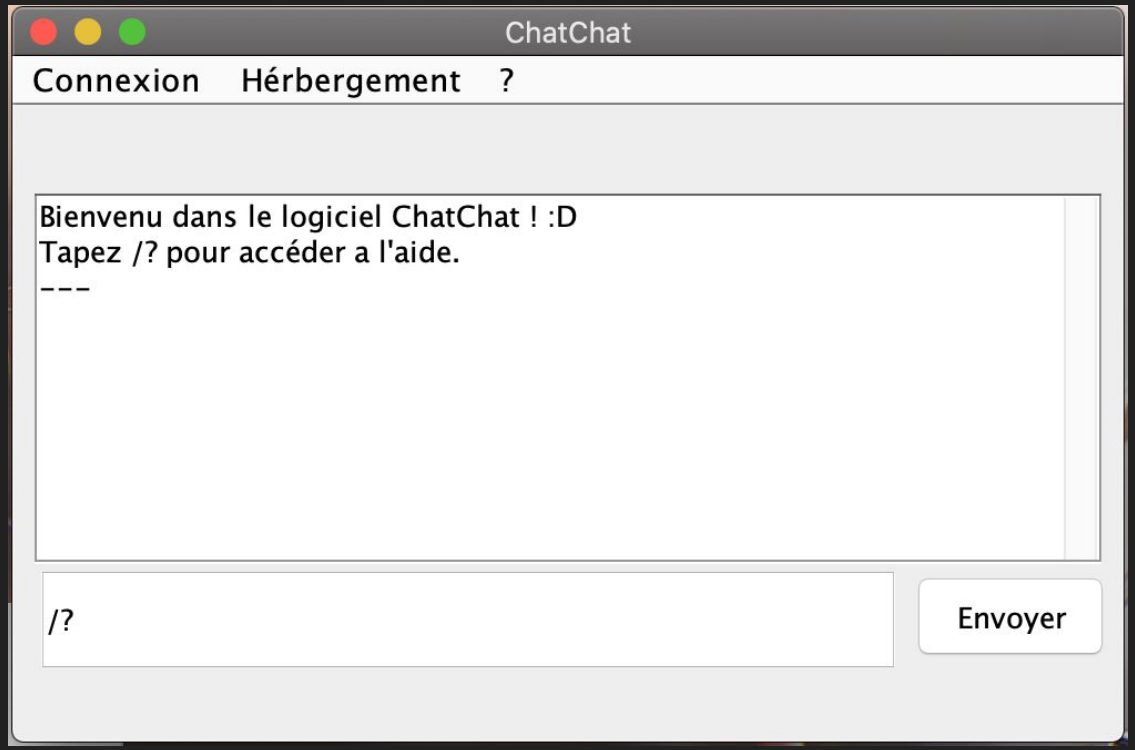
-Sommaire-

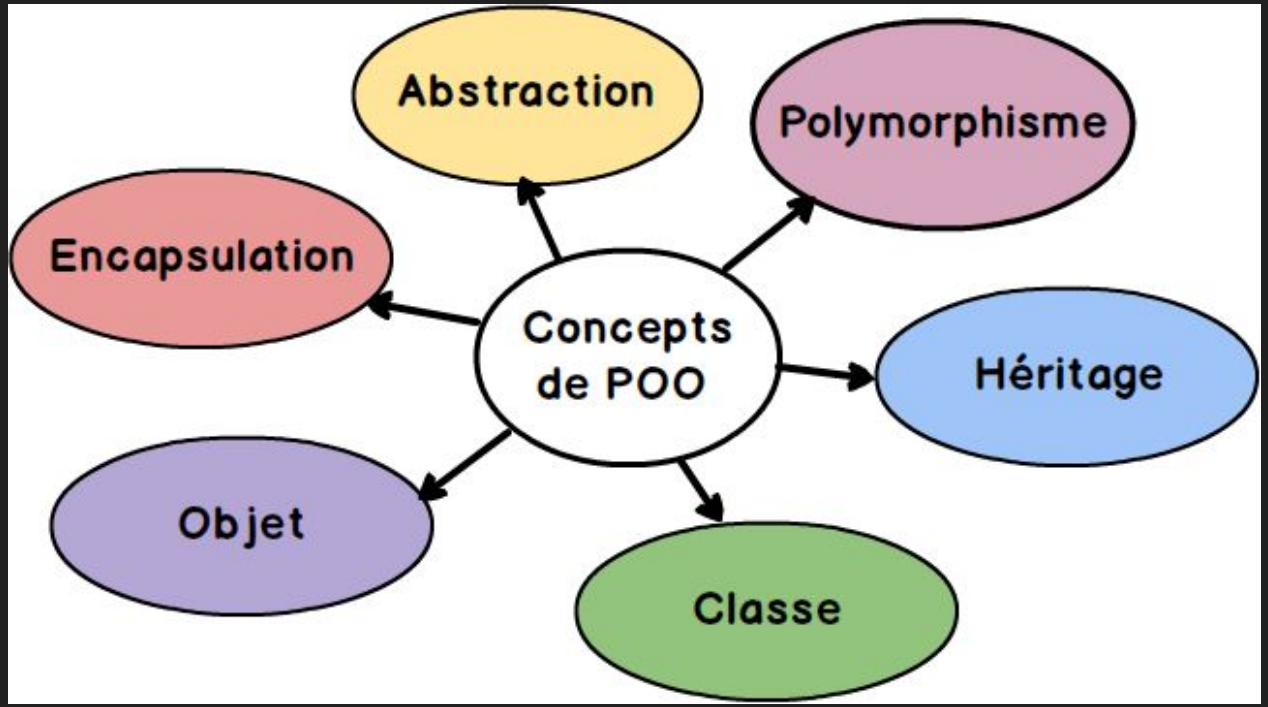
1. Présentation
2. La POO
3. La Structure
4. La GUI
5. Les Fonctionnalités
6. Le Réseau

Projet informatique

Logiciel de Chat - Niveau Expert



La POO ? Mais qu'est-ce que c'est ?



- Découverte
- Projet
- Swing
- Socket

