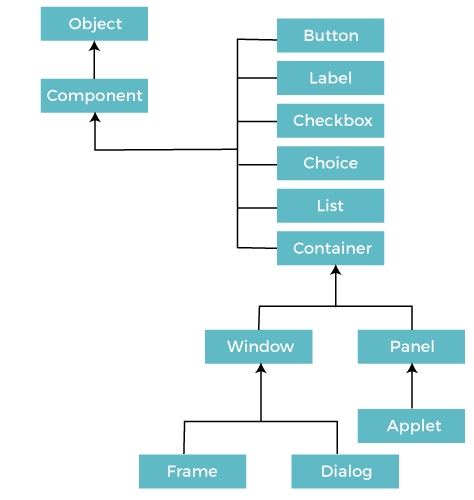
Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.

For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.

In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

## Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



### **Components**

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

### **Container**

The Container is a component in AWT that can contain another components like [buttons](https://www.javatpoint.com/java-awt-button), textfields, labels etc. The classes that extends Container class are known as container such as **Frame, Dialog** and **Panel**.

It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

#### **Note: A container itself is a component (see the above diagram), therefore we can add a container inside container.**

**Types of containers:**

There are four types of containers in Java AWT:

1. Window
2. Panel
3. Frame
4. Dialog

### **Window**

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

### **Panel**

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

### **Frame**

The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

## Useful Methods of Component Class

|  |  |
| --- | --- |
| **Method** | **Description** |
| public void add(Component c) | Inserts a component on this component. |
| public void setSize(int width,int height) | Sets the size (width and height) of the component. |
| public void setLayout(LayoutManager m) | Defines the layout manager for the component. |
| public void setVisible(boolean status) | Changes the visibility of the component, by default false. |

## Java AWT Example

To create simple AWT example, you need a frame. There are two ways to create a GUI using Frame in AWT.

1. By extending Frame class (**inheritance**)
2. By creating the object of Frame class (**association**)

# Java Swing

**Java Swing** is a part of Java Foundation Classes (JFC) that is used to create windowbased applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platformindependent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

### **Hierarchy of Java Swing classes**

The hierarchy of java swing API is given below.



### Components of Java Swing

Some of the important and common components of the Java Swing class are:

1. **JFrame:** JFrame is a top-level container that represents the main window of a GUI application. It provides a title bar, and minimizes, maximizes, and closes buttons.
2. **JPanel:** JPanel is a container that can hold other components. It is commonly used to group related components together.
3. **JButton:** JButton is a component that represents a clickable button. It is commonly used to trigger actions in a GUI application.
4. **JLabel:** JLabel is a component that displays text or an image. It is commonly used to provide information or to label other components.
5. **JTextField:** JTextField is a component that allows the user to input text. It is commonly used to get input from the user, such as a name or an address.
6. **JCheckBox:** JCheckBox is a component that represents a checkbox. It is commonly used to get a binary input from the user, such as whether or not to enable a feature.
7. **JList:** JList is a component that represents a list of elements. It is typically used to display a list of options from which the user can select one or more items.
8. **JTable:** JTable is a component that represents a data table. It is typically used to present data in a tabular fashion, such as a list of products or a list of orders.
9. **JScrollPane:** JScrollPane is a component that provides scrolling functionality to other components. It is commonly used to add scrolling to a panel or a table.

Java Swing is a powerful GUI toolkit that provides a rich set of components for creating desktop applications. Java Swing is a popular choice for desktop applications, and it continues to be widely used by developers worldwide. Ultimately, the choice of whether to use Swing or another GUI toolkit depends on the specific needs of the application and the preferences of the developer.

**How I Used Swing:**

In My chat application code, **Java Swing** is used to create the graphical user interface (GUI) for the application. Swing is a set of libraries in Java for building platform independent, rich, and interactive graphical applications. It provides a wide range of components for creating windows, dialogs, buttons, text fields, and other GUI elements. Here's a more detailed explanation of how Swing is used in this chat application:

1. **Swing Components**:

* **JFrame**: A `JFrame` is the top-level window for r application. It contains all other Swing components and is used as the main application frame.
  + used it to create the main application window in r chat application.
* **JPanel**: A JPanel is a lightweight container that can hold other Swing components. It's often used to group components and manage their lat.
* - used it for creating various panels within r frames to organize components and control their placement.
* **JLabel**: JLabel is used for displaying text or simple images. used it to display labels like "Login," "Username," and "Password.".
* **JTextField**: JTextField provides a single-line input field for user text input. used it for the username and password input fields in the login panel
* **JPasswordField**: is a secure text input field designed for password input. used it for the password field in the login panel.
* **JButton**: is used for creating clickable buttons. used it for the "Login," "Personal Chat," and "Group Chat" buttons.
* **JTextArea**: JTextArea is a multi-line text area that allows for the display of text content. used it to show chat messages in the chat windows.
* **JScrollPane**: is used to add scrollbars to components like JTextArea. This is useful when the content doesn't fit within the visible area.
* **ImageIcon** is used to load and display images. used it for displaying user profile pictures and the application logo.
* **Color:** is used to specify colors for various components. used it to set background and foreground colors to customize the appearance of r application.

**2. Lat Managers:**

* **GridBagLat**: is a flexible and powerful lat manager that allows precise control over the placement and sizing of components within a panel.
* used it in the login panel to define the lat of components like labels and text fields, specifying their positions and dimensions.
* **FlowLat**: arranges components in a row, respecting their preferred sizes. It's a simple lat manager used for arranging buttons horizontally.

**3. Event Handling:**

Action listeners are added to buttons to perform actions when buttons are clicked. For example, the "Login" button's action listener is used to handle authentication, and the "Send" button's action listener is used to send messages.

**4. Image Handling:**

Images for user profiles and the logo are loaded using the getClass().getResource() method. This method retrieves resources from the application's classpath.

**5. User Interface Customization:**

The code sets various properties for Swing components, such as fonts, colors, and dimensions, to customize the look and feel of the application.

**6. Swing Threading:**

The application follows the Swing threading model by invoking the Swingrelated code on the event dispatch thread (EDT). This is achieved by using SwingUtilities.invokeLater() in the main() method.

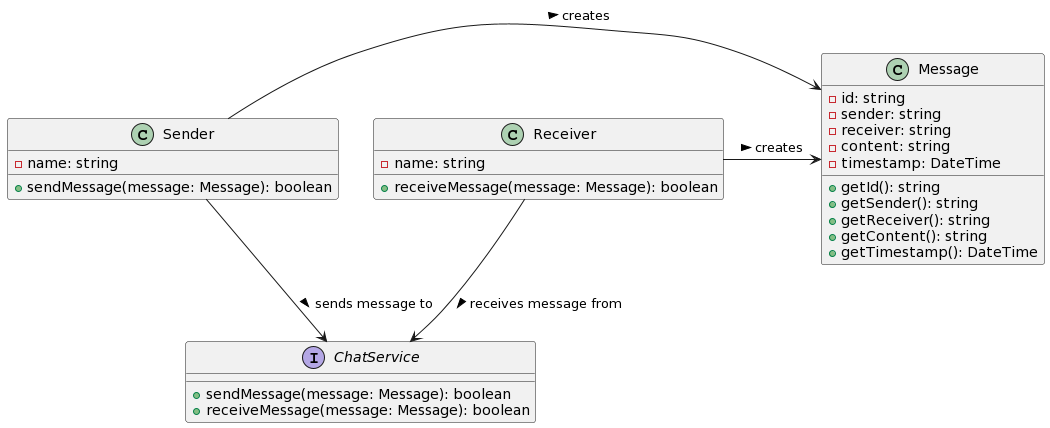
**How i Used Swing Components and Lat Managers and Their Usefulness:**

* **Login Panel:**  used GridBagLat to create a well-structured login panel. The lat manager allowed to align labels and input fields neatly. This is useful for creating an organized and aesthetically pleasing login interface.
* **Buttons:** JButton components were used for "Login," "Personal Chat," and "Group Chat." These buttons provide a way to trigger specific actions. added action listeners to handle login and navigation to chat options. This allows users to interact with r application.
* **Text Areas:** JTextArea components were used to display chat messages. Their ability to display multi-line text is essential for showing conversation history.
* **Scroll Panes:** used JScrollPane to wrap JTextArea components. This allows users to scroll through chat messages when the content exceeds the visible area.
* **User Profile Images:** The ImageIcon component was used to load and display user profile images. This enhances the visual appeal of r application and helps users identify chat participants.
* **Color Customization:** used the Color class to customize the appearance of various components. This makes r application visually appealing and aligned with r chosen color scheme.
* **Lat Control:** Lat managers, such as GridBagLat and FlowLat, helped control the arrangement of components in a responsive and structured manner. This ensures that r UI is well-organized and adapts to different screen sizes and resolutions.

In summary, the chat application uses Swing components to create a userfriendly graphical interface, including login screens, chat windows, and message input fields. It leverages lat managers for component organization, event handling for user interactions, and image handling to display images. The customization of fonts, colors, and dimensions helps improve the user experience. Additionally, it ensures that Swing code runs on the EDT to maintain responsiveness and avoid GUIrelated issues.

