

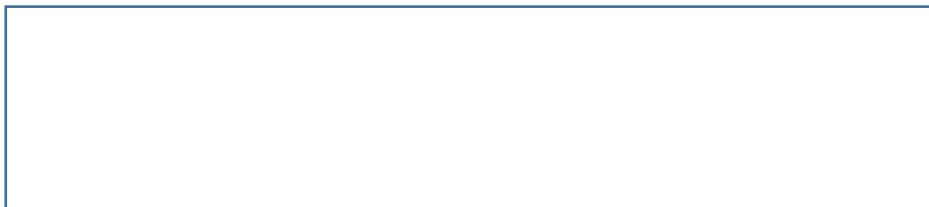
East London Science School

JarChat

OCR A-Level Computer Science Project

Adam Tazul

2022-05-09



Contents

1: Analysis	2
1.1: Aims of JarChat	2
1.2: Target Audience	2
1.3: Research.....	2
1.3.1: Existing IRC Clients	2
1.3.2: Features of JarChat	4
1.3.3: Limitations of JarChat	4
1.4: Community input	4
1.5: Requirements.....	5
1.5.1: Software	5
1.5.2: Hardware	5
1.6: Success Criteria	5
2: Design.....	5
2.1: UI Design	5
2.1.1: Start-up Screen	5
3: Development.....	6
3.1: Connection to IRC	6
3.2: The code used in the library	6
3.3: Sending messages, joining/leaving channels	7
4: Testing, Review and Evaluation	7
4.1: Achieved Success Criteria	7
4.1.1: Proof of achieved Success Criteria	8
4.2: Limitations	8
4.2.1: Avoiding these limitations & Maintenance	8
5: Final Code.....	9
5.1: IRCMessageLoop.java	9
5.2: JarChat.java	11

1: Analysis

The following two subsections analyses the aims I am going into JarChat as a project with, as well as who will most likely be a stakeholder in the project.

1.1: Aims of JarChat

JarChat will aim to be a cross-platform IRC client which will support Windows, MacOS, Linux, and maybe more Operating Systems. This will have the benefit of native cross-platform support as it is built-in to Java. Java's JDK version 8 (Java version 1.8) will be used to compile all binaries as most computers still run JRE version 8, even though Oracle has moved their LTS (Long Term Support) version to JRE 11.

1.2: Target Audience

I expect everyone who uses a computer to be able to take advantage of JarChat. Various FOSS (Free and Open-Sourced Software) use IRC as the chat protocol used for support.

JarChat can also expect to be popular with developers in any programming language, as there are still some programming help channels/servers hosted using IRC. The same will be true for new/existing users of most Linux distributions. If someone has a question regarding Linux and can't be answered by Google, they can ask in the appropriate channel on [Libera.chat](https://libera.chat).

More generally, the audience being targeted by JarChat are programmers and users of FOSS.

1.3: Research

In the following subsections, I research other clients to see what they do best and take inspiration from them. I also will be taking input from the community as for the kind of direction people would prefer JarChat to take.

1.3.1: Existing IRC Clients

1.3.1.1: Windows

Existing Windows clients I have considered include mIRC and XChat.

1.3.1.1.1: mIRC

mIRC is a paid and proprietary software which costs £17.94 (though it offers a 30-day free trial). It provides a slightly dated User Interface and was compiled in 32-bit form. This makes it compatible with 32-and-64-bit versions of Windows from Windows XP up to Windows 11. The UI makes use of a template to store different tabs as windows on a canvas. These windows cannot be moved or used in other parts of Windows. The settings menu leaves much to be desired and

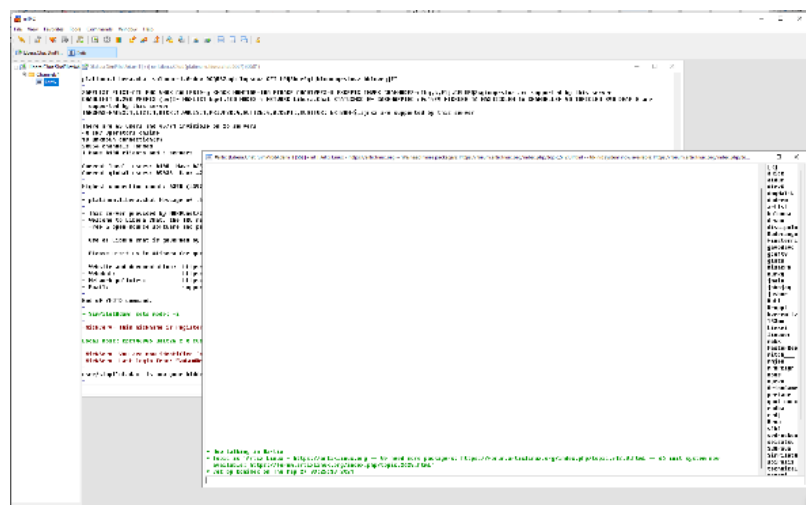


Figure 1: Default look of mIRC, running on Windows 11 build 22H2.1000 (Dev Insider Preview).

has few customization options. There is no option for IRC's VOIP feature included with the software.

I intend to take inspiration from mIRC's raw functionality as it works very well (albeit only on Windows). I do plan, however, to expand this functionality to take advantage of all of the features that are included on IRC's feature set.

1.3.1.1.2: XChat

XChat is another IRC client. Written in C, the Linux version is free and open-sourced under the GNU GPLv2 license. The client is precompiled for Windows and Fedora GNU/Linux, with forks available for Arch Linux (in the AUR), and with a precompiled *.deb file in the official Debian and Ubuntu repositories. The Windows version of the software is closed-sourced and proprietary, costing users US\$19.99 (equivalent to £14.42 in September of 2021), though it offers a 30-day free trial. The Windows version officially supports all Windows varieties from Windows 2000 up to Windows 10 (in 32-bit).

I tried to use XChat to connect to an IRC server, but the Windows version failed to do so on Windows Vista, 7, 8, 8.1, 10, and 11. This lack of ease of use is an issue that I would like to resolve with JarChat.

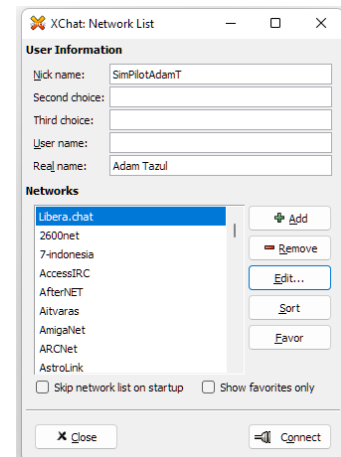


Figure 2: The first screen of the Windows version of XChat, running on Windows 11 build 22449.1000 (Dev Insider Preview).

1.3.1.2: Linux

Existing Linux IRC clients I have studied include Konversation and WeeChat.

1.3.1.2.1: Konversation

Konversation is a Linux IRC client which is tightly integrated with the KDE Plasma Desktop Environment. It is fully free & open-sourced under the GNU GPLv2 license. As a part of the KDE Applications suite. Konversation is precompiled for all major distributions of Linux & GNU/Linux, with source code hosted on the official KDE git repository (<https://invent.kde.org/network/konversation>).

I use Konversation frequently, as my desktop setup uses Artix Linux with KDE Plasma, and its tight integration allows accurate and appropriate theming in line with the global QT5 theme, no matter what theme is used. It also uses Kcolorchooser to allow for the user to create a custom colour for Konversation's theme, even if the user is not using a QT5-based theming engine.

Using Konversation, I found that it is a fully-featured client with support for every part of IRC, displayed in a user-friendly layout that is easy-to-use. I intend to take inspiration from this and remake these features in JarChat.

1.3.1.2.2: WeeChat

WeeChat is a Terminal User Interface IRC client for Linux which can be used on any Linux Distribution without the need for any Desktop Environment or Window Manager. It is also FOSS under the GNU GPLv3 license.

