** Noted that Access Modifier Notations can be listed below*

+ (public)
# (protected)
- (private)
Underline (static)
Italic (abstract)

Set-Up Instruction

- Don't forget to set your Eclipse workspace and working set.
- You must submit the JAR file, exported (with source code), from your Eclipse project.
- You must check your JAR file to make sure all the source files (.java files) are present. It can be opened with file compression programs such as 7-zip or Winrar.
- Failure to export properly will result in your work not getting marked.

Question 2 (expected time 60 minutes maximum)

1. Instruction

1) Create a Gradle Project.
2) Copy all folders in folder "Q2_toStudent" to your usual Gradle source code folder.
3) You are to implement the following classes (the details for each class are given in section 4). Make UML of these classes too. Missing UML will cap your score to only 80%.
   3.1. **Server (package logic)**
   3.2. **Channel (package logic)**
4) JUnit for testing is in package **test.student. Put them in a correct Gradle test folder.**
5) **To submit:**
   5.1. **Export your project to a JAR file, with source code.**
   5.2. **Submit the JAR file on MyCourseville.**

2. Problem Statement: Simple Discord

Let's make a simple text-based Discord.

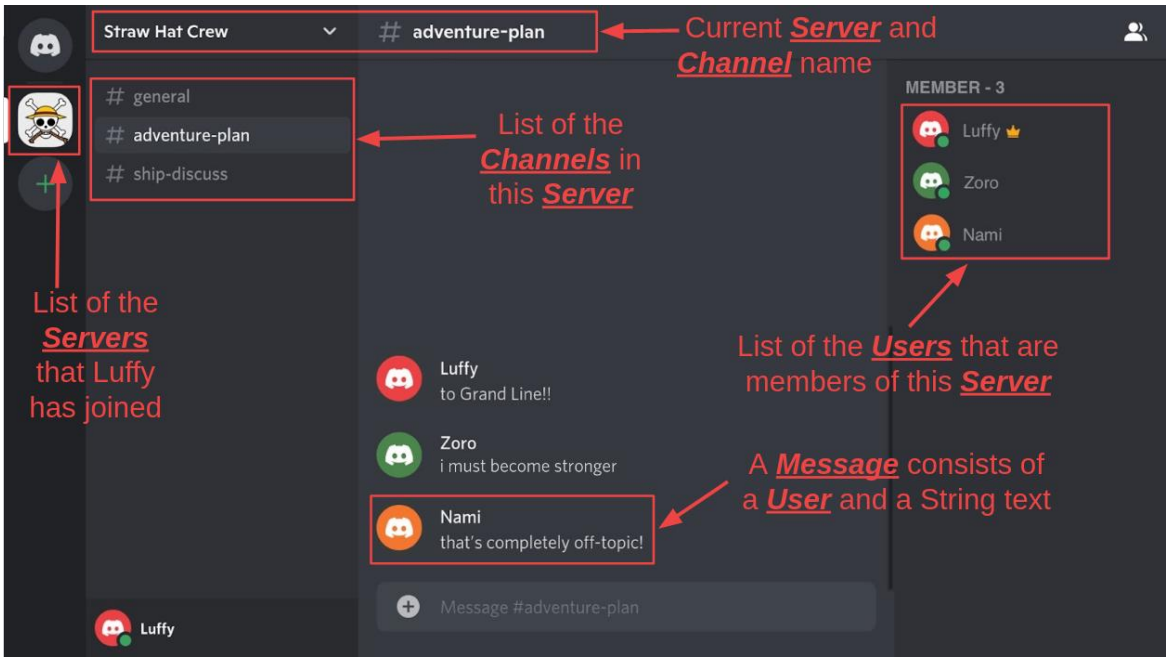

Figure 1: Parts of Discord with their names (Note that the program for this task is 100% text-based. This figure is only to help you visualize and confirm the terminology.)

Below are the Discord terms for this problem.

1) Server: A place in Discord for **Users** with the same interest to hang out. In Figure 1, the current **Server** being shown is Straw Hat Crew. The **User** that created the **Server** is the **owner** of that **Server**.