**Design**

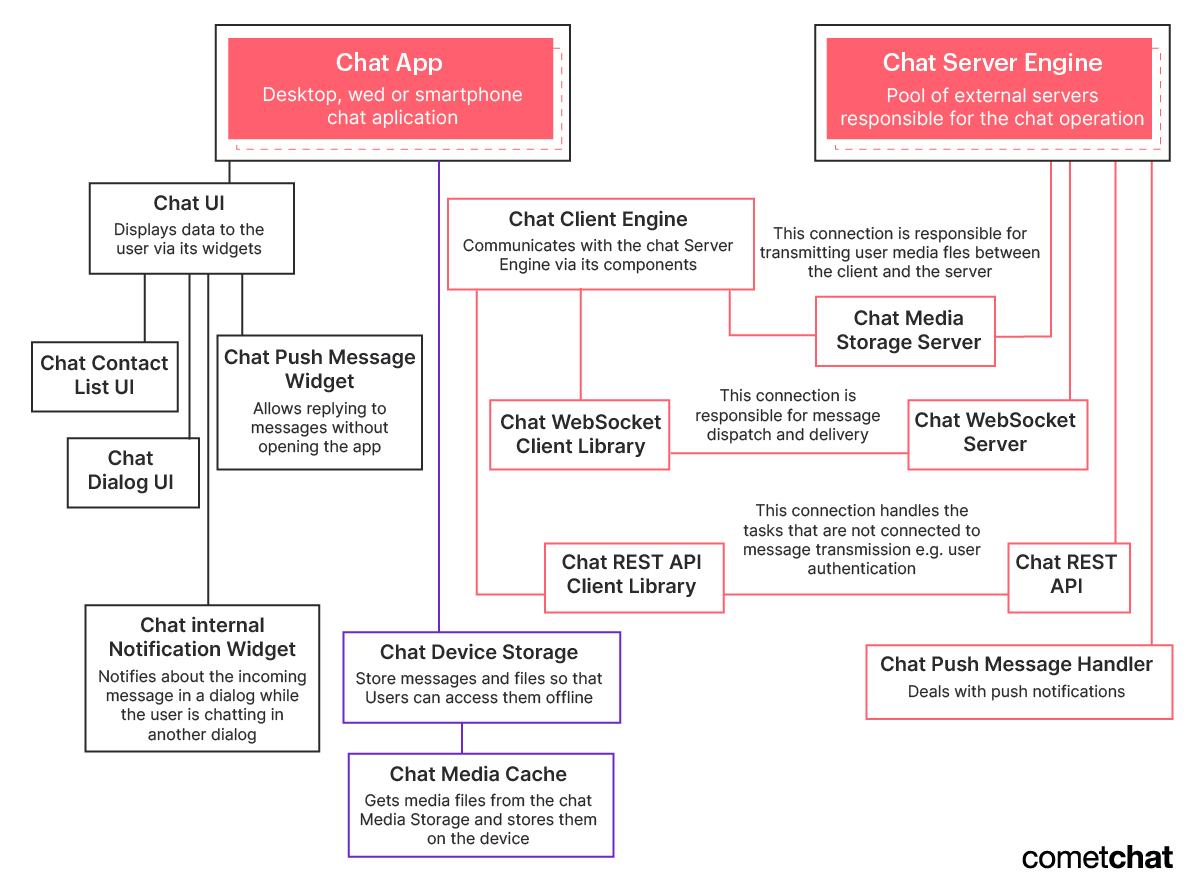
**Chat Application Architecture/workflow Diagram**



In this diagram, I have four entities: ChatService, Message, Sender, and Receiver. The ChatService is responsible for sending and receiving messages, and it provides two methods for doing so (sendMessage and receiveMessage).Message represents a message object with properties such as id, sender, receiver, content, and timestamp. The Sender and Receiver classes represent the clients who use the chat service to send and receive messages, respectively.

The diagram shows that a Sender creates a Message object and sends it to the ChatService by calling the sendMessage method. The ChatService then handles the message and sends it to the intended Receiver, who receives the message by calling the receiveMessage method.

Overall, this diagram shows the basic relationship between the Sender, Receiver, and ChatService in a chat system.



Sequence Diagram for Chat Application

**Features and Functionality of a Chat Application**

As mentioned above, the implementation and functionality of those components will vary depending on the app’s intended use. So, let’s cover the core features of a chat app before diving into the deep end of functionality.

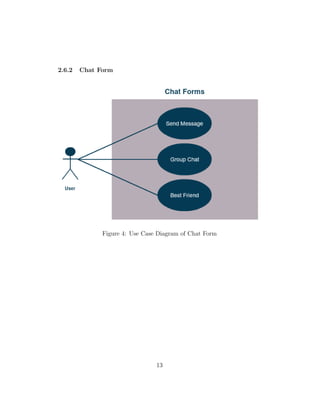
**User Registration and Authentication:** If you want users, you have to give people a way to use your chat app. Pretty straightforward. The most common way to register and authenticate users is with their email or phone number.

**Individual and group messaging:** The core feature of a chat app is the ability to send and receive messages. Both individual (peer to peer) messaging and group messaging are essential aspects of a good chat app.

**Push notifications:** You don’t want your users to have to open your app every few minutes to check for new messages. That would create a terrible user experience, especially on mobile. Push notifications to notify users of a new incoming message or reaction have become a must-have feature.