While strictly TUI and Keyboard Interaction only (unless the setting is changed), WeeChat still provides a fully-featured experience with support for every part and situation possible in IRC, with every customization as needed by anyone who would use it. Since WeeChat is TUI, it is aimed at people who are willing to learn how to use it, which can take some time. For this very reason, there are a lot of people who have installed it, tried to use it, and then immediately uninstalled it as they found it

extremely difficult to get started in (even with the existing documentation provided by the developers). This is understandable as WeeChat was designed with minimalism at the forefront of the developers' minds.

1.3.1.3: MacOS

Even though there are one or two IRC clients for MacOS, I do not have the means to test this. Creating virtual machines with MacOS installed on them have proved to create bugs at best, and hackintoshing (installing MacOS on computers not made by Apple) is also problematic, with stringent hardware requirements I do not meet. In any case, trying to do either will result in breaching Apple's EULA.

1.3.2: Features of JarChat

JarChat is intended to include:

- Basic functionality as an IRC Client
- Cross-compatibility between various different operating systems without compromising on features
- Vast configurability through an easy-to-use UI
- GNU GPLv3 Open-Sourced license to allow for the community to better help shape the future of JarChat

1.3.3: Limitations of JarChat

The main limitation of JarChat is the fact that it will be written in Java, which means that JarChat will require and expect the user to have a JVM compatible with Java version 1.8.0 already installed on their system. While Java is generally installed on millions of computer systems globally, it cannot be guaranteed that everyone will be able to use JarChat for lack of the ability to install the software.

1.4: Community input

I am already taking input from the community about the project. Since all source code and development will remain public under the GNU GPLv3 license, it is relatively easy to get community input at every stage in the development cycle. To start things off, I posted an anonymous survey in the [##libera](#) chat in the most popular IRC server, [libera.chat](#). All results of this survey can be found in the protected Microsoft Excel file in the same folder as this PDF.

The questions in this survey include:

1. How many hours per week do you usually spend logged into any IRC server?
2. Which servers do you frequent?
3. Which channels do you typically frequent when in those servers?
4. What do you mainly do when logged into IRC?
5. What do you look for in an IRC client?
6. Which IRC client do you use currently?
7. What OS are you using?

The most popular answers are as follows:

1. Between 25 and 30 hours
2. Freenode (before the change of ownership), Libera.chat
3. Various Linux Distribution/software development support channels
4. Ask for/provide support from/to other users of the channel
5. Usability, theming, FOSS
6. A mix of Konversation, HexChat, and WeeChat
7. Mostly Linux distributions, with some Windows users scattered around

1.5: Requirements

These are software and hardware requirements I am expecting JarChat to have.

1.5.1: Software

- Windows:
 - Vista SP2/Server 2008 R2 SP1 (or newer)
 - At least Internet Explorer 9, or Firefox (a dependency of Java)
- Linux:
 - Any distribution which has Java JRE version 8 in its repositories
 - Firefox (a dependency of Java)
- MacOS
 - MacOS X 10.8.3 or newer (not compatible with Apple Silicon)
 - Any 64-bit web browser (a dependency of Java)
- Java version 1.8 (OpenJDK JRE version 8 in most Linux repositories)

1.5.2: Hardware

- Windows & Linux:
 - Pentium 2 266MHz CPU or better
 - 128MB RAM or better
 - 124MB free secondary storage or better
- MacOS
 - An Intel CPU
 - 124MB free secondary storage or better

1.6: Success Criteria

In order to be successful, JarChat will need to:

- Function as a fully working IRC client
- Be user-friendly
- Be able to use SSL encryption for servers which use it
- Run on any OS that Java version 1.8.0 can be installed on

In order to check that each criterion is met, JarChat will need to

- Be tested in every aspect that is needed of any IRC client
- Be distributed to various users with various skill levels in computer usage
- Be tested on every OS JarChat is likely to be run on
- Be tested in connections with servers that use SSL encryption

2: Design

Most design will be happening after the core code of the program is written, as UI design takes time which I felt I did not have, due to the complication of the IRC specification.

2.1: UI Design

2.1.1: Start-up Screen

On startup, I intend for JarChat to show the main window where server output will go, with a message in there if the user has not added any IRC servers. Alongside the main text area, there will be two other areas



Figure 3: First (rough) sketch of how the UI may look like.

holding lists of connected servers/channels as well as active users in each channel.

3: Development

The following subsections show parts of the development and why many blocks of code are there.

3.1: Connection to IRC

The actual connection to IRC is going to be handled by code I found on a [GitHub Gist](#) by [Kaecy](#), which seems to handle the connection decently and more stable than others I have found. I have made some edits to the code provided by Kaecy, ensuring it is well-commented, has static methods in line with my main class (called JarChat), has support for secure TLS connections, and has unnecessary parts removed. The code in the provided gist was originally created to be used in a bot, so did not need many of these features I deem required by JarChat.

The IRCMessageLoop class provided by Kaecy is an abstract superclass which JarChat needs to slightly extend on in order to be able to set up any kind of connection. To test the connection, I started to write the program in CLI form, in order to quickly set the server/user information and establish the connection. [Lines 38 and 39](#) of the source code (figure 10) as of the state of the code on 2022-05-02 at 23:17 currently create an instance of an object using the JarChat class, and then starts the connection. Using methods already declared in IRCMessageLoop, I was able to test and validate that the connection to the server was successful in this method.

```
1  /*
2  * JarChat IRC Client Source Code
3  * Free and Open-sourced under the GNU GPL v3 Licence
4  *
5  * Build using the latest JDK 8 to ensure compatibility with all
6  * modern devices. Will change JDK once more devices use JRE 11.
7  *
8  * Last Edited: 2022-05-05 16:47 by SimPilotAdamT
9  */
10
11 package con.AdamT;
12
13 // Imports
14 import java.net.InetAddress;
15 import java.net.UnknownHostException;
16 import java.util.Scanner;
17
18 // Main class
19 public class JarChat extends IRCMessageLoop {
20     JarChat(String server, int port) { super(server, port); }
21 }
```

Figure 4: The beginning of the main class file, showing the constructor of the JarChat class, as well as the necessary libraries being imported.

3.2: The code used in the library

The “library” used, as made by Kaecy, has several parts which have been edited by myself. For example, the extra code between [lines 27 and 35](#) of IRCMessageLoop.java (figure 5) includes a custom implementation of the javax.net.ssl.SSLSocket and javax.net.ssl.SSLSocketFactory classes, in order to allow for most servers’ main connection method. Lines 45 through 57 (Figure 6) were made compact to reduce the overall size of the source code file.

```
27 // Both outcomes of this if statement can throw exceptions, so need to be enclosed in a try-catch statement
28 try {
29     // These ports are exclusively used by encrypted TLS connections
30     if (port == 6697 || port == 7000 || port == 7070) {
31         // Initialize and start the secure socket
32         SSLSocketFactory factory = (SSLSocketFactory)SSLSocketFactory.getDefault();
33         SSLSocket server = (SSLSocket)factory.createSocket(serverName, port);
34         server.setHandshake();
35         // Allow the program to read everything being received from the socket, as well as to send info back to the server
36         stream = server.getInputStream();
37         out = server.getOutputStream();
38     }
39     // Any other ports are going to be plaintext connections, so do not need SSL Sockets
40     else {
41         Socket server = new Socket(serverName, port); // Initialize plaintext connection (this is automatically started)
42         // Allow the program to read everything being received from the socket, as well as to send info back to the server
43         stream = server.getInputStream();
44         out = server.getOutputStream();
45     }
46 } catch (Exception info) { info.printStackTrace(); } // Print information about the error to the terminal for debugging in case it's needed
```