2) Channel: Inside a **Server** are **Channels**. This is where **Users** send **Messages**. For example, in Figure 1, **general**, **adventure-plan** and **ship-discuss** are **Channels**. Only the **owner** can add a new **Channel**.

3) User: Each **User** represents a person. **Users** can join other **Users**' **Server** or create their own **Servers**. Once they are a member of a **Server**, they can send **Messages** in a **Channel** in that **Server**. The **owner** is also a member of the **Server**.

4) Message: Represents a message. It has two components which are 1) the text itself (a String), and 2) the **User** who sends it. Each **Message** belongs to a **Channel**.

5) TemplateType: Determines the starting **Channel** at the creation of a **Server**. For this task, there are 3 **TemplateTypes**. (See part 3) for the complete implementation details and part5) figure 8 for a demonstration.) For example, if a new **Server** called "Our Little School" is being created with the **TemplateType** STUDY, then a **Channel** called "homework-help" will be generated in "Our Little School" **Server**.

Your task is to implement 2 classes from scratch: Server and Channel. The User, Message and TemplateType classes are already provided.

3. Implementation Detail

The two classes are summarized below.

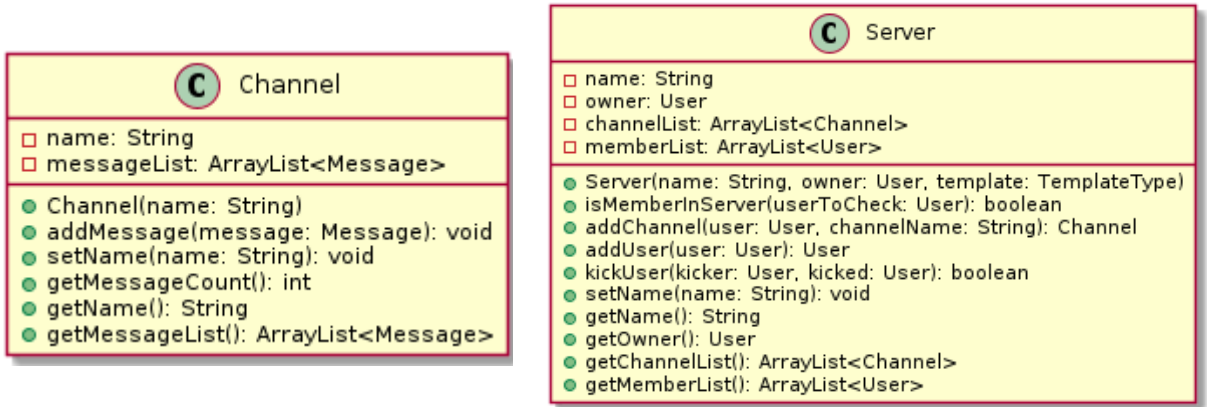

Figure 2. Class Diagram

You must write java classes using the UML diagram specified above.
*** In the following classes description, only details of IMPORTANT fields and methods are given. ***

3.1 Package logic

3.1.1 **Enum TemplateType:** Determines the starting Channel. It is an enumeration (a list of constants) that specifies a Server's starting Channel at the creation of a Server. (See Server's constructor)

/*ALREADY PROVIDED*/

Values

Name	Description
BASIC	Servers instantiated with BASIC TemplateType have one starting Channel called “general”.
GAMING	Servers instantiated with GAMING TemplateType have one starting Channel called “gaming”.
STUDY	Servers instantiated with STUDY TemplateType have one starting Channel called “homework-help”.

3.1.2 **Class Message:** This class represents messages that are sent in Channel.

/*ALREADY PROVIDED*/

Fields

Name	Description
- String text	The text to be sent.
- User sender	The sender of this message.

Constructor

Name	Description
+ Message(String text, User sender)	Create a new Message with the specified information.

Methods

Name	Description
+ getters/setters for each field	

3.1.3 **Class User:** This class represents users in our Discord system.

/*ALREADY PROVIDED*/

Fields

Name	Description

- String name	The name of the user.
- String status	The status of the user.
- ArrayList<Server> joinedServerList	An ArrayList containing Servers that this user has joined, including those that this user is the owner of.

Constructor

Name	Description
+ User(String name)	Create a new User with the specified name. Set the status to “hi there, I’m new here”. Initialized the joinedServerList.

