ChatApp

Mahmoud Ishag Adam Abdelrahim

541409

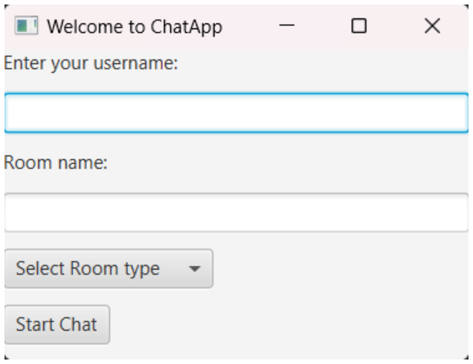
Description:

The Chat Messaging App is a Java-based object-oriented programming (OOP) project designed to facilitate communication by allowing users to send and receive real-time messages. User-friendly GUI is designed using JavaFx to enhance the experience. The project incorporates various OOP concepts such as modularity, hierarchy, composition, reuse, encapsulation, subtyping, information hiding, abstraction, inheritance, polymorphism, and exception handling.

Running Steps:

- Compile all folders using Javac command or appropriate IDLE like Eclipse or IntelliJ.
- Run Server class to initialize server
- Run ChatAppGUI class to start the application GUI, the following window appears:

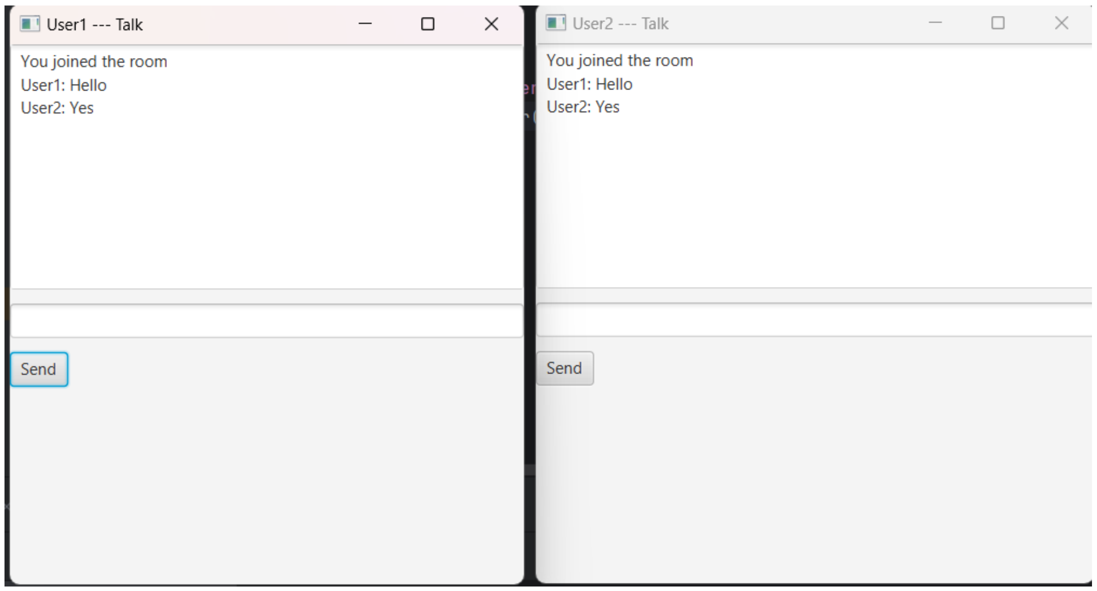
Welcome window



The first point of interaction for users providing the initial options or input field before heading to the main chat interface.

- Username: textField where user inserts his desired name. once inserted User object is created.
- roomName: users can join the room by inserting its name, associated random ID is generated and used for later verification.
- Room Selection: the program supports Public and private chats.
- Start Chat: to send the inserted information to back-end and then call ShowChatScene

Chat window



You joined the room
User1: Hello
User2: Yes

You joined the room
User1: Hello
User2: Yes

- Title: Username ---- Room
- TextArea: All Sent and received messages appear here
- TextField: To insert messages to be sent
- Send Button: creates new Message object with provided information then broadcast the message to connected users.

Object Oriented Programming Concepts and Aspect that have been implemented:

Modularity

The project is divided into multiple classes (ChatAppGUI, ChatConnectionHandler, Message, WelcomeScene, ChatScene, User, PrivateRoom, PublicRoom, Room, and Server) to promote modularity and enhance code organization.

Hierarchy and Inheritance:

Hierarchy is established with the Room class serving as the base class for PublicRoom and PrivateRoom.

Inheritance is utilized to create specialized room types, inheriting common attributes and behaviors from the base class.

```java
public abstract class Room
public class PublicRoom extends Room
public class PrivateRoom extends Room
```

Composition:

- Describes relationship between objects where one object contains one or more objects of other classes. It represents HAS-A relationship.
- ChatScene class has various objects of UI components. When instantiating Chatscene all these objects instantiated concurrently.