Figure 5: The code which handles the actual Socket which connects to the IRC Server, with my own edits applied.

```
115 // Send the message being sent with a (0, 0) line break and set up a 10 second timeout
116 try {
117     // Send the message
118     out.write(msg);
119     out.flush();
120 } catch (IOException info) { info.printStackTrace(); } // Try to send the message, and print out error info if it fails (without exiting the program)
121
122 // Set the rest of the message line
123 String msg = "END\r\n";
124 out.write(msg);
125 out.flush();
126
127 // Send the message
128 out.write(msg);
129 out.flush();
130
131 // Send the message
132 out.write(msg);
133 out.flush();
134
135 // Send the message
136 out.write(msg);
137 out.flush();
138
139 // Send the message
140 out.write(msg);
141 out.flush();
142
143 // Send the message
144 out.write(msg);
145 out.flush();
146
147 // Send the message
148 out.write(msg);
149 out.flush();
150
151 // Send the message
152 out.write(msg);
153 out.flush();
154
155 // Send the message
156 out.write(msg);
157 out.flush();
158
159 // Send the message
160 out.write(msg);
161 out.flush();
162
163 // Send the message
164 out.write(msg);
165 out.flush();
166
167 // Send the message
168 out.write(msg);
169 out.flush();
170
171 // Send the message
172 out.write(msg);
173 out.flush();
174
175 // Send the message
176 out.write(msg);
177 out.flush();
178
179 // Send the message
180 out.write(msg);
181 out.flush();
182
183 // Send the message
184 out.write(msg);
185 out.flush();
186
187 // Send the message
188 out.write(msg);
189 out.flush();
190
191 // Send the message
192 out.write(msg);
193 out.flush();
194
195 // Send the message
196 out.write(msg);
197 out.flush();
198
199 // Send the message
200 out.write(msg);
201 out.flush();
202
203 // Send the message
204 out.write(msg);
205 out.flush();
206
207 // Send the message
208 out.write(msg);
209 out.flush();
210
211 // Send the message
212 out.write(msg);
213 out.flush();
214
215 // Send the message
216 out.write(msg);
217 out.flush();
218
219 // Send the message
220 out.write(msg);
221 out.flush();
222
223 // Send the message
224 out.write(msg);
225 out.flush();
226
227 // Send the message
228 out.write(msg);
229 out.flush();
230
231 // Send the message
232 out.write(msg);
233 out.flush();
234
235 // Send the message
236 out.write(msg);
237 out.flush();
238
239 // Send the message
240 out.write(msg);
241 out.flush();
242
243 // Send the message
244 out.write(msg);
245 out.flush();
246
247 // Send the message
248 out.write(msg);
249 out.flush();
250
251 // Send the message
252 out.write(msg);
253 out.flush();
254
255 // Send the message
256 out.write(msg);
257 out.flush();
258
259 // Send the message
260 out.write(msg);
261 out.flush();
262
263 // Send the message
264 out.write(msg);
265 out.flush();
266
267 // Send the message
268 out.write(msg);
269 out.flush();
270
271 // Send the message
272 out.write(msg);
273 out.flush();
274
275 // Send the message
276 out.write(msg);
277 out.flush();
278
279 // Send the message
280 out.write(msg);
281 out.flush();
282
283 // Send the message
284 out.write(msg);
285 out.flush();
286
287 // Send the message
288 out.write(msg);
289 out.flush();
290
291 // Send the message
292 out.write(msg);
293 out.flush();
294
295 // Send the message
296 out.write(msg);
297 out.flush();
298
299 // Send the message
300 out.write(msg);
301 out.flush();
302
303 // Send the message
304 out.write(msg);
305 out.flush();
306
307 // Send the message
308 out.write(msg);
309 out.flush();
310
311 // Send the message
312 out.write(msg);
313 out.flush();
314
315 // Send the message
316 out.write(msg);
317 out.flush();
318
319 // Send the message
320 out.write(msg);
321 out.flush();
322
323 // Send the message
324 out.write(msg);
325 out.flush();
326
327 // Send the message
328 out.write(msg);
329 out.flush();
330
331 // Send the message
332 out.write(msg);
333 out.flush();
334
335 // Send the message
336 out.write(msg);
337 out.flush();
338
339 // Send the message
340 out.write(msg);
341 out.flush();
342
343 // Send the message
344 out.write(msg);
345 out.flush();
346
347 // Send the message
348 out.write(msg);
349 out.flush();
350
351 // Send the message
352 out.write(msg);
353 out.flush();
354
355 // Send the message
356 out.write(msg);
357 out.flush();
358
359 // Send the message
360 out.write(msg);
361 out.flush();
362
363 // Send the message
364 out.write(msg);
365 out.flush();
366
367 // Send the message
368 out.write(msg);
369 out.flush();
370
371 // Send the message
372 out.write(msg);
373 out.flush();
374
375 // Send the message
376 out.write(msg);
377 out.flush();
378
379 // Send the message
380 out.write(msg);
381 out.flush();
382
383 // Send the message
384 out.write(msg);
385 out.flush();
386
387 // Send the message
388 out.write(msg);
389 out.flush();
390
391 // Send the message
392 out.write(msg);
393 out.flush();
394
395 // Send the message
396 out.write(msg);
397 out.flush();
398
399 // Send the message
400 out.write(msg);
401 out.flush();
402
403 // Send the message
404 out.write(msg);
405 out.flush();
406
407 // Send the message
408 out.write(msg);
409 out.flush();
410
411 // Send the message
412 out.write(msg);
413 out.flush();
414
415 // Send the message
416 out.write(msg);
417 out.flush();
418
419 // Send the message
420 out.write(msg);
421 out.flush();
422
423 // Send the message
424 out.write(msg);
425 out.flush();
426
427 // Send the message
428 out.write(msg);
429 out.flush();
430
431 // Send the message
432 out.write(msg);
433 out.flush();
434
435 // Send the message
436 out.write(msg);
437 out.flush();
438
439 // Send the message
440 out.write(msg);
441 out.flush();
442
443 // Send the message
444 out.write(msg);
445 out.flush();
446
447 // Send the message
448 out.write(msg);
449 out.flush();
450
451 // Send the message
452 out.write(msg);
453 out.flush();
454
455 // Send the message
456 out.write(msg);
457 out.flush();
458
459 // Send the message
460 out.write(msg);
461 out.flush();
462
463 // Send the message
464 out.write(msg);
465 out.flush();
466
467 // Send the message
468 out.write(msg);
469 out.flush();
470
471 // Send the message
472 out.write(msg);
473 out.flush();
474
475 // Send the message
476 out.write(msg);
477 out.flush();
478
479 // Send the message
480 out.write(msg);
481 out.flush();
482
483 // Send the message
484 out.write(msg);
485 out.flush();
486
487 // Send the message
488 out.write(msg);
489 out.flush();
490
491 // Send the message
492 out.write(msg);
493 out.flush();
494
495 // Send the message
496 out.write(msg);
497 out.flush();
498
499 // Send the message
500 out.write(msg);
501 out.flush();
502
503 // Send the message
504 out.write(msg);
505 out.flush();
506
507 // Send the message
508 out.write(msg);
509 out.flush();
510
511 // Send the message
512 out.write(msg);
513 out.flush();
514
515 // Send the message
516 out.write(msg);
517 out.flush();
518
519 // Send the message
520 out.write(msg);
521 out.flush();
522
523 // Send the message
524 out.write(msg);
525 out.flush();
526
527 // Send the message
528 out.write(msg);
529 out.flush();
530
531 // Send the message
532 out.write(msg);
533 out.flush();
534
535 // Send the message
536 out.write(msg);
537 out.flush();
538
539 // Send the message
540 out.write(msg);
541 out.flush();
542
543 // Send the message
544 out.write(msg);
545 out.flush();
546
547 // Send the message
548 out.write(msg);
549 out.flush();
550
551 // Send the message
552 out.write(msg);
553 out.flush();
554
555 // Send the message
556 out.write(msg);
557 out.flush();
558
559 // Send the message
560 out.write(msg);
561 out.flush();
562
563 // Send the message
564 out.write(msg);
565 out.flush();
566
567 // Send the message
568 out.write(msg);
569 out.flush();
570
571 // Send the message
572 out.write(msg);
573 out.flush();
574
575 // Send the message
576 out.write(msg);
577 out.flush();
578
579 // Send the message
580 out.write(msg);
581 out.flush();
582
583 // Send the message
584 out.write(msg);
585 out.flush();
586
587 // Send the message
588 out.write(msg);
589 out.flush();
590
591 // Send the message
592 out.write(msg);
593 out.flush();
594
595 // Send the message
596 out.write(msg);
597 out.flush();
598
599 // Send the message
600 out.write(msg);
601 out.flush();
602
603 // Send the message
604 out.write(msg);
605 out.flush();
606
607 // Send the message
608 out.write(msg);
609 out.flush();
610