Methods

Name	Description
+ boolean equals(Object other)	If the User other is not null, check whether that User is equal to this User by comparing the name of the Users. Otherwise return false.
+ void addJoinedServersList(Server server)	If server is null, do nothing. Otherwise add the server to this User’s joinedServersList.
+ void setStatus(String status)	If the specified status is not blank, set this User’s status to that status. Otherwise do nothing (the status is unchanged).
+ void setName(String name)	If the specified name is not blank, set this User’s name to that name. Otherwise set the User’s name to “Nobody”.
+ getters for name, status, joinedServersList	

3.1.4 **Class Channel:** This class represents channels in a Discord Server. It contains the channel's name and all the Messages that are sent in the channel.

/* You must create this class from scratch. */

Fields

Name	Description
- String name	The name of the channel.
- ArrayList<Message> messageList	An ArrayList containing all the Messages sent in the channel.

Constructor

Name	Description
+ Channel(String name)	Create a Channel with the specified name. Do not forget to initialize the messageList.

Methods

Name	Description
+ void addMessage(Message message)	Add the message to the messageList.
+ void setName(String name)	Set the name of this Channel. If the specified name is a blank string, set this Channel's name to "off-topic". Otherwise set it to that name. **Hint**: use name.isBlank().
+ int getMessageCount()	Return the size of the messageList.
+ remaining getters	Write code for the rest of the getters.

3.1.5 **Class Server:** This class represents the servers in Discord. It contains data related to the Server, including a list of channels within that server and a list of Users that are members.

/* You must create this class from scratch. */

Fields

Name	Description
- String name	The name of the Server.
- User owner	The owner of the Server.
- ArrayList<Channel> channelList	An ArrayList containing all the Channels in the Server.
- ArrayList<User> memberList	An ArrayList containing all the members of the Server, including the owner.

Constructor

Name	Description
+ Server(String name, User owner, TemplateType template)	(**Note: The order of initializations is important!!**) Set the owner of the Server to the specified owner. Initialize the memberList. Initialize the channelList. Add the owner to the memberList and add this Server to the owner's joinedServersList. (**Hint**: use the addUser method of this class) Set the name of the server. Then add 1 Channel to the channelList depending on the TemplateType specified. If the template is 1) BASIC: create and add a Channel called "general" 2) GAMING: create and add a Channel called "gaming" 3) STUDY: create and add a Channel called "homework-help" (**Hint**: Use the **addChannel** method of this class, and pass

	the owner of the Server as one of the arguments.)

Methods

Name	Description
+ Channel addChannel(User user, String channelName)	If the user in the argument is the owner of the Server, create a new Channel with the specified channelName. Then add it to the channelList and return the added Channel. Otherwise, if the user is not the owner of the server, return null and do nothing else.
+ User addUser(User user)	There are 2 cases. 1) If the user is not already in the memberList, add the user to the memberList. **Hint**: use the *contains* method of ArrayList to check. Also, add this server to the user's joinedServersList (See 3.1.3 Class User for details), then return the newly added user. 2) If the user is already in the memberList, return null and do nothing else.
+ boolean kickUser(User kicker, User **kicked**) throws Exception	There are three cases. 1) If the kicker is not the Server's owner: Throw an Exception and do nothing else. 2) Else If the kicker is the owner and the user to be kicked is either **i)** not in the memberList or **ii)** is the Server's owner himself/herself: Return false and do nothing else. (Owner can't be removed):

	3) Else: Remove this Server from **kicked**'s joinedServersList. Then, remove **kicked** from the Server's memberList and return true. (Note: there are 2 variants of the ArrayList method *remove.* Make sure you're using the one that returns boolean and not the polymorphic one)
+ void setName(String name)	If the name is blank, set the Server's name to <owner's name> + " home". (For example, "Citron home") Otherwise, set it to the specified name
+ remaining getter for each field	

3.2 Package main **/*ALREADY PROVIDED */**

3.2.1 **Class Main**: This class is the main application. You do not need to know any details about this class to be able to complete the problem. Its purpose is for the program demonstration (see section 5).