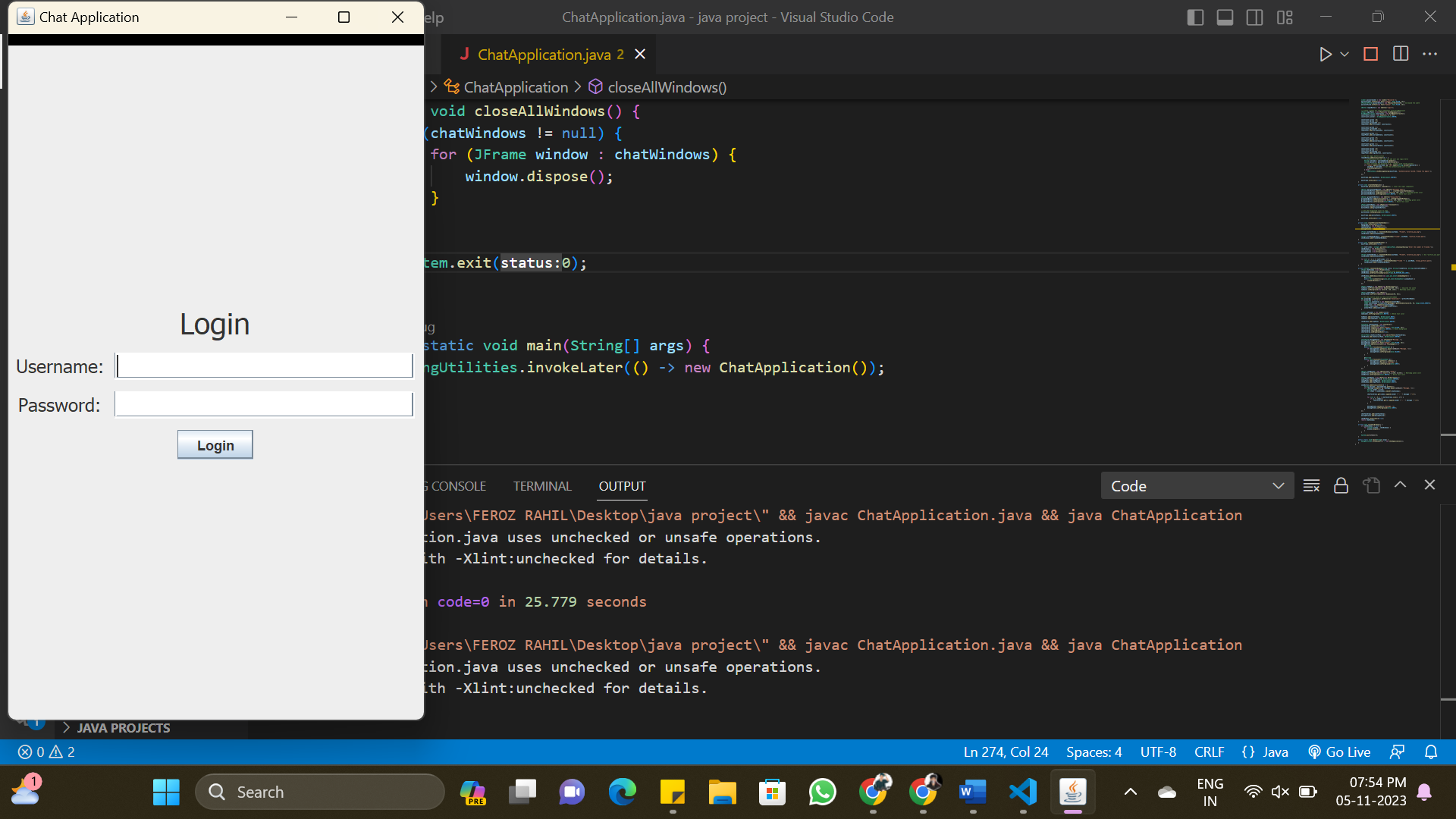
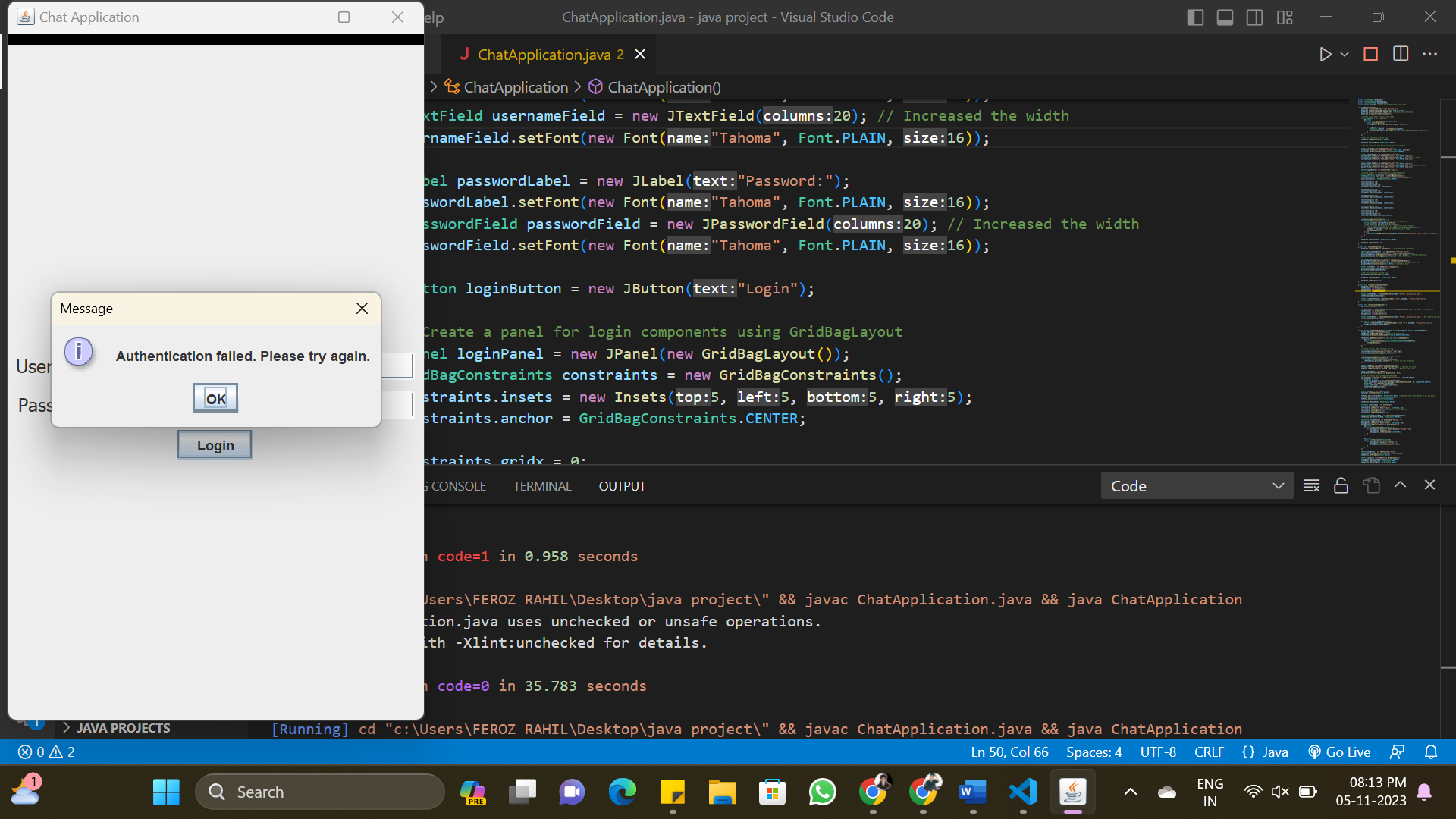
**Sharing of media files:** Gone are the days of plain text messages. (Remember T9? My, how far we’ve come.) Gifs, emojis, audio, images and video files are the new standard. So your app will need local media storage along with cloud storage for fast retrieval.

**User settings & profile customization:** Customization lets your users express their individuality and creativity. Whether it’s the ability to change their font, background color, nickname or something else—it’s important to allow users some flexibility to personalize their experience.