La **structure globale** de notre programme

```
public class ChatChat implements Runnable, ActionListener, ComponentListener, KeyListener
{
```

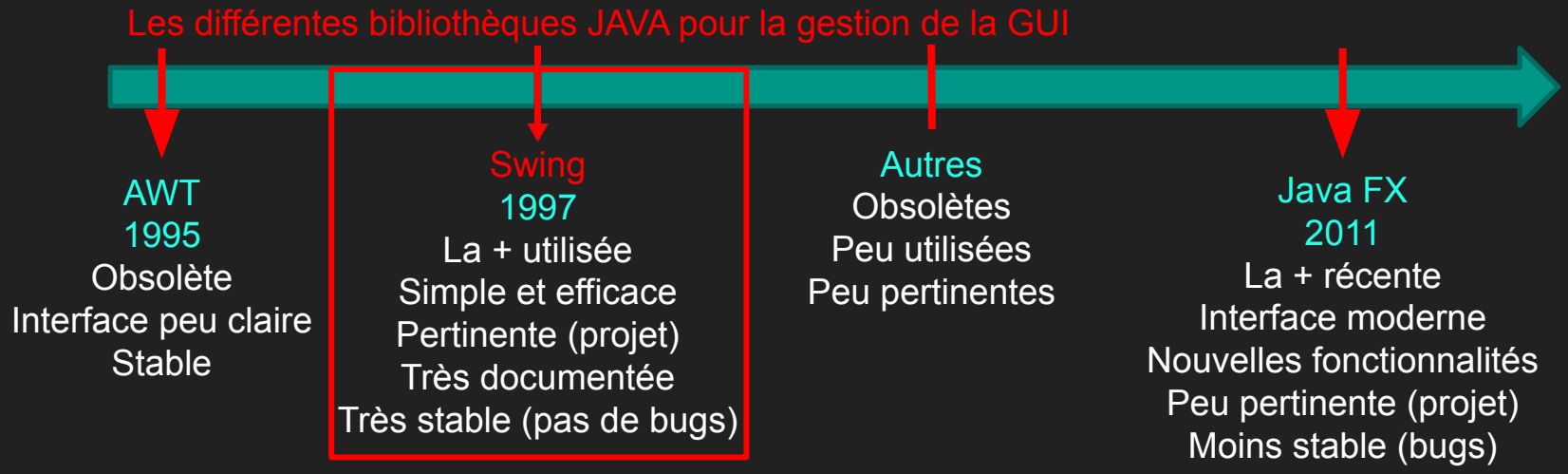
```
class ReceivedMessagesHandler implements Runnable
{
```

```
public class Server implements Runnable {
```

```
class User
{
```