4. Score Criteria (30 marks)

Make UML diagrams of Server and Channel. Missing UML will cap your score to only 80%.

ChannelTest

- testConstructor (1 mark)
- testConstructorEmptyName (1 mark)
- testSetName (1 mark)
- testSetNameEmpty (1 mark)
- testAddMessage (1 mark)
- testGetMessageCount (1 mark)

ServerTest

- testConstructorBasicTemplate (2 mark)
- testConstructorGamingTemplate (2 mark)
- testConstructorStudyTemplate (2 mark)
- testConstructorEmptyServerName (2 mark)
- testSetName (1 mark)
- testSetNameEmpty (1 mark)

- testAddUser (2 mark)
- testAddDuplicateUser (2 mark)
- testKickUser (2 mark)
- testKickUserNotOwner (2 mark)
- testKickUserNonExistingUserOrKickingOwner (2 mark)
- testAddChannelOwner (2 mark)
- testAddChannelNotOwner (2 mark)

5. Program Demonstration

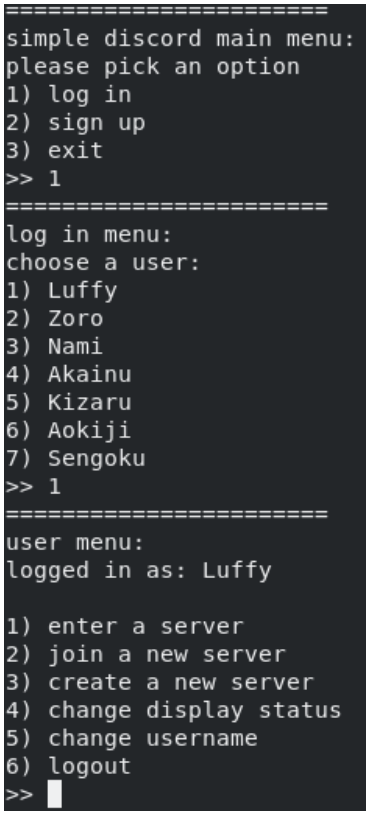

Figure 3: Start of the program → log in → user menu

1. Start of the program
2. Choose to log In
3. Log in as Luffy (The program is populated with 7 Users from the beginning)