611 // Send the message
612 out.write(msg);
613 out.flush();
614
615 // Send the message
616 out.write(msg);
617 out.flush();
618
619 // Send the message
620 out.write(msg);
621 out.flush();
622
623 // Send the message
624 out.write(msg);
625 out.flush();
626
627 // Send the message
628 out.write(msg);
629 out.flush();
630
631 // Send the message
632 out.write(msg);
633 out.flush();
634
635 // Send the message
636 out.write(msg);
637 out.flush();
638
639 // Send the message
640 out.write(msg);
641 out.flush();
642
643 // Send the message
644 out.write(msg);
645 out.flush();
646
647 // Send the message
648 out.write(msg);
649 out.flush();
650
651 // Send the message
652 out.write(msg);
653 out.flush();
654
655 // Send the message
656 out.write(msg);
657 out.flush();
658
659 // Send the message
660 out.write(msg);
661 out.flush();
662
663 // Send the message
664 out.write(msg);
665 out.flush();
666
667 // Send the message
668 out.write(msg);
669 out.flush();
670
671 // Send the message
672 out.write(msg);
673 out.flush();
674
675 // Send the message
676 out.write(msg);
677 out.flush();
678
679 // Send the message
680 out.write(msg);
681 out.flush();
682
683 // Send the message
684 out.write(msg);
685 out.flush();
686
687 // Send the message
688 out.write(msg);
689 out.flush();
690
691 // Send the message
692 out.write(msg);
693 out.flush();
694
695 // Send the message
696 out.write(msg);
697 out.flush();
698
699 // Send the message
700 out.write(msg);
701 out.flush();
702
703 // Send the message
704 out.write(msg);
705 out.flush();
706
707 // Send the message
708 out.write(msg);
709 out.flush();
710
711 // Send the message
712 out.write(msg);
713 out.flush();
714
715 // Send the message
716 out.write(msg);
717 out.flush();
718
719 // Send the message
720 out.write(msg);
721 out.flush();
722
723 // Send the message
724 out.write(msg);
725 out.flush();
726
727 // Send the message
728 out.write(msg);
729 out.flush();
730
731 // Send the message
732 out.write(msg);
733 out.flush();
734
735 // Send the message
736 out.write(msg);
737 out.flush();
738
739 // Send the message
740 out.write(msg);
741 out.flush();
742
743 // Send the message
744 out.write(msg);
745 out.flush();
746
747 // Send the message
748 out.write(msg);
749 out.flush();
750
751 // Send the message
752 out.write(msg);
753 out.flush();
754
755 // Send the message
756 out.write(msg);
757 out.flush();
758
759 // Send the message
760 out.write(msg);
761 out.flush();
762
763 // Send the message
764 out.write(msg);
765 out.flush();
766
767 // Send the message
768 out.write(msg);
769 out.flush();
770
771 // Send the message
772 out.write(msg);
773 out.flush();
774
775 // Send the message
776 out.write(msg);
777 out.flush();
778
779 // Send the message
780 out.write(msg);
781 out.flush();
782
783 // Send the message
784 out.write(msg);
785 out.flush();
786
787 // Send the message
788 out.write(msg);
789 out.flush();
790
791 // Send the message
792 out.write(msg);
793 out.flush();
794
795 // Send the message
796 out.write(msg);
797 out.flush();
798
799 // Send the message
800 out.write(msg);
801 out.flush();
802
803 // Send the message
804 out.write(msg);
805 out.flush();
806
807 // Send the message
808 out.write(msg);
809 out.flush();
810
811 // Send the message
812 out.write(msg);
813 out.flush();
814
815 // Send the message
816 out.write(msg);
817 out.flush();
818
819 // Send the message
820 out.write(msg);
821 out.flush();
822
823 // Send the message
824 out.write(msg);
825 out.flush();
826
827 // Send the message
828 out.write(msg);
829 out.flush();
830
831 // Send the message
832 out.write(msg);
833 out.flush();
834
835 // Send the message
836 out.write(msg);
837 out.flush();
838
839 // Send the message
840 out.write(msg);
841 out.flush();
842
843 // Send the message
844 out.write(msg);
845 out.flush();
846
847 // Send the message
848 out.write(msg);
849 out.flush();
850
851 // Send the message
852 out.write(msg);
853 out.flush();
854
855 // Send the message
856 out.write(msg);
857 out.flush();
858
859 // Send the message
860 out.write(msg);
861 out.flush();
862
863 // Send the message
864 out.write(msg);
865 out.flush();
866
867 // Send the message
868 out.write(msg);
869 out.flush();
870
871 // Send the message
872 out.write(msg);
873 out.flush();
874
875 // Send the message
876 out.write(msg);
877 out.flush();
878
879 // Send the message
880 out.write(msg);
881 out.flush();
882
883 // Send the message
884 out.write(msg);
885 out.flush();
886
887 // Send the message
888 out.write(msg);
889 out.flush();
890
891 // Send the message
892 out.write(msg);
893 out.flush();
894
895 // Send the message
896 out.write(msg);
897 out.flush();
898
899 // Send the message
900 out.write(msg);
901 out.flush();
902
903 // Send the message
904 out.write(msg);
905 out.flush();
906
907 // Send the message
908 out.write(msg);
909 out.flush();
910
911 // Send the message
912 out.write(msg);
913 out.flush();
914
915 // Send the message
916 out.write(msg);
917 out.flush();
918
919 // Send the message
920 out.write(msg);
921 out.flush();
922
923 // Send the message
924 out.write(msg);
925 out.flush();
926
927 // Send the message
928 out.write(msg);
929 out.flush();
930
931 // Send the message
932 out.write(msg);
933 out.flush();
934
935 // Send the message
936 out.write(msg);
937 out.flush();
938
939 // Send the message
940 out.write(msg);
941 out.flush();
942
943 // Send the message
944 out.write(msg);
945 out.flush();
946
947 // Send the message
948 out.write(msg);
949 out.flush();
950
951 // Send the message
952 out.write(msg);
953 out.flush();
954
955 // Send the message
956 out.write(msg);
957 out.flush();
958
959 // Send the message
960 out.write(msg);
961 out.flush();
962
963 // Send the message
964 out.write(msg);
965 out.flush();
966
967 // Send the message
968 out.write(msg);
969 out.flush();
970
971 // Send the message
972 out.write(msg);
973 out.flush();
974
975 // Send the message
976 out.write(msg);
977 out.flush();
978
979 // Send the message
980 out.write(msg);
981 out.flush();
982
983 // Send the message
984 out.write(msg);
985 out.flush();
986
987 // Send the message
988 out.write(msg);
989 out.flush();
990
991 // Send the message
992 out.write(msg);
993 out.flush();
994
995 // Send the message
996 out.write(msg);
997 out.flush();
998
999 // Send the message
1000 out.write(msg);
1001 out.flush();
1002
1003 // Send the message
1004 out.write(msg);
1005 out.flush();
1006
1007 // Send the message
1008 out.write(msg);
1009 out.flush();
1010
1011 // Send the message
1012 out.write(msg);
1013 out.flush();
1014
1015 // Send the message
1016 out.write(msg);
1017 out.flush();
1018
1019 // Send the message
1020 out.write(msg);
1021 out.flush();
1022
1023 // Send the message
1024 out.write(msg);
1025 out.flush();
1026
1027 // Send the message
1028 out.write(msg);
1029 out.flush();
1030
1031 // Send the message
1032 out.write(msg);
1033 out.flush();
1034
1035 // Send the message
1036 out.write(msg);
1037 out.flush();
1038
1039 // Send the message
1040 out.write(msg);
1041 out.flush();
1042
1043 // Send the message
1044 out.write(msg);
1045 out.flush();
1046
1047 // Send the message
1048 out.write(msg);
1049 out.flush();
1050
1051 // Send the message
1052 out.write(msg);
1053 out.flush();
1054
1055 // Send the message
1056 out.write(msg);
1057 out.flush();
1058
1059 // Send the message
1060 out.write(msg);
1061 out.flush();
1062
1063 // Send the message
1064 out.write(msg);
1065 out.flush();
1066
1067 // Send the message
1068 out.write(msg);
1069 out.flush();
1070
1071 // Send the message
1072 out.write(msg);
1073 out.flush();
1074
1075 // Send the message
1076 out.write(msg);
1077 out.flush();
1078
1079 // Send the message
1080 out.write(msg);
1081 out.flush();
1082
1083 // Send the message
1084 out.write(msg);
1085 out.flush();
1086
1087 // Send the message
1088 out.write(msg);
1089 out.flush();
1090
1091 // Send the message
1092 out.write(msg);
1093 out.flush();
1094
1095 // Send the message
1096 out.write(msg);
1097 out.flush();
1098
1099 // Send the message
1100 out.write(msg);
1101 out.flush();
1102
1103 // Send the message
1104 out.write(msg);
1105 out.flush();
1106
1107 // Send the message
1108 out.write(msg);
1109 out.flush();
1110
1111 // Send the message
1112 out.write(msg);
1113 out.flush();
1114
1115 // Send the message
1116 out.write(msg);
1117 out.flush();
1118
1119 // Send the message
1120 out.write(msg);
1121 out.flush();