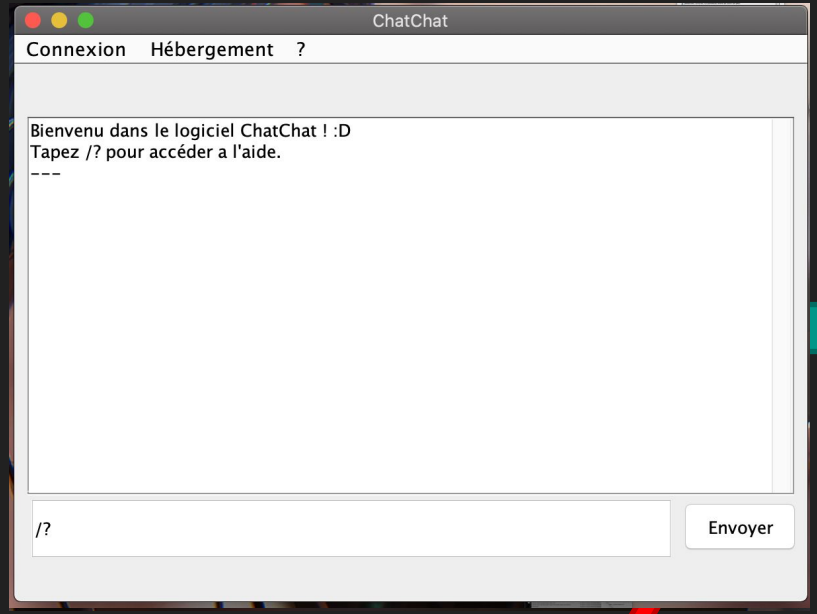
```
class UserHandler implements Runnable
```

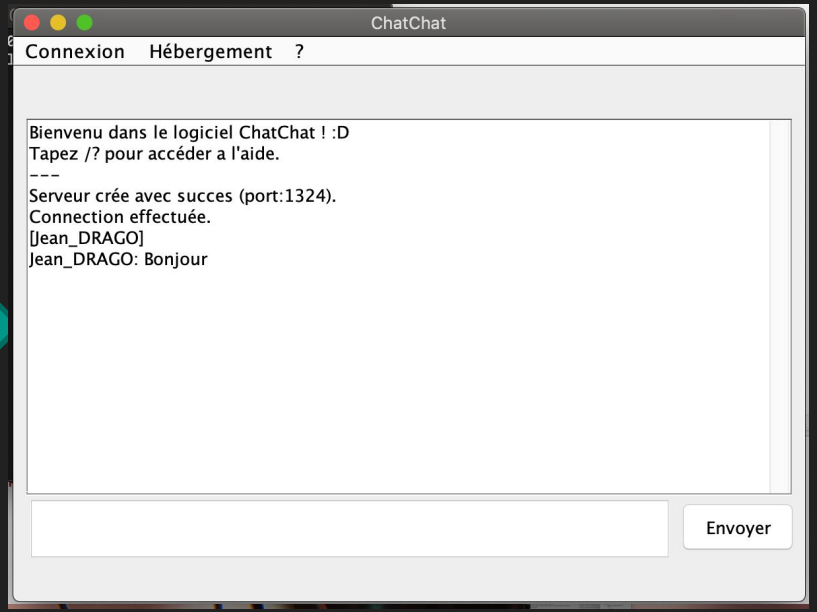
La gestion de la GUI (interface graphique)