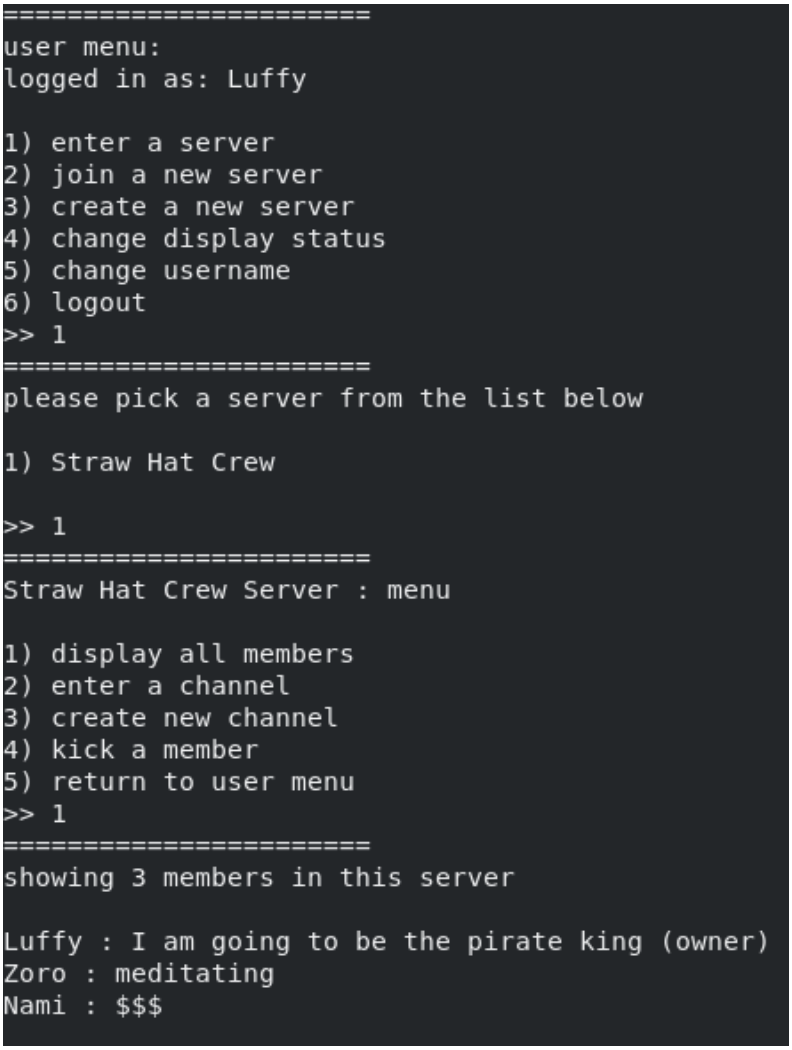

Figure 4: user menu → server menu → display all members

1. Logged in as Luffy, currently in the user menu
2. Choose to enter a server that Luffy has joined
3. A list of Servers that Luffy has joined is shown (there is only 1 Server)
4. Choose to enter “Straw Hat Crew” Server
5. Currently in the “Straw Hat Crew” Server menu
6. Choose to display all members in this Server
7. A list of all the User’s name and their status is shown. (total of 3 Users) Also note that the (owner) is marked.

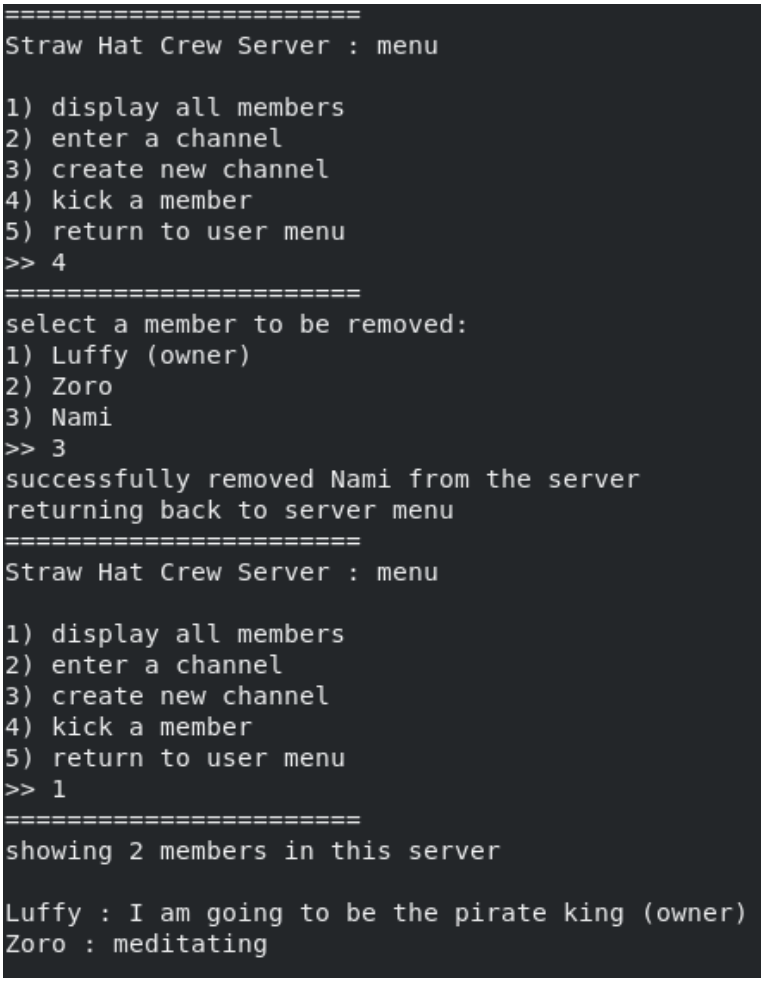

Figure 5: server menu → kick member

1. Currently in the “Straw Hat Crew” Server menu
2. Kick a member
3. Kicking the 3rd member Nami
4. The kick is successful, because the kicker is Luffy (the Server’s owner)
5. There are 2 members left