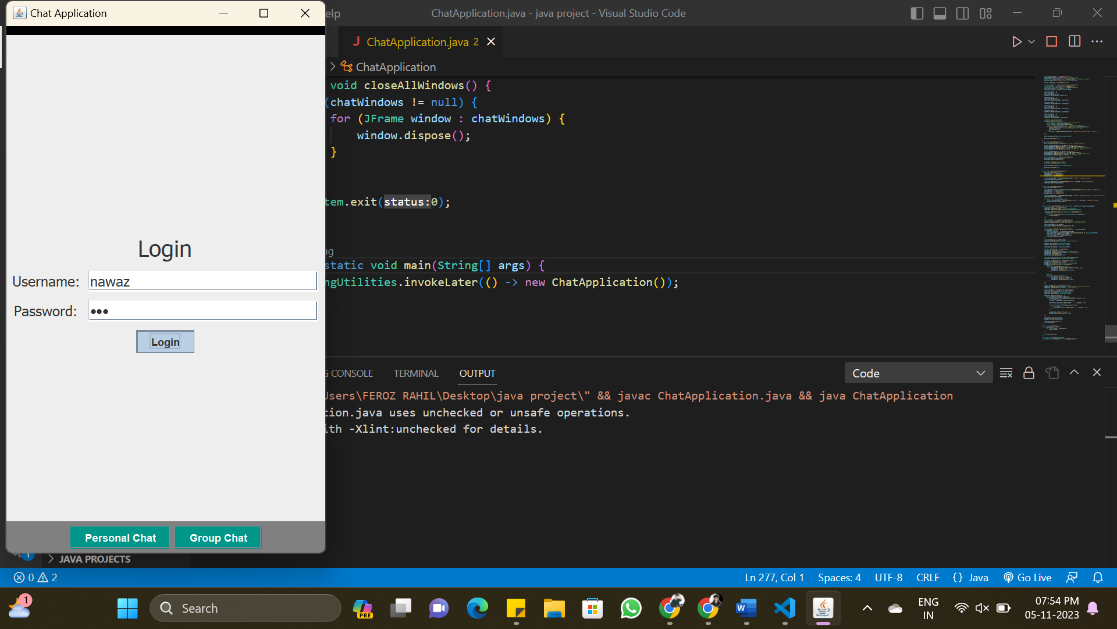


**Screens**

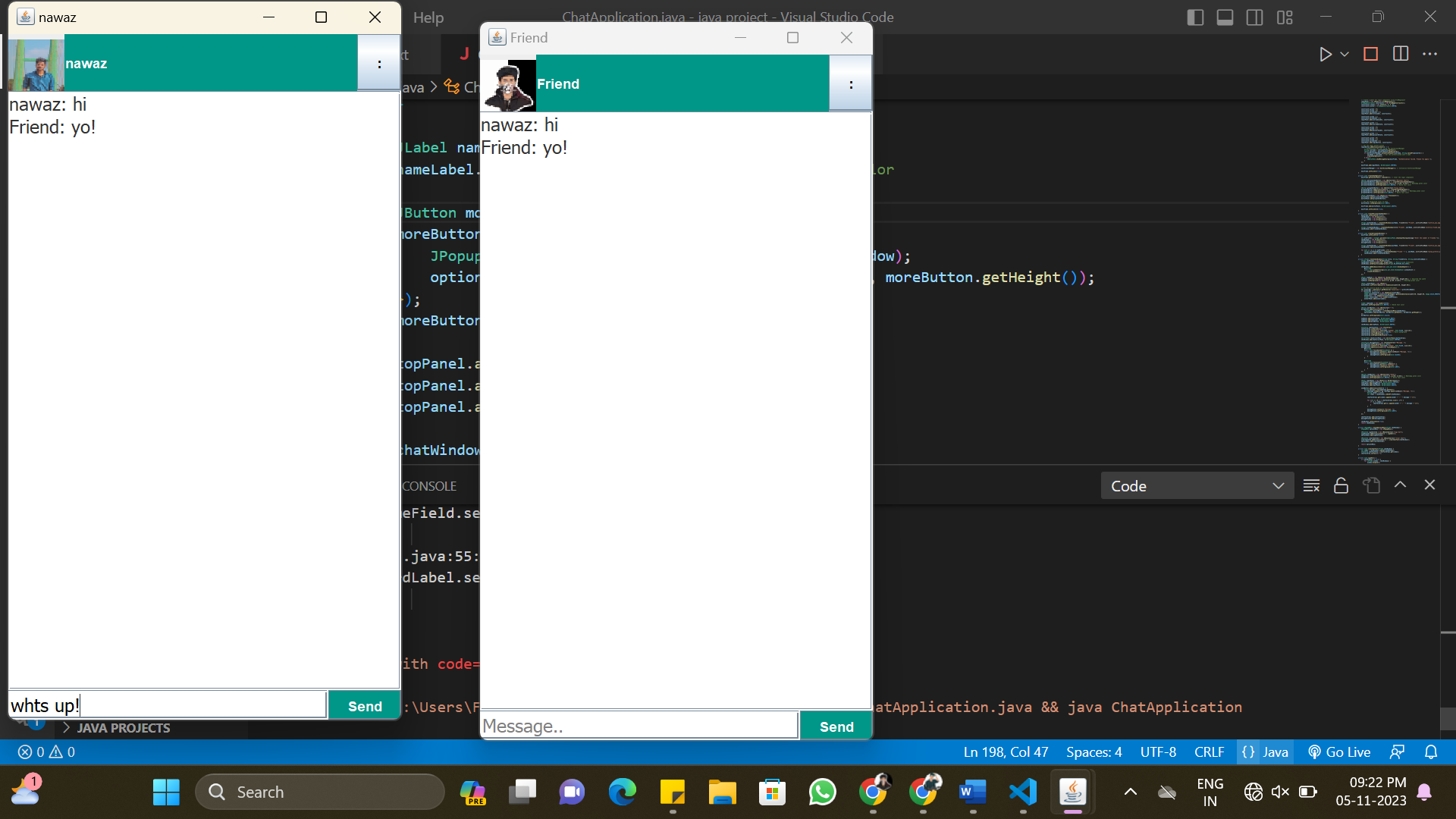
**Login window**

****

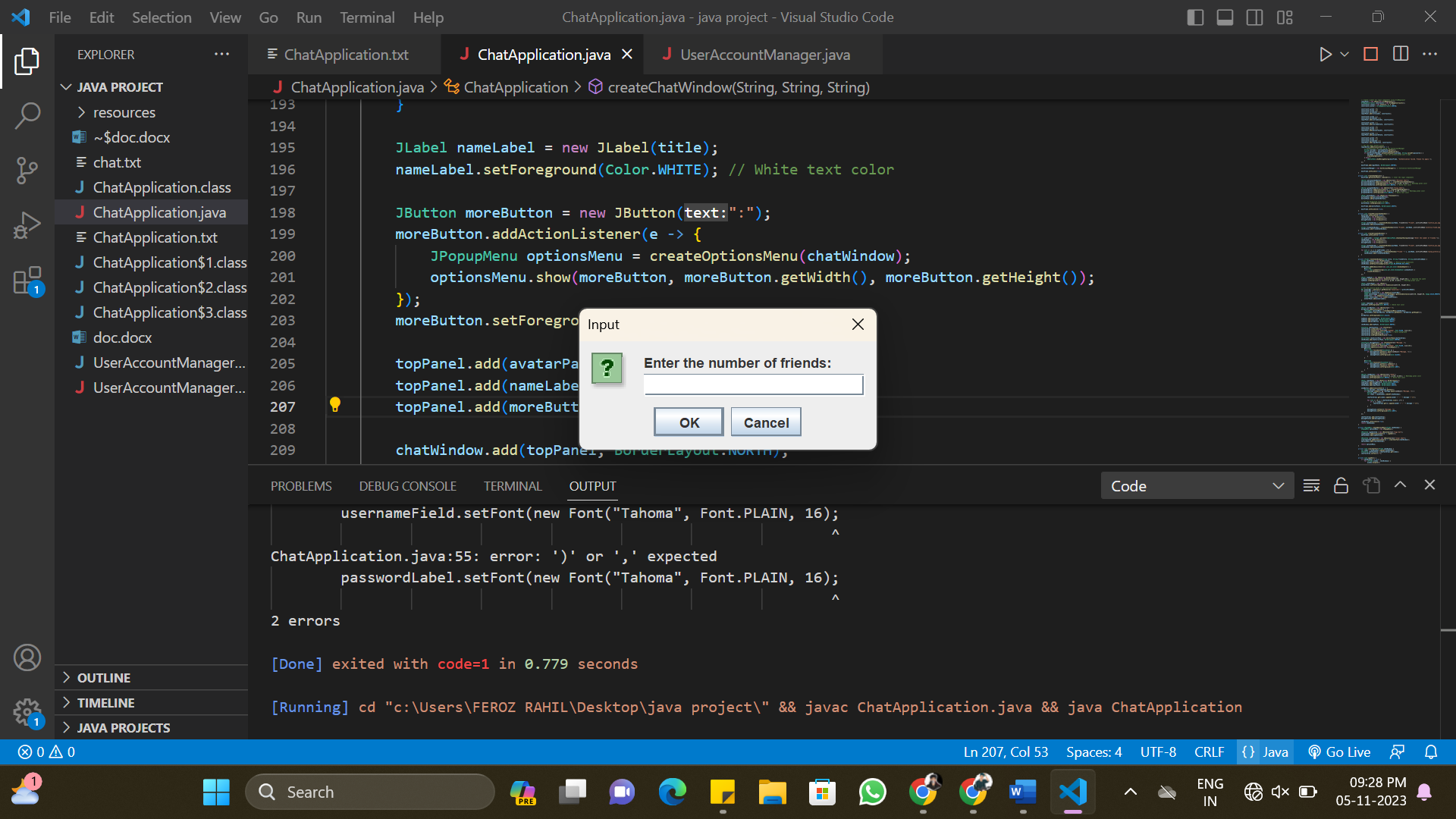
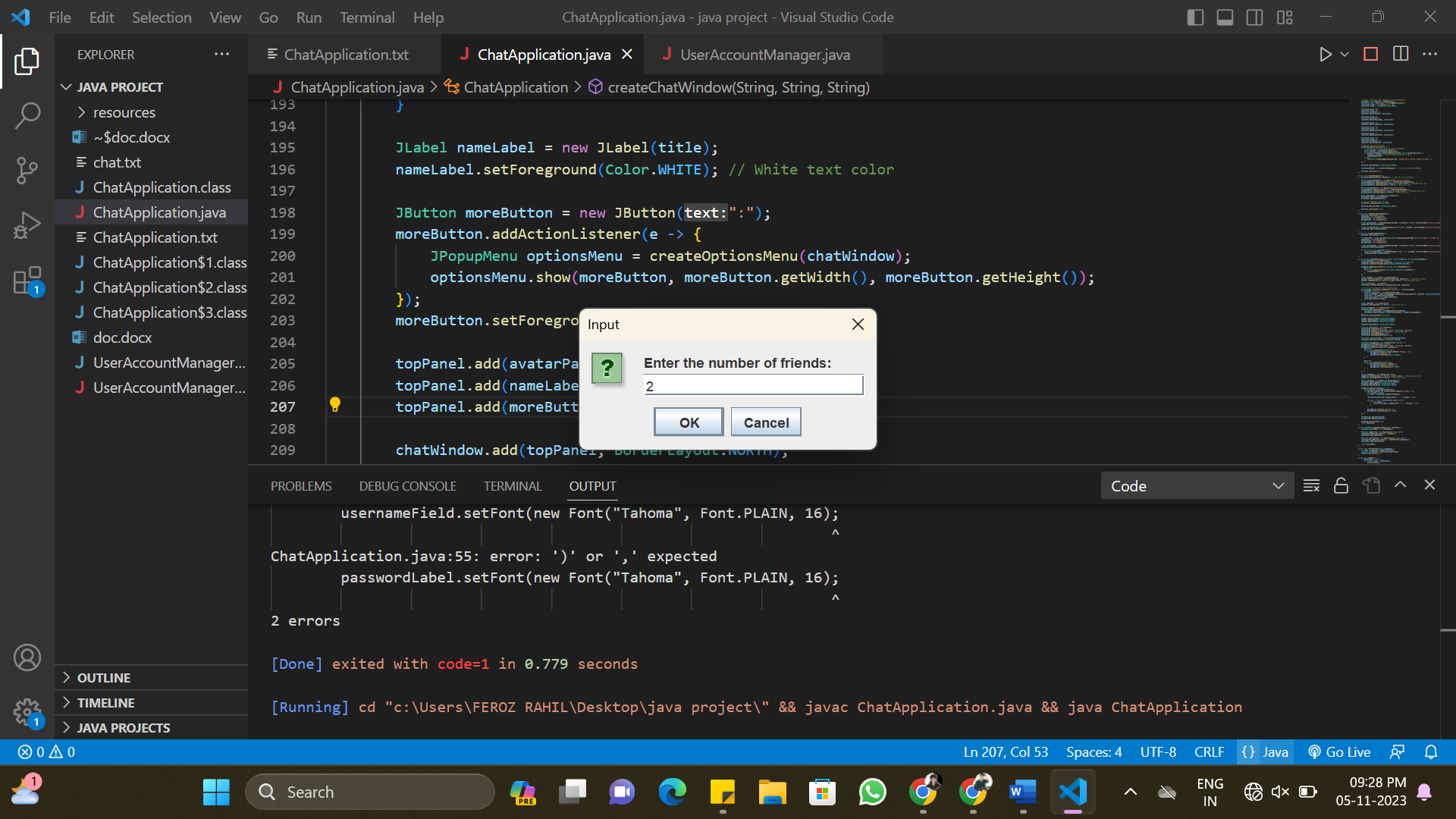
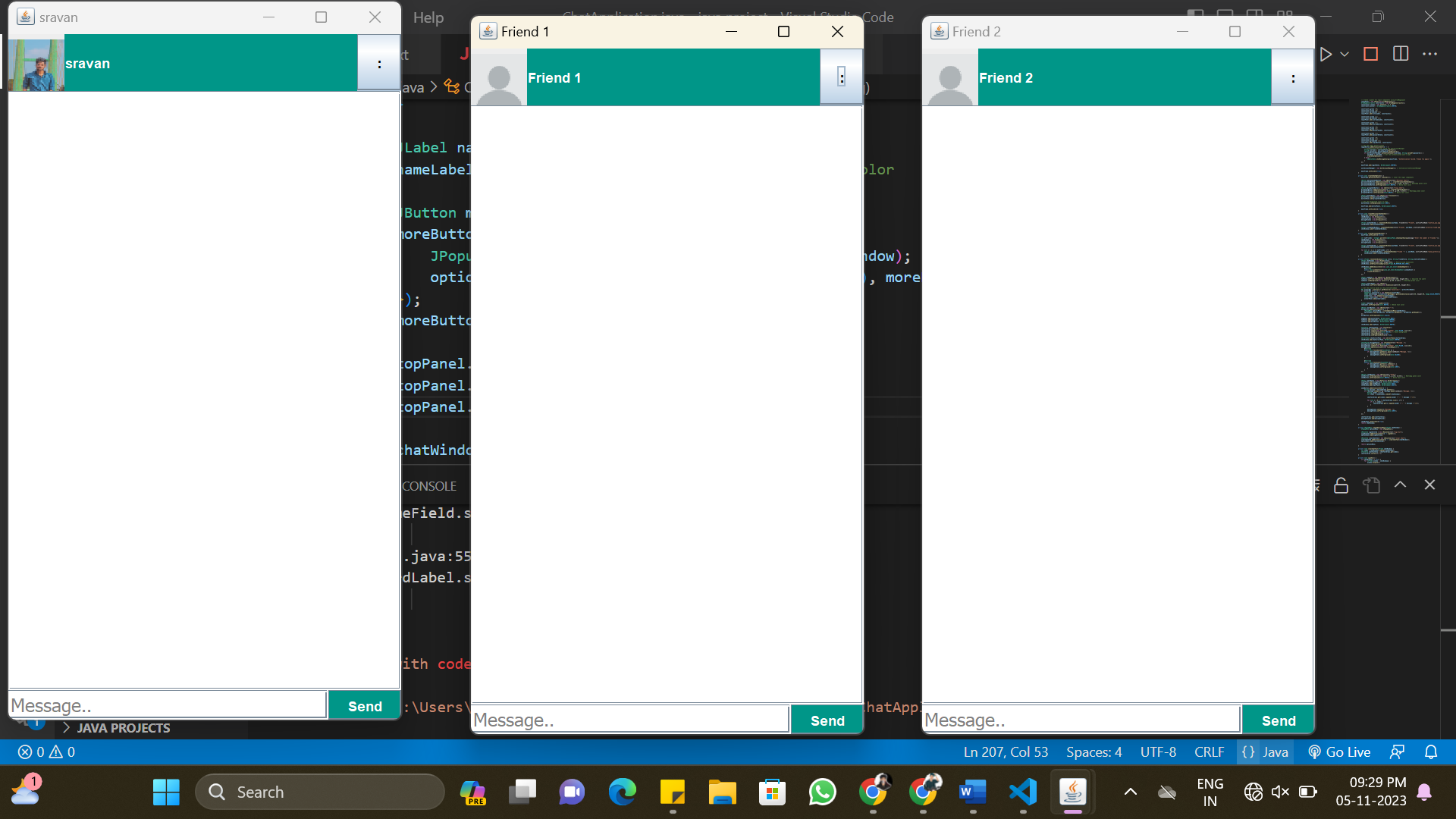
**Select Personal/Group Chat**