Classe **JFrame** (objet frame) La fenêtre créée et agrémentée d'icônes (“Aide”,...)
Classe **JTextField** (objet textfield) La barre de saisie du message de taille 1 ligne texte
Classe **JTextArea** (objet textarea) La zone texte principale où sont affichés les messages envoyés
Classe **JButton** (objet bouton) Le bouton d’envoi du message saisi

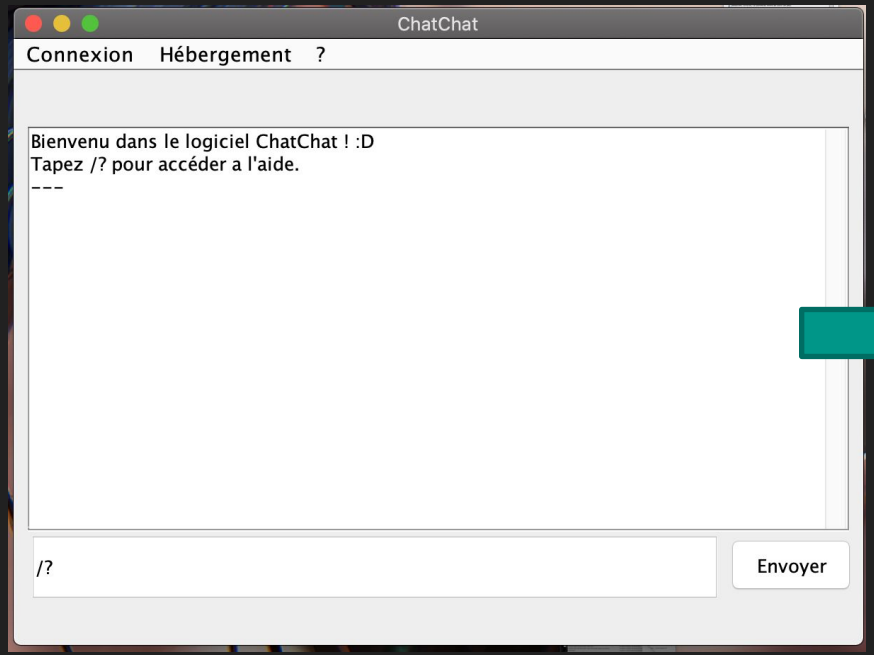
Les fonctionnalités du programme n°1

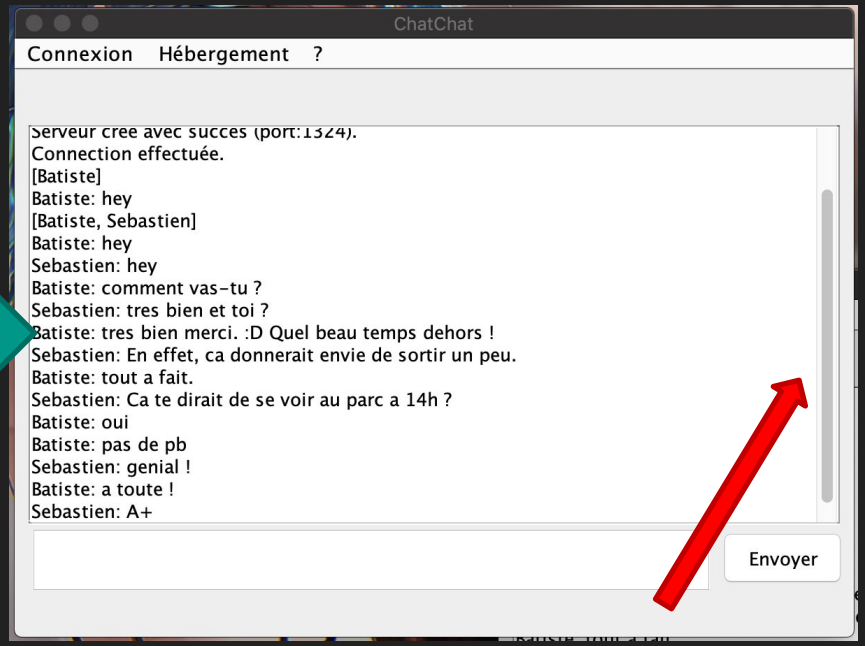




private void validateInput()