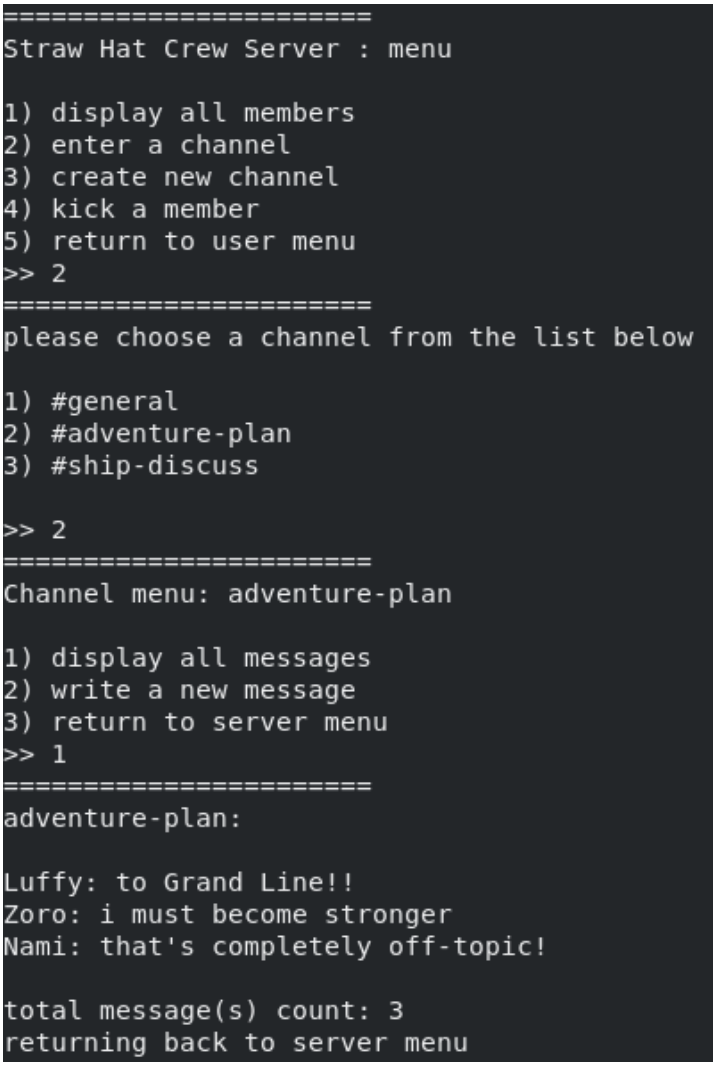

Figure 6: server menu → channel menu → display all Messages

1. Currently in the “Straw Hat Crew” Server menu
2. Choose to enter a Channel
3. A list of Channels that is in this Server is shown (total of 3 channels)
4. Choose to enter the “adventure-plan” Channel
5. Currently in the “adventure-plan” channel menu
6. Choose to display all Messages sent in this Channel
7. A list of all Messages that is sent in this Channel is shown (currently there are 3 Messages)