1122
1123 // Send the message
1124 out.write(msg);
1125 out.flush();
1126
1127 // Send the message
1128 out.write(msg);
1129 out.flush();
1130
1131 // Send the message
1132 out.write(msg);
1133 out.flush();
1134
1135 // Send the message
1136 out.write(msg);
1137 out.flush();
1138
1139 // Send the message
1140 out.write(msg);
1141 out.flush();
1142
1143 // Send the message
1144 out.write(msg);
1145 out.flush();
1146
1147 // Send the message
1148 out.write(msg);
1149 out.flush();
1150
1151 // Send the message
1152 out.write(msg);
1153 out.flush();
1154
1155 // Send the message
1156 out.write(msg);
1157 out.flush();
1158
1159 // Send the message
1160 out.write(msg);
1161 out.flush();
1162
1163 // Send the message
1164 out.write(msg);
1165 out.flush();
1166
1167 // Send the message
1168 out.write(msg);
1169 out.flush();
1170
1171 // Send the message
1172 out.write(msg);
1173 out.flush();
1174
1175 // Send the message
1176 out.write(msg);
1177 out.flush();
1178
1179 // Send the message
1180 out.write(msg);
1181 out.flush();
1182
1183 // Send the message
1184 out.write(msg);
1185 out.flush();
1186
1187 // Send the message
1188 out.write(msg);
1189 out.flush();
1190
1191 // Send the message
1192 out.write(msg);
1193 out.flush();
1194
1195 // Send the message
1196 out.write(msg);
1197 out.flush();
1198
1199 // Send the message
1200 out.write(msg);
1201 out.flush();
1202
1203 // Send the message
1204 out.write(msg);
1205 out.flush();
1206
1207 // Send the message
1208 out.write(msg);
1209 out.flush();
1210
1211 // Send the message
1212 out.write(msg);
1213 out.flush();
1214
1215 // Send the message
1216 out.write(msg);
1217 out.flush();
1218
1219 // Send the message
1220 out.write(msg);
1221 out.flush();
1222
1223 // Send the message
1224 out.write(msg);
1225 out.flush();
1226
1227 // Send the message
1228 out.write(msg);
1229 out.flush();
1230
1231 // Send the message
1232 out.write(msg);
1233 out.flush();
1234
1235 // Send the message
1236 out.write(msg);
1237 out.flush();
1238
1239 // Send the message
1240 out.write(msg);
1241 out.flush();
1242
1243 // Send the message
1244 out.write(msg);
1245 out.flush();
1246
1247 // Send the message
1248 out.write(msg);
1249 out.flush();
1250
1251 // Send the message
1252 out.write(msg);
1253 out.flush();
1254
1255 // Send the message
1256 out.write(msg);
1257 out.flush();
1258
1259 // Send the message
1260 out.write(msg);
1261 out.flush();
1262
1263 // Send the message
1264 out.write(msg);
1265 out.flush();
1266
1267 // Send the message
1268 out.write(msg);
1269 out.flush();
1270
1271 // Send the message
1272 out.write(msg);
1273 out.flush();
1274
1275 // Send the message
1276 out.write(msg);
1277 out.flush();
1278
1279 // Send the message
1280 out.write(msg);
1281 out.flush();
1282
1283 // Send the message
1284 out.write(msg);
1285 out.flush();
1286
1287 // Send the message
1288 out.write(msg);
1289 out.flush();
1290
1291 // Send the message
1292 out.write(msg);
1293 out.flush();
1294
1295 // Send the message
1296 out.write(msg);
1297 out.flush();
1298
1299 // Send the message
1300 out.write(msg);
1301 out.flush();
1302
1303 // Send the message
1304 out.write(msg);
1305 out.flush();
1306
1307 // Send the message
1308 out.write(msg);
1309 out.flush();
1310
1311 // Send the message
1312 out.write(msg);
1313 out.flush();
1314
1315 // Send the message
1316 out.write(msg);
1317 out.flush();
1318
1319 // Send the message
1320 out.write(msg);
1321 out.flush();
1322
1323 // Send the message
1324 out.write(msg);
1325 out.flush();
1326
1327 // Send the message
1328 out.write(msg);
1329 out.flush();
1330
1331 // Send the message
1332 out.write(msg);
1333 out.flush();
1334
1335 // Send the message
1336 out.write(msg);
1337 out.flush();
1338
1339 // Send the message
1340 out.write(msg);
1341 out.flush();
1342
1343 // Send the message
1344 out.write(msg);
1345 out.flush();
1346
1347 // Send the message
1348 out.write(msg);
1349 out.flush();
1350
1351 // Send the message
1352 out.write(msg);
1353 out.flush();
1354
1355 // Send the message
1356 out.write(msg);
1357 out.flush();
1358
1359 // Send the message
1360 out.write(msg);
1361 out.flush();
1362
1363 // Send the message
1364 out.write(msg);
1365 out.flush();
1366
1367 // Send the message
1368 out.write(msg);
1369 out.flush();
1370
1371 // Send the message
1372 out.write(msg);
1373 out.flush();
1374
1375 // Send the message
1376 out.write(msg);
1377 out.flush();
1378
1379 // Send the message
1380 out.write(msg);
1381 out.flush();
1382
1383 // Send the message
1384 out.write(msg);
1385 out.flush();
1386
1387 // Send the message
1388 out.write(msg);
1389 out.flush();
1390
1391 // Send the message
1392 out.write(msg);
1393 out.flush();
1394
1395 // Send the message
1396 out.write(msg);
1397 out.flush();
1398
1399 // Send the message
1400 out.write(msg);
1401 out.flush();
1402
1403 // Send the message
1404 out.write(msg);
1405 out.flush();
1406
1407 // Send the message
1408 out.write(msg);
1409 out.flush();
1410
1411 // Send the message
1412 out.write(msg);
1413 out.flush();
1414
1415 // Send the message
1416 out.write(msg);
1417 out.flush();
1418
1419 // Send the message
1420 out.write(msg);
1421 out.flush();
1422
1423 // Send the message
1424 out.write(msg);
1425 out.flush();
1426
1427 // Send the message
1428 out.write(msg);
1429 out.flush();
1430
1431 // Send the message
1432 out.write(msg);
1433 out.flush();
1434
1435 // Send the message
1436 out.write(msg);
1437 out.flush();
1438
1439 // Send the message
1440 out.write(msg);
1441 out.flush();
1442
1443 // Send the message
1444 out.write(msg);
1445 out.flush();
1446
1447 // Send the message
1448 out.write(msg);
1449 out.flush();
1450
1451 // Send the message
1452 out.write(msg);
1453 out.flush();
1454
1455 // Send the message
1456 out.write(msg);
1457 out.flush();
1458
1459 // Send the message
1460 out.write(msg);
1461 out.flush();
1462
1463 // Send the message
1464 out.write(msg);
1465 out.flush();
1466
1467 // Send the message
1468 out.write(msg);
1469 out.flush();
1470
1471 // Send the message
1472 out.write(msg);
1473 out.flush();
1474
1475 // Send the message
1476 out.write(msg);
1477 out.flush();
1478
1479 // Send the message
1480 out.write(msg);
1481 out.flush();
1482
1483 // Send the message
1484 out.write(msg);
1485 out.flush();
1486
1487 // Send the message
1488 out.write(msg);
1489 out.flush();
1490
1491 // Send the message
1492 out.write(msg);
1493 out.flush();
1494
1495 // Send the message
1496 out.write(msg);
1497 out.flush();
1498
1499 // Send the message
1500 out.write(msg);
1501 out.flush();
1502
1503 // Send the message
1504 out.write(msg);
1505 out.flush();
1506
1507 // Send the message
1508 out.write(msg);
1509 out.flush();
1510
1511 // Send the message
1512 out.write(msg);
1513 out.flush();
1514
1515 // Send the message
1516 out.write(msg);
1517 out.flush();
1518
1519 // Send the message
1520 out.write(msg);
1521 out.flush();
1522
1523 // Send the message
1524 out.write(msg);
1525 out.flush();
1526
1527 // Send the message
1528 out.write(msg);
1529 out.flush();
1530
1531 // Send the message
1532 out.write(msg);
1533 out.flush();
1534
1535 // Send the message
1536 out.write(msg);
1537 out.flush();
1538
1539 // Send the message
1540 out.write(msg);
1541 out.flush();
1542
1543 // Send the message
1544 out.write(msg);
1545 out.flush();
1546
1547 // Send the message
1548 out.write(msg);
1549 out.flush();
1550
1551 // Send the message
1552 out.write(msg);
1553 out.flush();
1554
1555 // Send the message
1556 out.write(msg);
1557 out.flush();
1558
1559 // Send the message
1560 out.write(msg);
1561 out.flush();
1562
1563 // Send the message
1564 out.write(msg);
1565 out.flush();
1566
1567 // Send the message
1568 out.write(msg);
1569 out.flush();
1570
1571 // Send the message
1572 out.write(msg);
1573 out.flush();
1574
1575 // Send the message
1576 out.write(msg);
1577 out.flush();
1578
1579 // Send the message
1580 out.write(msg);
1581 out.flush();
1582
1583 // Send the message
1584 out.write(msg);
1585 out.flush();
1586
1587 // Send the message
1588 out.write(msg);
1589 out.flush();
1590
1591 // Send the message
1592 out.write(msg);
1593 out.flush();
1594
1595 // Send the message
1596 out.write(msg);
1597 out.flush();
1598
1599 // Send the message
1600 out.write(msg);
1601 out.flush();
1602
1603 // Send the message
1604 out.write(msg);
1605 out.flush();
1606
1607 // Send the message
1608 out.write(msg);
1609 out.flush();
1610
161
```