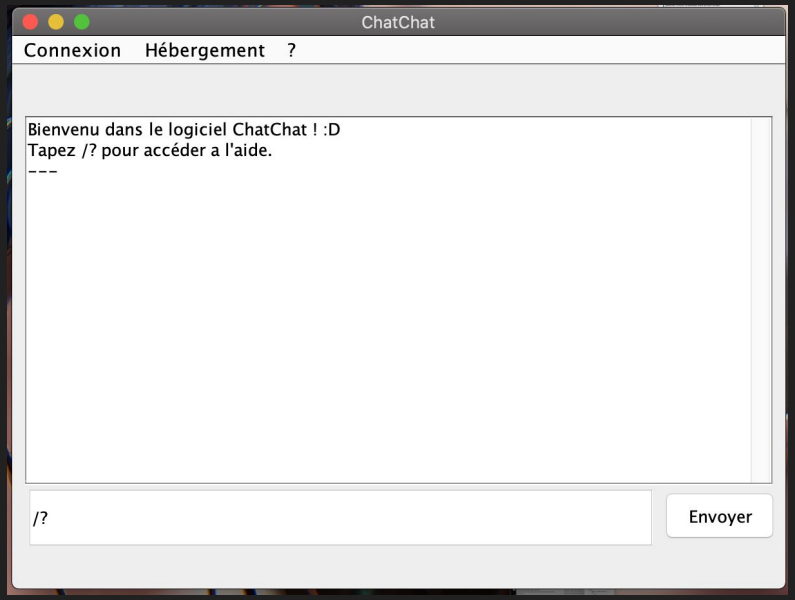
Les fonctionnalités du programme n°2

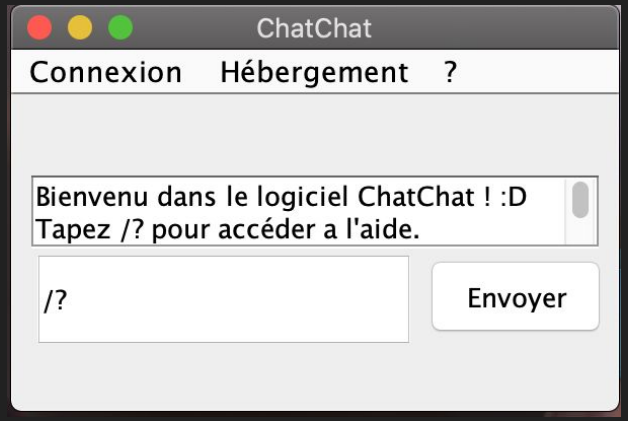




public void write(String message)