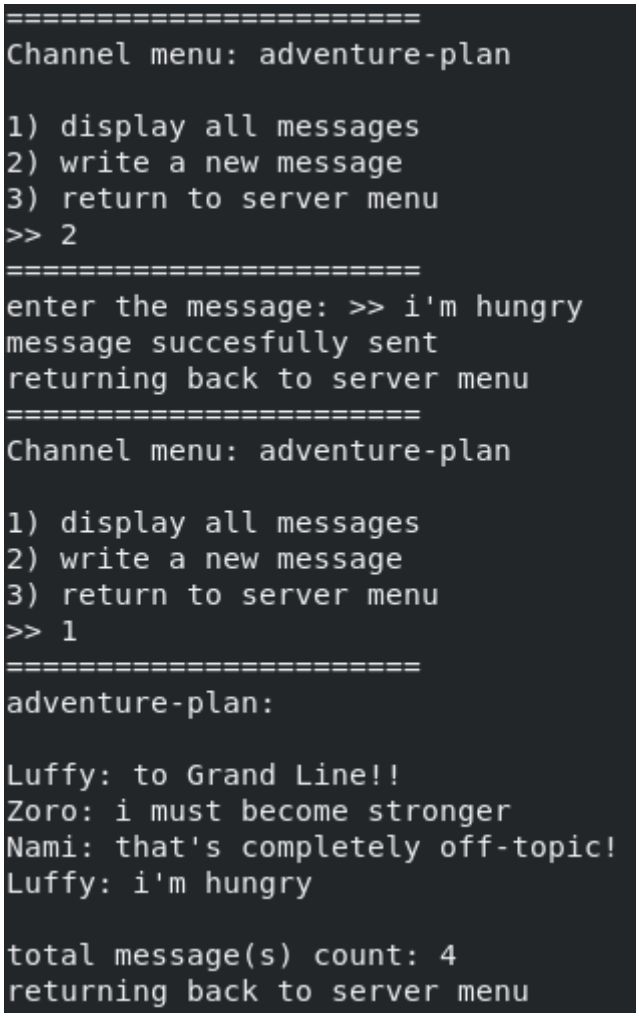

Figure 7: writing a new Message to the channel

1. Currently in the “adventure-plan” Channel menu
2. Choose to write a new message (“i’m hungry”)
3. Now there are a total of 4 messages in this Channel

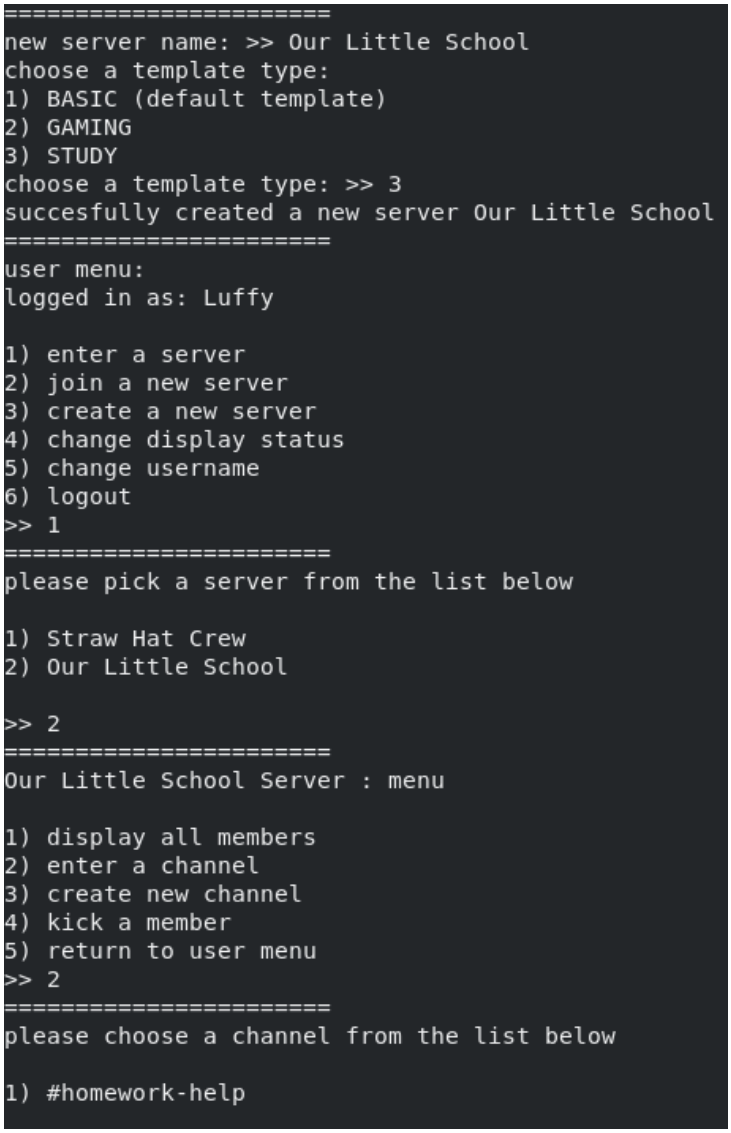

Figure 8: creating a new Server with TemplateType STUDY

1. Create a new Server called “Our Little School”
2. Choose the TemplateType as STUDY
3. There are now 2 Servers that the User Luffy is a member of
4. In Our Little School Server, there is a #homework-help channel, because the chosen TemplateType was STUDY