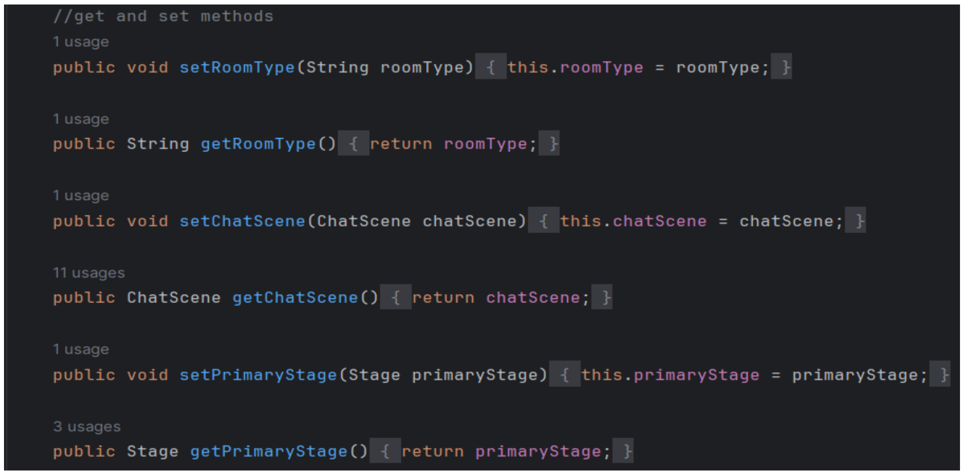
```java
• public ChatScene() {

      // Initialize UI components
      root = new VBox(10);
      chatTextArea = new TextArea();
      messageField = new TextField();
      sendButton = new Button("Send");
```

Reuse:

Code reuse is demonstrated through various OOP principles to avoid code repetition. For instance, JavaFx librabry is used to build GUI benefiting from its pre-defined components. TextField, VBox and Button are used to build WelcomeScene and ChatScene

Encapsulation:

Refers to putting together set of members (attributes and methods) in single class unit. Restricting direct access to some of object components and providing secure access by utilizing getter and setter methods.



```java
//get and set methods
1 usage
public void setRoomType(String roomType) { this.roomType = roomType; }

1 usage
public String getRoomType() { return roomType; }

1 usage
public void setChatScene(ChatScene chatScene) { this.chatScene = chatScene; }

11 usages
public ChatScene getChatScene() { return chatScene; }

1 usage
public void setPrimaryStage(Stage primaryStage) { this.primaryStage = primaryStage; }

3 usages
public Stage getPrimaryStage() { return primaryStage; }
```

Information Hiding:

ensures that an object's internal structure is hidden from other objects. It restricts direct access to an object's internal state and separate between the public interface and private implementation. From the perspective of a user only the interface of an object is visible and accessible. Is achieved using access modifiers private, public

Abstraction:

Room is defined as abstract class to provide member variables and methods that are wholly shared by all subclasses. This class is not instantiable.

addNewUser is declared as abstract method does not have a body. The body is provided by the subclass.

```
public abstract class Room {
        // attributes and methods shared

        public abstract void addNewUser(User user);
        //
}
```

Inheritance:

- Subclasses PublicRoom and PrivateRoom inherit the full implementation of their superclass Room which promotes code reuse and allows extension. Existed method addNewUser is overridden to function differently depending on subclass. Additional methods are added.

```
public class PrivateRoom extends Room{
        //
        @Override
        public void addNewUser(User user) {
                if (!isRoomFull()){
                getUsers().add(user);
    }

}
public class PublicRoom extends Room{
        //
        @Override
        public void addNewUser(User user) {
                getUsers().add(user);
        }

}
```

- On the GUI part, ChatAppGUI extends Application class which is required for create JavaFx application. Start() method is overridden to initialize UI components

```
public class ChatAppGUI extends Application implements ConnectionHandler {
        //
        @Override
        public void start(Stage primaryStage) {
                //
        }

}
```

Interface:

ConnectionHandler interface contains abstract methods without providing their implementation. Classes implement this interface must define the body of these methods.

Runnable interface is used to encapsulate the code that should run in new thread. By overriding run method and including tasks to be executed concurrently.

```
public interface ConnectionHandler {
    //Public abstract methods
    void startReadingMessages();
    void sendMessage(Message message);
    void closeConnection();
}
```

```
public class ChatConnectionHandler implements ConnectionHandler, Runnable {
        //
        @Override
        public void run()  {
                //method from Runnable interface
        }

        @Override
        public void startReadingMessages(){
                //method from ConnectionHandler interface
        }


}
```

Polymorphism:

- Overloading: Having multiple Message methods in the same class but with different parameters.

```
public Message(String room, String content) {
     //
}
public Message(User sender, String room, String content) { //
}
```

- Polymorphism by inclusion: achieved by overriding existing methods of superclass in their subclasses. enables method calls on objects to be resolved dynamically at runtime (dynamic binding). All abstract methods in the project have been overridden in their subclasses or subtype.

```
getCurrentRoom().addNewUser(getUser());     //invokes addNewUser according to room Type (public, Private)
```