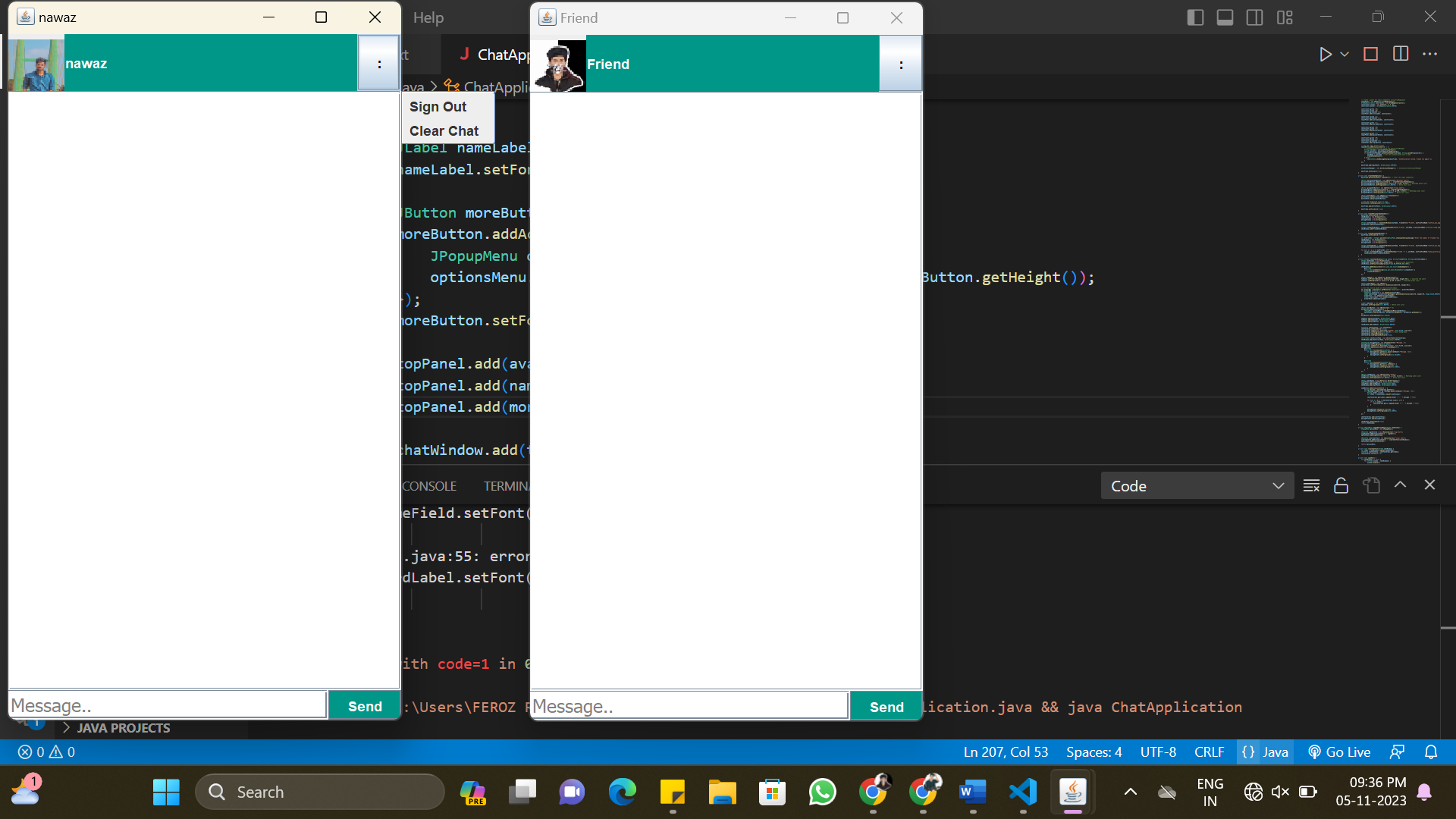
****After Authentication will get access to

choose the option you want to do

* Personal chat
* Group chat
* **Personal Chat**
* **Group Chat (Broadcast):**

**Select the number of friends you want to chat with:**

****



**More Option:**

* sign out
* clear chat

**IMPLEMENTATION**

**Code :**

**UserAccountManager.java**

import java.util.HashMap;

import java.util.Map;

public class UserAccountManager {

    private Map<String, String> userAccounts;

    public UserAccountManager() {

        userAccounts = new HashMap<>();

        // You can load existing user accounts from a file or a database here.