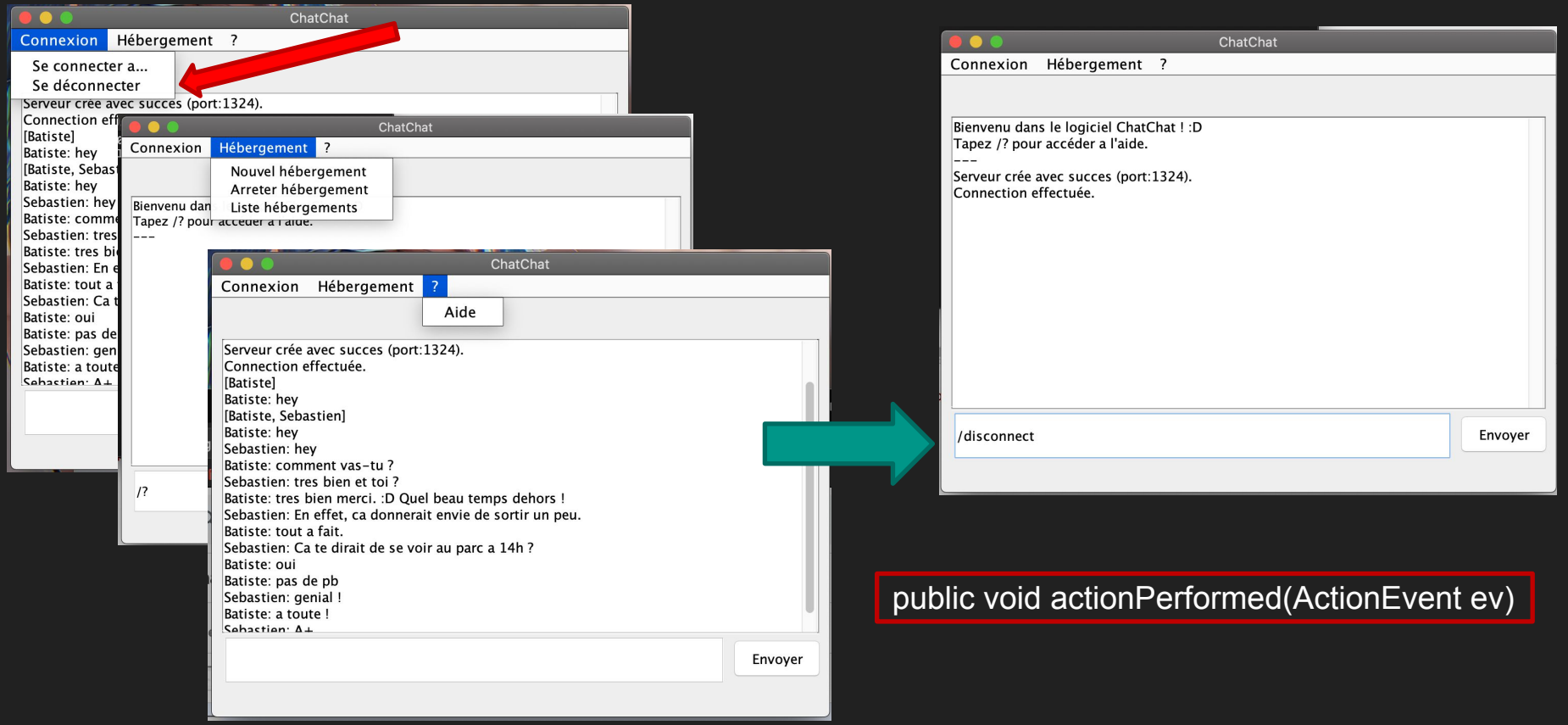
Les fonctionnalités du programme n°3





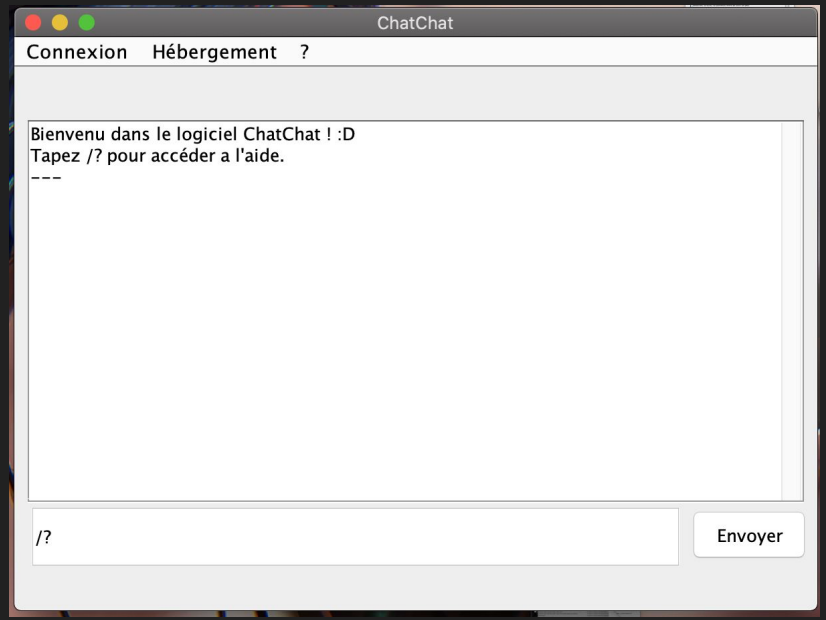
private void adaptComponentsBounds()