The [Message and MessageBuffer classes](#) (figure 8) are used so that there is a buffer to store messages in and to ensure that each message can be stored in its own object, setting each attribute. No constructor method is needed in the Message class because Java automatically handles this on creation of each object of type Message.

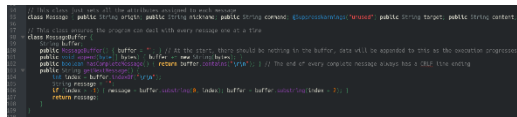


Figure 8: The Message and MessageBuffer classes.

The [MessageParser class](#) (figure 9, called by IRCMessageLoop.processMessage()) parses each message received from the server into a way better readable by IRCMessageLoop.processMessage(). It also slightly beautifies each message for when it gets put into STDOUT.

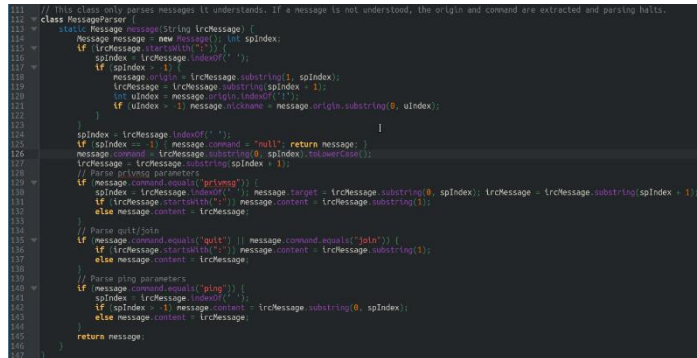


Figure 9: The MessageParser class.

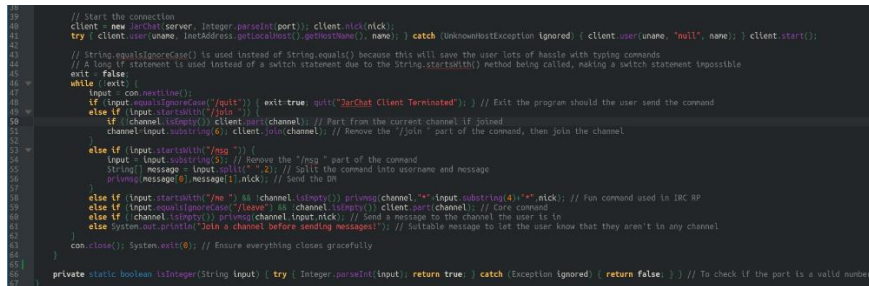


Figure 10: The part of the JarChat class which handles the connection and the user's inputs to STDIN

sent to STDIN, the long if statement (used instead of a switch statement due to a [quirk in the way switch statements work](#)) checks for valid messages or commands, with an appropriate error message for anything invalid. So far everything can only take place in a single IRC text channel. The user is able to send direct messages to several users at a time, but can only be connected to a single channel at any given time. I hope to be able to make use of the Thread class within a javax.swing GUI in order to allow for the user to send/receive messages to/from multiple channels at once.

[Lines 41 thru 60](#) of my code (figure 10) as of 2022-05-02 at 23:23 show a while loop which keeps going unless the program is exited (either by [issuing a /quit command to STDIN](#), or by [running into some kind of connection error](#)). For every thing

4: Testing, Review and Evaluation

In this section I will define the criteria I have achieved, alongside proof of this, limitations of JarChat in its current form, and how it can/will be maintained.

4.1: Achieved Success Criteria

So far, as of 2022-05-05 at 22:30, JarChat has achieved the following:

- Function as at least a semi-workable IRC Client
- Ability to use SSL encrypted connections
- Ability to be run on any given OS that has a JVM
- Ability to chat with multiple users directly at the same time

4.1.1: Proof of achieved Success Criteria

I have been able to connect to Libera.Chat's IRC server using JarChat in CLI (figures 11-12). The test shown here uses the TLS encrypted connection method and shows its success.

Figure 11: Collecting server/user info and connecting to the server. An IP address is shown but it is a mobile network one, not a DSL Broadband one.

Figure 12 shows the rest of the successful connection, with the login as managed by NickServ (a popular login bot used in IRC servers), with the password censored out.

Figure 12: Successful connection, with login. Password is censored for security reasons.

Figures 13 & 14 show several messages being sent back and forth between my account on Jar Chat and my main account. Included there is a use of the /me command, which seems to fully work as intended.

Figures 13 & 14: Messages being sent back and forth in the test channel I created on Libera.Chat

4.2: Limitations

The most important parts of this project that are so far missing are the following:

- OP/Administrative commands
- javax.swing GUI
- Ability to join multiple channels
- Ability to join multiple servers

The lack of a GUI is a major limitation, as this is one of the goals I had in mind when I first thought of JarChat. This significantly reduces ease of use as JarChat in its current state expects the user to be able to open the command line to run the appropriate command.