        // For demonstration purposes, we'll add a default user.

        userAccounts.put("nawaz", "786");

        userAccounts.put("sravan", "java");

userAccounts.put("", "");

userAccounts.put("", "");

userAccounts.put("", "");

userAccounts.put("", "");

    }

    public boolean authenticateUser(String username, String password) {

        // Check if the provided username and password match an existing user account.

        String storedPassword = userAccounts.get(username);

        return storedPassword != null && storedPassword.equals(password);

    }

    public boolean addUser(String username, String password) {

        // Add a new user to the userAccounts map.

        if (!userAccounts.containsKey(username)) {

            userAccounts.put(username, password);

            return true;

        }

        return false; // User with the same username already exists.

    }

    public boolean updateUserPassword(String username, String newPassword) {

        // Update the password for an existing user.

        if (userAccounts.containsKey(username)) {

            userAccounts.put(username, newPassword);

            return true;

        }

        return false; // User not found.

    }

    public boolean deleteUser(String username) {

        // Delete a user account.

        if (userAccounts.containsKey(username)) {

            userAccounts.remove(username);

            return true;

        }

        return false; // User not found.

    }

}

This code represents a simple `UserAccountManager` class that is responsible for managing user accounts in a chat application. Here's an explanation of the code:

1. `UserAccountManager` class: This class encapsulates the logic for managing user accounts. It contains methods to authenticate users, add new users, update user passwords, and delete user accounts.

2. `userAccounts` map: This private field is a `Map` that associates usernames with passwords. It is used to store user account information.

3. `UserAccountManager` constructor: The constructor initializes the `userAccounts` map. In this example, it adds two default user accounts for demonstration purposes. In a real application, you would typically load user account information from a file or a database.

4. `authenticateUser` method: This method takes a username and a password as parameters and checks if the provided username and password match an existing user account. It retrieves the stored password for the given username and compares it to the provided password. If the username and password match, the method returns `true`, indicating successful authentication. If the username is not found or the passwords do not match, the method returns `false`.

5. `addUser` method: This method allows adding a new user to the `userAccounts` map. It takes a username and a password as parameters and checks if the username already exists in the map. If the username is not found, it adds the new user to the map and returns `true`. If the username already exists, it returns `false` to indicate that the user with the same username already exists.

6. `updateUserPassword` method: This method is used to update the password for an existing user. It takes a username and a new password as parameters. If the provided username exists in the `userAccounts` map, it updates the password for that user and returns `true`. If the username is not found, it returns `false` to indicate that the user was not found.

7. `deleteUser` method: This method allows deleting a user account. It takes a username as a parameter and checks if the username exists in the `userAccounts` map. If the username is found, it removes the user from the map and returns `true`. If the username is not found, it returns `false` to indicate that the user was not found.

This code provides basic user account management functionalities, including user authentication, adding new users, updating passwords, and deleting user accounts. In a real-world scenario, you would typically integrate this class with a database or a more secure storage mechanism to manage user accounts.

**ChatApplication.java**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.;

import java.util.ArrayList;

import java.util.List;

import java.net.URL;

import javax.swing.ImageIcon;

public class ChatApplication {

    private JFrame mainFrame;

    private JPanel loginPanel;

    private List<JFrame> chatWindows;

    private List<JTextArea> chatTextAreas;

    private List<JTextField> messageFields;

    private String yourName; // Store the authenticated user's name

    private UserAccountManager userAccountManager;

    public ChatApplication() {

        mainFrame = new JFrame("Chat Application");

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

        mainFrame.setSize(380, 640); // Mobile-sized dimensions

        mainFrame.setLayout(new BorderLayout());

        // Create a logo panel with your logo image

        JPanel logoPanel = new JPanel() {

            @Override

            protected void paintComponent(Graphics g) {

                super.paintComponent(g);

                URL logoURL = getClass().getResource("logo.png");

                if (logoURL != null) {

                    ImageIcon logoIcon = new ImageIcon(logoURL);

                    g.drawImage(logoIcon.getImage(), 500, 100, 500, 300, this);

                }

            }

        };

        // Set the background color to gray

        logoPanel.setBackground(Color.black);

        mainFrame.add(logoPanel, BorderLayout.NORTH);

        // Create labels and text fields for username and password

        JLabel titleLabel = new JLabel("Login");

        titleLabel.setFont(new Font("Tahoma", Font.PLAIN, 26));

        titleLabel.setHorizontalAlignment(SwingConstants.CENTER);

        JLabel usernameLabel = new JLabel("Username:");

        usernameLabel.setFont(new Font("Tahoma", Font.PLAIN, 16));

        JTextField usernameField = new JTextField(20); // Increased the width

        usernameField.setFont(new Font("Tahoma", Font.PLAIN, 16));

        JLabel passwordLabel = new JLabel("Password:");

        passwordLabel.setFont(new Font("Tahoma", Font.PLAIN, 16));

        JPasswordField passwordField = new JPasswordField(20); // Increased the width

        passwordField.setFont(new Font("Tahoma", Font.PLAIN, 16));

        JButton loginButton = new JButton("Login");

        // Create a panel for login components using GridBagLayout

        loginPanel = new JPanel(new GridBagLayout());

        GridBagConstraints constraints = new GridBagConstraints();

        constraints.insets = new Insets(5, 5, 5, 5);

        constraints.anchor = GridBagConstraints.CENTER;

        constraints.gridx = 0;

        constraints.gridy = 0;

        constraints.gridwidth = 2;

        loginPanel.add(titleLabel, constraints);

        constraints.gridy = 1;

        constraints.gridwidth = 1;

        loginPanel.add(usernameLabel, constraints);

        constraints.gridx = 1;

        loginPanel.add(usernameField, constraints);

        constraints.gridx = 0;

        constraints.gridy = 2;

        loginPanel.add(passwordLabel, constraints);

        constraints.gridx = 1;

        loginPanel.add(passwordField, constraints);

        constraints.gridx = 0;

        constraints.gridy = 3;

        constraints.gridwidth = 2;

        loginPanel.add(loginButton, constraints);

        // Set the login button action

        loginButton.addActionListener(e -> {

            // Authenticate the user using the UserAccountManager

            String username = usernameField.getText();

            char[] password = passwordField.getPassword();

            if (userAccountManager.authenticateUser(username, String.valueOf(password))) {

                yourName = username; // Set the authenticated user's name

                createChatOptions();

            } else {

                JOptionPane.showMessageDialog(mainFrame, "Authentication failed. Please try again.");

            }

        });

        mainFrame.add(loginPanel, BorderLayout.CENTER);

        userAccountManager = new UserAccountManager(); // Initialize UserAccountManager

        mainFrame.setVisible(true);

    }

    private void createChatOptions() {

        mainFrame.getContentPane().removeAll(); // Clear the login components

        JButton personalChatButton = new JButton("Personal Chat");

        personalChatButton.addActionListener(e -> createPersonalChatWindow());

        personalChatButton.setBackground(new Color(0, 150, 136)); // WhatsApp green color

        personalChatButton.setForeground(Color.WHITE); // White text color

        JButton groupChatButton = new JButton("Group Chat");

        groupChatButton.addActionListener(e -> createGroupChatWindow());

        groupChatButton.setBackground(new Color(0, 150, 136)); // WhatsApp green color

        groupChatButton.setForeground(Color.WHITE); // White text color

        JPanel buttonPanel = new JPanel(new FlowLayout());

        buttonPanel.add(personalChatButton);

        buttonPanel.add(groupChatButton);

        // Set the background color to gray

        buttonPanel.setBackground(Color.GRAY);

        mainFrame.add(buttonPanel, BorderLayout.SOUTH);

        mainFrame.setVisible(true);

    }

    private void createPersonalChatWindow() {

        mainFrame.setVisible(false);

        chatWindows = new ArrayList<>();

        chatTextAreas = new ArrayList<>();

        messageFields = new ArrayList<>();

        JFrame youChatWindow = createChatWindow(yourName, "Friend", "profile\_you.png");

        chatWindows.add(youChatWindow);

        JFrame friendChatWindow = createChatWindow("Friend", yourName, "profile\_friend.png");

        chatWindows.add(friendChatWindow);

    }

    private void createGroupChatWindow() {

        mainFrame.setVisible(false);

        int numFriends = Integer.parseInt(JOptionPane.showInputDialog("Enter the number of friends:"));

        chatWindows = new ArrayList<>();

        chatTextAreas = new ArrayList<>();

        messageFields = new ArrayList<>();

        JFrame youChatWindow = createChatWindow(yourName, "Friend", "profile\_you.png"); // Use "profile\_you.png" here

        chatWindows.add(youChatWindow);

        for (int i = 1; i <= numFriends; i++) {

            JFrame friendChatWindow = createChatWindow("Friend " + i, yourName, "group\_profile.png");

            chatWindows.add(friendChatWindow);

        }

    }

    private JFrame createChatWindow(String title, String friendTitle, String profileFileName) {

        JFrame chatWindow = new JFrame(title);

        chatWindow.setSize(360, 640); // Mobile-sized dimensions

        chatWindow.setDefaultCloseOperation(JFrame.DO\_NOTHING\_ON\_CLOSE);

        chatWindow.addWindowListener(new java.awt.event.WindowAdapter() {

            @Override

            public void windowClosing(java.awt.event.WindowEvent windowEvent) {

                closeAllWindows();