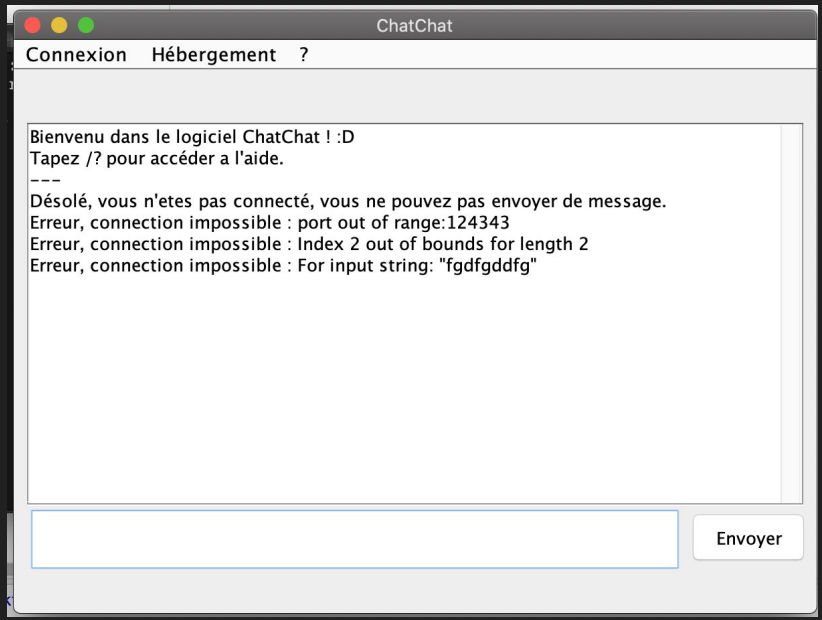
Les fonctionnalités du programme n°4



public void actionPerformed(ActionEvent ev)

Les fonctionnalités du programme n°5





Le **réseau**

- Socket
- SocketServer
- Gestion des flux

Recherche

➔ Exemples

➔ Forums

➔ Documentation

Client **Server**

1. OutputStream -->\ /--> 2. InputStream -->

Socket <--> network <--> ServerSocket

4. InputStream <--/ \<--3. OutputStream ←

Source : stackoverflow.com

La gestion des commandes utilisateurs

- Mécanisme des “ Exceptions ”
- Fonction handleCommand(String command)

```
private void handleCommand(String command)
{
    String[] subCommands = command.split(" ");

    if( subCommands[0].equals("?") )
    {
    else if( subCommands[0].equals("clear") || command
    {
    else if( subCommands[0].equals("startserver") || s
    {
    else if( subCommands[0].equals("stopserver") )
    {
    else if( subCommands[0].equals("connect") )
    {
    else if( subCommands[0].equals("listserver") )
    {
    else if( subCommands[0].equals("disconnect") )
    {
    else if( subCommands[0].equals("getip") )
    {
    else
    {
}
```

Exemple de la commande “startserver”

```
else if( subCommands[0].equals("startserver") || subCommands[0].equals("news
{
    try { // create a new server
        final int port = Integer.parseInt(subCommands[1]);
        servers.add(new Server(port));
        new Thread(servers.get(servers.size()-1)).start();
        write("Serveur cr\u00E9e avec succes (port:"+port+").");
    }
    catch(Throwable e)
    {
        write("Erreur, impossible de cr\u00E9er le server: "+e.getMessage())
    }
}
```

=> Le *message d’erreur lié aux exceptions* est affiché.

Merci d’avoir visionné notre présentation