The lack of OP/Administrator commands is also a fairly major limitation, since part of the IRC specification is being able to use these commands to moderate chats in channels where you have the rights to. This also allows server administrators to do everything they may need to do when signed in to the server to keep it running smoothly.

4.2.1: Avoiding these limitations & Maintenance

There is no surefire way accessible to the user to avoid them. JarChat is open-sourced and hosted on GitHub (<https://github.com/SimPilotAdamT/JarChat/>), under the GNU GPLv3 license, so anyone can have a look at the code.

The codebase for the project itself is very modular. In order to add a GUI all that there is needed to do is import the swing classes and build the GUI by hand. This will take tremendous amounts of time, and would ideally need a large team of people working on the project together, which is something which can be seen with Konversation (discussed in section 1.3.1.2.1), as well as HexChat (discussed in section 1.3.1.2.3).

Future maintenance should be simple to carry on, even if I abandon the project. The GPLv3 license and GitHub allow for other users to fork the project and make updates to the codebase to keep it up-to-date, fix any bugs that slip through the cracks, and add any new features. The codebase has been divided into two parts, each with their own source code file (see section 5).

I have annotated the code fairly well and anything that has no annotation is code which I believe is self-explanatory. I have kept all subroutine and variable names reasonable so anybody new looking at the codebase who understands Java's syntax should be able to decipher how the code functions.

Future versions will most likely finally incorporate the GUI I originally set out to create, but will likely incorporate any missing commands first. I do intend to keep maintaining this project for as long as I can, so all references to code in this document which have links to code in the GitHub repository by commit, so that references in here stay valid.

5: Final Code

5.1: IRCMessageLoop.java

```
1 // All of these classes are taken from Kaecy's gist at https://gist.github.com/kaecy/286f8ad334aec3fcb588516feb727772,
2 // with my own edits to ensure they better suited for use as an actual client, as well as to add comments to the code.
3
4 package com.AdamT;
5
6 // Imports
7 import com.sun.istack.internal.Nullable;
8 import java.io.IOException;
9 import java.io.InputStream;
10 import java.io.OutputStream;
11 import java.net.Socket;
12 import javax.net.ssl.SSLSocket;
13 import javax.net.ssl.SSLSocketFactory;
14 import java.util.ArrayList;
15 import java.util.Arrays;
16
17 abstract class IRCMessageLoop extends Thread {
18     // Variables (local to the class as many methods require them)
19     static OutputStream out;
20     ArrayList<String> channellist = new ArrayList<>();
21     boolean initial_setup_status;
22     InputStream stream;
23     IRCMessageLoop(String serverName, int port) {
24         // Both outcomes of this if statement can throw exceptions, so need to be encased in a try-catch statement
25         try {
26             // These ports are exclusively used by encrypted TLS connections
27             if (port == 6697 || port == 7000 || port == 7070) {
28                 // Initialise and start the secure socket
29                 SSLSocketFactory factory = (SSLSocketFactory)SSLSocketFactory.getDefault();
30                 SSLSocket server = (SSLSocket)factory.createSocket(serverName, port);
31                 server.startHandshake();
32                 // Allow the program to read everything being received from the socket, as well as to send info back to the server
33                 out = server.getOutputStream();
34                 stream = server.getInputStream();
35             }
36             // Any other ports are going to be plaintext connections, so do not need SSL Sockets
37             else {
38                 Socket server = new Socket(serverName, port); // Initialise plaintext connection (this is automatically started)
39                 // Allow the program to read everything being received from the socket, as well as to send info back to the server
40                 out = server.getOutputStream();
41                 stream = server.getInputStream();
42             }
43         }
44     }
45 }
```

~\JarChat\Client\src\com\AdamT\IRCMessageLoop.java [FORMAT=dos] [TYPE=JAVA] [POS=1,1][0%] [BUFFER=1] 06/05/2022 17:29:23

```

43     } catch (Exception info) { info.printStackTrace(); } // Print information about the error to the terminal for debugging in case it's needed
44 }
45 static void send(String text) {
46     byte[] bytes = (text + "\r\n").getBytes(); // Ensure the message being sent ends with a CRLF line break and not a LF line break
47     try { out.write(bytes); } catch (IOException info) { info.printStackTrace(); } // Try to send the message, and print out error info if it fails (without exiting the program)
48 }
49 void nick(String nickname) { String msg = "NICK " + nickname; send(msg); } // Set the nickname as seen by the server and other users
50 // Set the rest of the user's info
51 void user(String username, String hostname, String real_name) { String msg = "USER " + username + " " + hostname + " " + "null" + " " + real_name; send(msg); }
52 void join(String channel) { if (!initial_setup_status) { channelList.add(channel); return; } String msg = "JOIN " + channel; send(msg); } // Join the channel as requested by the user
53 void part(String channel) { String msg = "PART " + channel; send(msg); } // Leave the channel as requested by the user, without disconnecting from the server
54 // Send messages, either directly or as a DM
55 static void privmsg(String to, String text, @Nullable String from) { String msg = "PRIVMSG " + to + " : " + text; send(msg); System.out.println("PRIVMSG: " + from + ": " + text); }
56 void pong(String server) { String msg = "PONG " + server; send(msg); } // Respond back to the server every few minutes to ensure the connection isn't forcibly removed
57 static void quit(String reason) { String msg = "QUIT :Quit: " + reason; send(msg); } // Disconnect from the server as requested by the user
58 void initial_setup() {
59     initial_setup_status = true;
60     for (String channel: channelList) { join(channel); } // Now you can join the channels. You need to wait for message 001 before you join a channel.
61 }
62 // Method to call the other methods here as required depending on the message received from the server or other users.
63 void processMessage(String ircMessage) {
64     Message msg = MessageParser.message(ircMessage);
65     switch (msg.command) {
66         case "privmsg": if (msg.content.equals("\001VERSION\001")) { privmsg(msg.nickname, "JarChat", null); return; } // Reflect this client's name in the response
67         case "ping": System.out.println("PRIVMSG: " + msg.nickname + ": " + msg.content); break; // Show the received message in the console log
68         case "001": initial_setup(); break; // Initial message as sent by the server
69         case "ping": pong(msg.content); break; // see comment on line 53
70     }
71 }
72
73 public void run() {
74     try {
75         MessageBuffer messageBuffer = new MessageBuffer();
76         byte[] buffer = new byte[512];
77         int count;
78         while (true) {
79             count = stream.read(buffer);
80             if (count == -1) break; // There is nothing being sent by the server, so no need to continue the loop
81             messageBuffer.append(Arrays.copyOfRange(buffer, 0, count));
82             // Process every complete IRC message in the message buffer
83             while (messageBuffer.hasCompleteMessage()) {
84                 String ircMessage = messageBuffer.getNextMessage();
85                 System.out.println("\n" + ircMessage + "\n");
86                 processMessage(ircMessage);
87             }
88         }
89     } catch (IOException info) { quit("error in IRCMessageLoop"); info.printStackTrace(); System.exit(1); } // We cannot continue execution, so we disconnect and halt execution
90 }
91 }
92 }
93
94 // This class just sets all the attributes assigned to each message
95 class Message { public String origin; public String nickname; public String command; @SuppressWarnings("unused") public String target; public String content; }
96
97 // This class ensures the program can deal with every message one at a time
98 class MessageBuffer {
99     String buffer;
100     public MessageBuffer() { buffer = ""; } // At the start, there should be nothing in the buffer, data will be appended to this as the execution progresses
101     public void append(byte[] bytes) { buffer += new String(bytes); }
102     public boolean hasCompleteMessage() { return buffer.contains("\r\n"); } // The end of every complete message always has a CRLF line ending
103     public String getNextMessage() {
104         int index = buffer.indexOf("\r\n");
105         String message = "";
106         if (index > -1) { message = buffer.substring(0, index); buffer = buffer.substring(index + 2); }
107         return message;
108     }
109 }
110
111 // This class only parses messages it understands. If a message is not understood, the origin and command are extracted and parsing halts.
112 class MessageParser {
113     static Message message(String ircMessage) {
114         Message msg = new Message(); int spIndex;
115         if (ircMessage.startsWith(":")) {
116             spIndex = ircMessage.indexOf(' ');
117             if (spIndex > -1) {
118                 msg.origin = ircMessage.substring(1, spIndex);
119                 ircMessage = ircMessage.substring(spIndex + 1);
120                 int uIndex = message.origin.indexOf(':');
121                 if (uIndex > -1) msg.nickname = message.origin.substring(0, uIndex);
122             }
123         }
124     }
125 }