            }

        });

        JPanel topPanel = new JPanel(new BorderLayout());

        topPanel.setPreferredSize(new Dimension(360, 50)); // Adjusted the width

        topPanel.setBackground(new Color(0, 150, 136)); // WhatsApp green color

        JPanel avatarPanel = new JPanel();

        avatarPanel.setPreferredSize(new Dimension(50, 50));

        // Set the profile based on the profileFileName

        URL avatarURL = getClass().getResource("resources/" + profileFileName);

        if (avatarURL != null) {

            ImageIcon avatarIcon = new ImageIcon(avatarURL);

            Image avatarImage = avatarIcon.getImage().getScaledInstance(50, 50, Image.SCALE\_SMOOTH);

            avatarIcon = new ImageIcon(avatarImage);

            JLabel avatarLabel = new JLabel(avatarIcon);

            avatarPanel.add(avatarLabel);

        }

        JLabel nameLabel = new JLabel(title);

        nameLabel.setForeground(Color.WHITE); // White text color

        JButton moreButton = new JButton(":");

        moreButton.addActionListener(e -> {

            JPopupMenu optionsMenu = createOptionsMenu(chatWindow);

            optionsMenu.show(moreButton, moreButton.getWidth(), moreButton.getHeight());

        });

        moreButton.setForeground(Color.black);

        topPanel.add(avatarPanel, BorderLayout.WEST);

        topPanel.add(nameLabel, BorderLayout.CENTER);

        topPanel.add(moreButton, BorderLayout.EAST);

        chatWindow.add(topPanel, BorderLayout.NORTH);

        JTextArea chatTextArea = new JTextArea();

        chatTextArea.setEditable(false);

        chatTextArea.setFont(new Font("Tahoma", Font.PLAIN, 16));

        chatTextArea.setBackground(Color.WHITE); // White background

        chatTextArea.setLineWrap(true);

        chatTextArea.setWrapStyleWord(true);

        JScrollPane chatScrollPane = new JScrollPane(chatTextArea);

        chatWindow.add(chatScrollPane, BorderLayout.CENTER);

        JTextField messageField = new JTextField("Message..");

        messageField.setForeground(Color.GRAY);

        messageField.setFont(new Font("Tahoma", Font.PLAIN, 16));

        messageField.addFocusListener(new FocusAdapter() {

            @Override

            public void focusGained(FocusEvent e) {

                if (messageField.getText().equals("Message..")) {

                    messageField.setText("");

                    messageField.setForeground(Color.BLACK);

                }

            }

            @Override

            public void focusLost(FocusEvent e) {

                if (messageField.getText().isEmpty()) {

                    messageField.setText("Message..");

                    messageField.setForeground(Color.GRAY);

                }

            }

        });

private JFrame createChatWindow(String title, String friendTitle, String profileFileName) {

        JFrame chatWindow = new JFrame(title);

        chatWindow.setSize(360, 640); // Mobile-sized dimensions

        chatWindow.setDefaultCloseOperation(JFrame.DO\_NOTHING\_ON\_CLOSE);

        chatWindow.addWindowListener(new java.awt.event.WindowAdapter() {

            @Override

            public void windowClosing(java.awt.event.WindowEvent windowEvent) {

                closeAllWindows();

            }

        });

        JButton moreButton = new JButton(":");

        moreButton.addActionListener(e -> {

            JPopupMenu optionsMenu = createOptionsMenu(chatWindow);

            optionsMenu.show(moreButton, moreButton.getWidth(), moreButton.getHeight());

        });

        moreButton.setForeground(Color.black);

        topPanel.add(avatarPanel, BorderLayout.WEST);

        topPanel.add(nameLabel, BorderLayout.CENTER);

        topPanel.add(moreButton, BorderLayout.EAST);

        chatWindow.add(topPanel, BorderLayout.NORTH);

        JTextArea chatTextArea = new JTextArea();

        chatTextArea.setEditable(false);

        chatTextArea.setFont(new Font("Tahoma", Font.PLAIN, 16));

        chatTextArea.setBackground(Color.WHITE); // White background

        chatTextArea.setLineWrap(true);

        chatTextArea.setWrapStyleWord(true);

        JButton sendButton = new JButton("Send");

        sendButton.setBackground(new Color(0, 150, 136)); // WhatsApp green color

        sendButton.setForeground(Color.WHITE); // White text color

        JPanel inputPanel = new JPanel(new BorderLayout());

        inputPanel.add(messageField, BorderLayout.CENTER);

        inputPanel.add(sendButton, BorderLayout.EAST);

        chatWindow.add(inputPanel, BorderLayout.SOUTH);

        sendButton.addActionListener(e -> {

            String message = messageField.getText();

            if (!message.isEmpty() && !message.equals("Message..")) {

                String sender = title;

                int index = chatWindows.indexOf(chatWindow);

                chatTextAreas.get(index).append(sender + ": " + message + "\n");

                for (int i = 0; i < chatTextAreas.size(); i++) {

                    if (i != index) {

                        chatTextAreas.get(i).append(sender + ": " + message + "\n");

                    }

                }

                messageField.setText("Message..");

                messageField.setForeground(Color.GRAY);

            }

        });

        chatTextAreas.add(chatTextArea);

        messageFields.add(messageField);

        chatWindow.setVisible(true);

        return chatWindow;

    }

    private JPopupMenu createOptionsMenu(JFrame chatWindow) {

        JPopupMenu optionsMenu = new JPopupMenu();

        JMenuItem signOutItem = new JMenuItem("Sign Out");

        signOutItem.addActionListener(e -> signOut());

        optionsMenu.add(signOutItem);

        JMenuItem clearChatItem = new JMenuItem("Clear Chat");

        clearChatItem.addActionListener(e -> clearChatText(chatWindow));

        optionsMenu.add(clearChatItem);

        return optionsMenu;

    }

private JFrame createChatWindow(String title, String friendTitle, String profileFileName) {

        JFrame chatWindow = new JFrame(title);

        chatWindow.setSize(360, 640); // Mobile-sized dimensions

        chatWindow.setDefaultCloseOperation(JFrame.DO\_NOTHING\_ON\_CLOSE);

        chatWindow.addWindowListener(new java.awt.event.WindowAdapter() {

            @Override

            public void windowClosing(java.awt.event.WindowEvent windowEvent) {

                closeAllWindows();

            }

        });

    private void clearChatText(JFrame chatWindow) {

        int index = chatWindows.indexOf(chatWindow);

        JTextArea chatTextArea = chatTextAreas.get(index);

        chatTextArea.setText("");

    }

    private void signOut() {

        if (chatWindows != null) {

            for (JFrame window : chatWindows) {

                window.dispose();

            }

        }

        chatWindows.clear();

        chatTextAreas.clear();

        messageFields.clear();

        yourName = null;

        mainFrame.getContentPane().removeAll();

        mainFrame.add(loginPanel);

        mainFrame.setVisible(true);

    }

    private void closeAllWindows() {

        if (chatWindows != null) {

            for (JFrame window : chatWindows) {

                window.dispose();

            }

        }

        System.exit(0);

    }

    public static void main(String[] args) {

        SwingUtilities.invokeLater(() -> new ChatApplication());

    }

}

This code represents a simple chat application in Java using Swing for the graphical user interface. It allows users to log in and engage in personal and group chats. Here's an explanation of the code:

1. **ChatApplication Class:** This is the main class representing the chat application.

2. **Instance Variables:**

- `**mainFrame`:** A `JFrame` is the main window of the application.

- `**loginPanel`:** A `JPanel` is used for the login interface.

- `**chatWindows`:** A list of `JFrame` instances for personal and group chat windows.

- **`chatTextAreas`:** A list of `JTextArea` instances for chat message display.

- `**messageFields`:** A list of `JTextField` instances for composing and sending messages.

- `**yourName`:** A string variable to store the authenticated user's name.

- **`userAccountManager`:** An instance of the `UserAccountManager` class for user authentication.

3. **Constructor `ChatApplication()`:**

- Creates the main application window (`mainFrame`) with a logo panel at the top.

- Sets up the login interface with labels, text fields for username and password, and a login button.

- The login button's action listener checks the provided username and password against the user accounts managed by `UserAccountManager`. If authentication succeeds, the user is allowed to create chat options.

4. `**createChatOptions()`:**

- Removes the login components from the main frame.

- Adds buttons for personal and group chat options to the bottom panel.

5. **`createPersonalChatWindow()`:**

- Hides the main frame.

- Creates personal chat windows by invoking `createChatWindow()`, one for the authenticated user and one for a friend.

6. **`createGroupChatWindow()`:**

- Hides the main frame.

- Prompts the user to enter the number of friends for a group chat.

- Creates a personal chat window for the authenticated user.

- Creates multiple chat windows for group members using `createChatWindow()`.

7. **`createChatWindow(String title, String friendTitle, String profileFileName)`:**

- Creates a chat window with a title (user's name), a friend's title, and a profile image.

- Handles window close events to prompt the user before exiting.

- Sets up the chat window with user's and friend's information and an option button.

- Allows sending and displaying messages in a chat area.

8. **`createOptionsMenu(JFrame chatWindow)`:**

- Creates a popup menu with options for signing out and clearing the chat.

9. **`clearChatText(JFrame chatWindow)`:**

- Clears the chat area for a specific chat window.

10. **`signOut()`:**

- Closes all chat windows when the user signs out.

- Clears the lists of chat windows, text areas, and message fields.

- Returns to the login panel.

11. **`closeAllWindows()`:**

- Closes all chat windows and exits the application when the user closes the main window.

12. **`main(String[] args)`:**

- Entry point of the application.

- Invokes the `ChatApplication` class on the Event Dispatch Thread (EDT) to ensure Swing components are created and updated on the EDT.

Overall, this code provides a basic chat application with a graphical user interface for login and chat functionality. It includes personal and group chat options, user authentication, and message management.

The given two Java files, **`UserAccountManager.java`** and **`ChatApplication.java`**, are interlinked through the following mechanisms:

1**. User Account Management:**

- `UserAccountManager.java` is responsible for managing user accounts. It provides methods for authenticating users, adding new users, updating user passwords, and deleting user accounts.

- `ChatApplication.java` uses an instance of `UserAccountManager` to authenticate users when they attempt to log in. If a user is authenticated, they can access the chat application's features.

2. **User Authentication:**

- In `ChatApplication.java`, when the user clicks the login button, it calls the `authenticateUser` method from `UserAccountManager` to check if the provided username and password match an existing user account.

- If authentication succeeds, the authenticated user's name is stored in the `yourName` variable in `ChatApplication.java`, and the chat options become available.

3. **Chat Windows and Messages:**

- `ChatApplication.java` manages the chat interface, including creating personal and group chat windows using `JFrame` instances, handling message input, and displaying messages in `JTextArea` components.

- When a user sends a message, it's appended to the relevant `JTextArea` in the `chatTextAreas` list.

- The user's name is used in the chat windows, and messages are displayed with the sender's name, which is obtained from the chat window's title.

4. **Sign Out and Clear Chat:**

- Both `ChatApplication.java` and `UserAccountManager.java` work together to implement the "Sign Out" and "Clear Chat" functionalities.

- The "Sign Out" option in `ChatApplication.java` invokes the `signOut` method, which closes chat windows and resets the user's state. This functionally requires `UserAccountManager.java` to clear its stored user name.

- The "Clear Chat" option in `ChatApplication.java` uses `clearChatText`, which operates on chat text areas and clears the chat messages in the selected window.

These interconnections enable the chat application to provide user authentication, user account management, and chat functionality while maintaining user data integrity and chat window interactions. The two files work together to create a complete chat application.

**Add, Edit**, or **Delete** **Username**s and **Password**s

public UserAccountManager() {

        userAccounts = new HashMap<>();

        // You can load existing user accounts from a file or a database here.

        // For demonstration purposes, we'll add a default user.

        userAccounts.put("nawaz", "786");

        userAccounts.put("sravan", "java");

    }

To **add, edit**, or **delete** **username**s and **password**s in the code, you'll need to make changes in both UserAccountManager.java and ChatApplication.java. Here are the steps for each operation:

**1. Adding a User:**

In UserAccountManager.java:

java

Copy code

public boolean addUser(String username, String password) {

// Add a new user to the userAccounts map.

if (!userAccounts.containsKey(username)) {

userAccounts.put(username, password);

return true;

}

return false; // User with the same username already exists.

}

This method allows you to add a new user with a username and password. Call this method with the desired username and password when you want to add a new user.

In ChatApplication.java:

You can add a UI component, such as a "Sign Up" button, to trigger the user addition process. When the user clicks this button, you can prompt them to enter a new username and password. Then, you call the addUser method from UserAccountManager with the entered credentials to add the new user.

**2. Editing a Password:**

In UserAccountManager.java:

java

Copy code

public boolean updateUserPassword(String username, String newPassword) {

// Update the password for an existing user.

if (userAccounts.containsKey(username)) {

userAccounts.put(username, newPassword);

return true;

}

return false; // User not found.

}

This method allows you to update the password for an existing user. Call this method with the username of the user whose password you want to change and the new password.

In ChatApplication.java:

You need to provide a way for users to edit their passwords. This could be done through a "Change Password" button or menu item. When a user wants to change their password, you can prompt them to enter the new password and then call the updateUserPassword method from UserAccountManager with the current username and the new password.

**3. Deleting a User:**

In UserAccountManager.java:

java

Copy code

public boolean deleteUser(String username) {

// Delete a user account.

if (userAccounts.containsKey(username)) {

userAccounts.remove(username);

return true;

}

return false; // User not found.

}

This method allows you to delete a user account. Call this method with the username of the user you want to delete.

In ChatApplication.java:

You can add a UI component, such as a "Delete Account" button, to trigger the user deletion process. When the user clicks this button, you can prompt them to confirm the deletion and then call the deleteUser method from UserAccountManager with the username of the user to be deleted.

Make sure to add appropriate user interface elements in ChatApplication.java to interact with these methods in UserAccountManager.java. For example, buttons, input fields, and dialogs can be used to get user input and trigger these actions

**Test case:**

Table of the basic test cases for a chat application:

Based on Authentication, User Management, Chat

|  |  |  |  |
| --- | --- | --- | --- |
| Category | Test Case | Description | Expected Result |
| Authentication | Valid Credentials | Enter valid username and password, and click the "Login" button. | User is authenticated and the chat options are displayed. |
| Authentication | Invalid Username | Enter an invalid username and a valid password, and click the "Login" button. | Application displays an error message. |
| Authentication | Invalid Password | Enter a valid username and an invalid password, and click the "Login" button. | Application displays an error message. |
| User Management | Add User | Click the "Sign Up" button, enter a new username and password, and click the "Submit" button. | New user is added to the UserAccountManager. |
| User Management | Edit Password | Click the "Change Password" button, enter the current username, old password, and new password, and click the "Submit" button. | Password is updated in the UserAccountManager. |
| User Management | Delete User | Click the "Delete Account" button, enter the username to be deleted, and confirm the deletion. | User is removed from the UserAccountManager. |
| Chat | Personal Chat | Click the "Personal Chat" button, verify that a personal chat window is opened, send a message from one user to the other, and verify that the message is displayed in both users' chat windows. | Personal chat window opens, message is sent, and message is displayed in both users' chat windows. |
| Chat | Group Chat | Click the "Group Chat" button, enter the number of friends and click "Submit", verify that a group chat window is opened with the specified number of friends, send a message in the group chat, and verify that the message is displayed in all group members' chat windows. | Group chat window opens, group members are added, message is sent, and message is displayed in all group members' chat windows. |
| Chat | Clear Chat | Click the "Clear Chat" option in the chat window, and verify that the chat window is cleared. | Chat window is cleared. |
| Chat | Sign Out | Click the "Sign Out" option in the chat window, and verify that the user is signed out, and the login page is displayed. | User is signed out, and the login page is displayed. |

The project consists of two main Java classes: ChatApplication and UserAccountManager, which work together to create a simple chat application with user account management. The ChatApplication class is responsible for the user interface and chat functionality, while the UserAccountManager class handles user authentication and account management.

The ChatApplication class features a GUI for user login and chat options. Users can log in, access personal and group chat windows, send and receive messages, and sign out. It provides a user-friendly interface with profile pictures and a clean design.

The UserAccountManager class stores user accounts with usernames and hashed passwords. It supports user authentication, adding new users, updating passwords, and deleting user accounts. For demonstration purposes, it contains default user accounts.

**Conclusion**

The chat application presented in this project showcases the value and versatility of modern communication tools. Chat applications have transformed the way we connect, collaborate, and communicate. Whether for personal use, business purposes, or educational environments, these applications offer an array of benefits. They enable users to exchange messages, multimedia content, and files instantly. Group chats and channels support teamwork and collective decision-making. With a global reach and cost-efficient communication options, users can stay connected with friends, family, and colleagues across the world. Chat applications are essential for sharing rich media, including images, videos, and documents. Additionally, their notifications keep users updated in real-time. Robust user account management ensures privacy and security, making these applications a reliable choice for personal, professional, and educational needs. As technology continues to advance, chat applications remain at the forefront of modern communication, enhancing connectivity and collaboration in our increasingly interconnected world.

There is always a room for improvements in any apps. Right now we are just dealing with text communication. There are several android apps which serve similar purpose as this project, but these apps were rather difficult to use and provide confusing interfaces. This project hopes to develop a chat service Android app with high quality user interface In future we can extend to include features such as:File Transfer, Voice Message, Video Message, Audio Call, Video Call, Group Call

**Sites Referred:**

<https://github.com/>

<https://chat.openai.com/>

<https://www.javatpoint.com/>