```

```

106     if (index > -1) { message = buffer.substring(0, index); buffer = buffer.substring(index + 2); }
107     return message;
108 }
109 }
110
111 // This class only parses messages it understands. If a message is not understood, the origin and command are extracted and parsing halts.
112 class MessageParser {
113     static Message message(String ircMessage) {
114         Message message = new Message(); int spIndex;
115         if (ircMessage.startsWith(":")) {
116             spIndex = ircMessage.indexOf(' ');
117             if (spIndex > -1) {
118                 message.origin = ircMessage.substring(1, spIndex);
119                 ircMessage = ircMessage.substring(spIndex + 1);
120                 int uIndex = message.origin.indexOf('!');
121                 if (uIndex > -1) message.nickname = message.origin.substring(0, uIndex);
122             }
123         }
124         spIndex = ircMessage.indexOf(' ');
125         if (spIndex == -1) { message.command = "null"; return message; }
126         message.command = ircMessage.substring(0, spIndex).toLowerCase();
127         ircMessage = ircMessage.substring(spIndex + 1);
128         // Parse privmsg parameters
129         if (message.command.equals("privmsg")) {
130             spIndex = ircMessage.indexOf(' '); message.target = ircMessage.substring(0, spIndex); ircMessage = ircMessage.substring(spIndex + 1);
131             if (ircMessage.startsWith(":")) message.content = ircMessage.substring(1);
132             else message.content = ircMessage;
133         }
134         // Parse quit/join
135         if (message.command.equals("quit") || message.command.equals("join")) {
136             if (ircMessage.startsWith(":")) message.content = ircMessage.substring(1);
137             else message.content = ircMessage;
138         }
139         // Parse ping parameters
140         if (message.command.equals("ping")) {
141             spIndex = ircMessage.indexOf(' ');
142             if (spIndex > -1) message.content = ircMessage.substring(0, spIndex);
143             else message.content = ircMessage;
144         }
145         return message;
146     }
147 }

```

~\JarChat\Client\src\com\AdamT\IRCMessageLoop.java [FORMAT=dos] [TYPE=JAVA] [POS=146,1][99%] [BUFFER=1] 06/05/2022 17:33:23

5.2: JarChat.java

```

1 /*
2  * JarChat IRC Client Source Code
3  * Free and Open-sourced under the GNU GPL v3 Licence
4  *
5  * Build using the latest JDK 8 to ensure compatibility with all
6  * modern devices. Will change JDK once more devices use JRE 11.
7  *
8  * Last Edited: 2022-05-05 16:47 by SimPilotAdamT
9  */
10
11 package com.AdamT;
12
13 // Imports
14 import java.net.InetAddress;
15 import java.net.UnknownHostException;
16 import java.util.Scanner;
17
18 // Main class
19 public class JarChat extends IRCMessageLoop {
20
21     JarChat(String server, int port) { super(server, port); }
22
23     public static void main(String[] args) {
24         // Variables (local to this method instead of the class in case any variables with these names or similar are required elsewhere)
25         boolean exit; String input; String channel = ""; JarChat client; boolean valid; Scanner con; String server; String port; String nick; String uname; String name;
26
27         // Welcome and entering server details
28         System.out.println("\nHi!"); con = new Scanner(System.in); System.out.print("\nEnter server IP/Hostname: ");
29         server = con.nextLine(); System.out.print("Enter server port: "); port = con.nextLine();
30
31         // Check the port is valid (most servers use a port with 4 digits
32         valid = false;
33         while(!valid) { if (isInteger(port) && port.length() == 4) valid=true; else { System.out.print("Error! Invalid port!\nEnter server port: "); port=con.nextLine(); } }
34
35         // Get user's details
36         System.out.print("\nEnter nickname: "); nick = con.nextLine(); System.out.print("Enter username (most of the time this is the same as the nickname): ");
37         uname = con.nextLine(); System.out.print("Enter real name: "); name = con.nextLine(); System.out.print("\n");
38
39         // Start the connection
40         client = new JarChat(server, Integer.parseInt(port)); client.nick(nick);
41         try { client.user(uname, InetAddress.getLocalHost().getHostName(), name); } catch (UnknownHostException ignored) { client.user(uname, "null", name); } client.start();
42     }

```

~\JarChat\Client\src\com\AdamT\JarChat.java [FORMAT=dos] [TYPE=JAVA] [POS=1,1][1%] [BUFFER=1] 06/05/2022 17:35:24

```
27 // Welcome and entering server details
28 System.out.println("\nHi!"); con = new Scanner(System.in); System.out.print("\nEnter server IP/Hostname: ");
29 server = con.nextLine(); System.out.print("Enter server port: "); port = con.nextLine();
30
31 // Check the port is valid (most servers use a port with 4 digits
32 valid = false;
33 while(!valid) { if (isInteger(port) && port.length() == 4) valid=true; else { System.out.print("Error! Invalid port!\nEnter server port: "); port=con.nextLine(); } }
34
35 // Get user's details
36 System.out.print("\nEnter nickname: "); nick = con.nextLine(); System.out.print("Enter username (most of the time this is the same as the nickname): ");
37 uname = con.nextLine(); System.out.print("Enter real name: "); name = con.nextLine(); System.out.print("\n");
38
39 // Start the connection
40 client = new JarChat(server, Integer.parseInt(port)); client.nick(nick);
41 try { client.user(uname, InetAddress.getLocalHost().getHostName(), name); } catch (UnknownHostException ignored) { client.user(uname, "null", name); } client.start();
42
43 // String.equalsIgnoreCase() is used instead of String.equals() because this will save the user lots of hassle with typing commands
44 // A long if statement is used instead of a switch statement due to the String.startsWith() method being called, making a switch statement impossible
45 exit = false;
46 while (!exit) {
47     input = con.nextLine();
48     if (input.equalsIgnoreCase("/quit")) { exit=true; quit("JarChat Client Terminated"); } // Exit the program should the user send the command
49     else if (input.startsWith("/join ")) {
50         if (!channel.isEmpty()) client.part(channel); // Part from the current channel if joined
51         channel=input.substring(6); client.join(channel); // Remove the "/join " part of the command, then join the channel
52     }
53     else if (input.startsWith("/msg ")) {
54         input = input.substring(5); // Remove the "/msg " part of the command
55         String[] message = input.split(" "); // Split the command into username and message
56         privmsg(message[0],message[1],nick); // Send the DM
57     }
58     else if (input.startsWith("/me ") && !channel.isEmpty()) privmsg(channel,""+input.substring(4)+"",nick); // Fun command used in IRC RP
59     else if (input.equalsIgnoreCase("/leave") && !channel.isEmpty()) client.part(channel); // Core command
60     else if (!channel.isEmpty()) privmsg(channel,input,nick); // Send a message to the channel the user is in
61     else System.out.println("Join a channel before sending messages!"); // Suitable message to let the user know that they aren't in any channel
62 }
63 con.close(); System.exit(0); // Ensure everything closes gracefully
64 }
65
66 private static boolean isInteger(String input) { try { Integer.parseInt(input); return true; } catch (Exception ignored) { return false; } } // To check if the port is a valid number
67 }
```

"\JarChat\client\src\com\AdamT\JarChat.java [FORMAT=dos] [TYPE=JAVA] [POS=55,1][82%] [BUFFER=1] 06/05/2022 17:36:39