Final Project

Object-Oriented Programming

Project Name: **Alfinanator but Java**

Name: **Muhammad Alfin Rizqullah**

Student ID: **2502036842**

Class: **L2BC**

Course Code: **COMP6699001**

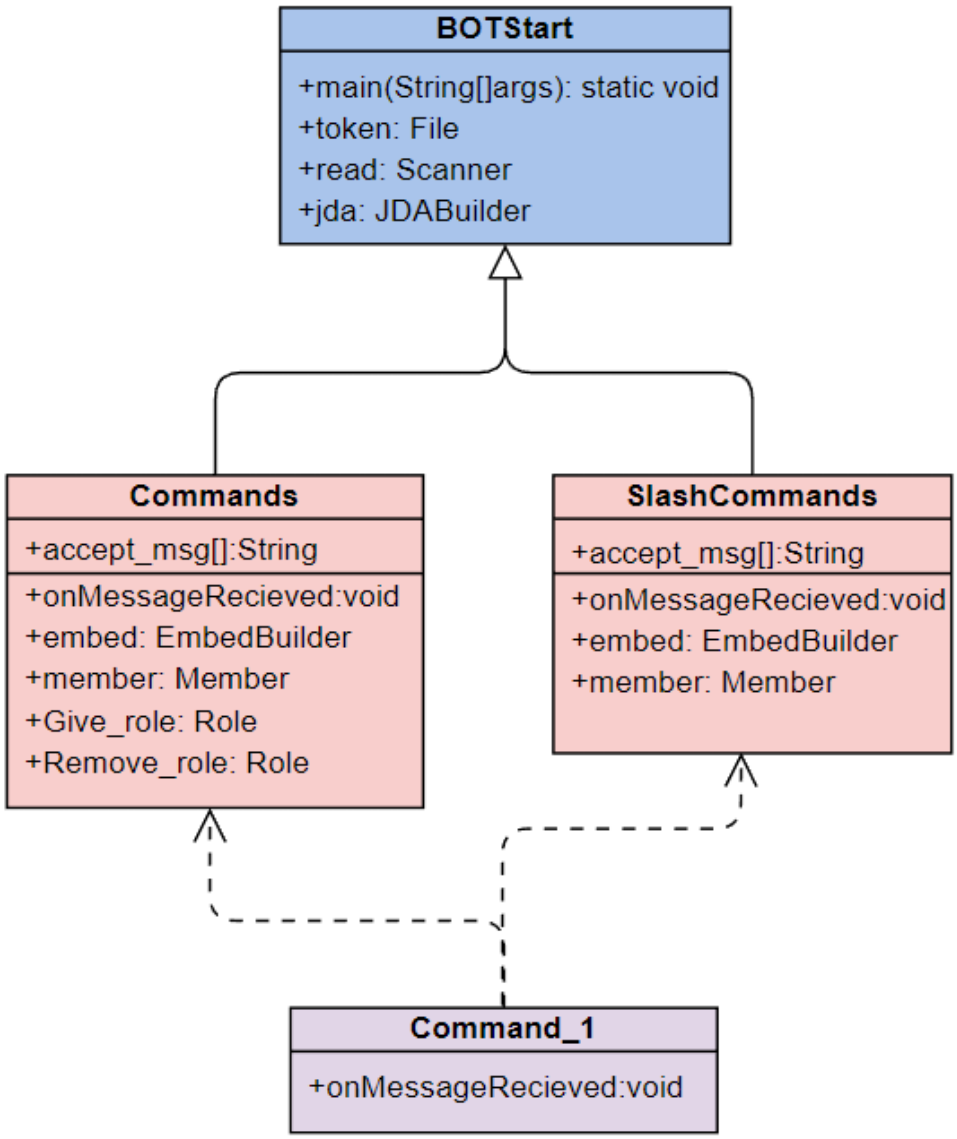
Program Description

My original plan for this program was to create a GUI a relatively simple application intended to help a business/ school that needs the capability of booking seats. My main issue was the GUI itself, needing to learn how to make GUI elements. After spending around 40 to 50 hours on the GUI, and with the deadline approaching I decided to create a Discord Bot Again.

I wanted the bot to have similar functionality to my previous Discord Bot, but with added features. With the same modularity that would allow the developer to expand on this bot without too many issues. What I ended up making with the time crunch that I had, was not the best and not as good as the previous version.

The bot itself would have the capability to edit the roles of each user that is a member of a certain server. The ability to add a role to a user while also removing them. The original idea also had the capability of banning specific users, and kick them out of the server, but it is currently not available for testing due to running out of time.

Class Diagram



Input

1. .test
2. .giverole @ROLE @USER
3. \ahsan
4. \adrian

5. \edi

Output

1. Will print out “Bot is online”
2. Give use of the specified role
3. Print out Teehee
4. Print out Wassup
5. Print out Shap!

Solution Design

1. Either .commands or \commands
2. Give input of either user and roles
3. Will either remove or add the role to the user

Implementation of Code

JDA

In Java to store a lot of data locally for functions, it uses JAR files, which stands for Java Archive. JDA is an external JAR file that has to be manually inputted to a person’s IDE because it does not come with any pre-installed IDE while also carrying all the custom classes that have been designed by a third party.

DISCORD

In Discord, there are a couple of words that have different names internally and externally. **Guild** is what is known internally by discord as **Servers**, while **Users** or **Members** are used interchangeably depending on what part of the code you are doing.

BOTStart.java:

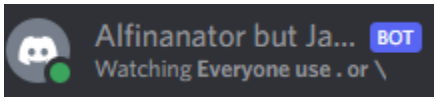
Token

```
public static void main(String[]args) throws LoginException, FileNotFoundException{
    File token = new File("C:\\Users\\alfin\\eclipse-workspace\\token.txt");
    Scanner read = new Scanner(token);
```

Like other discord bots, a token is needed before you are allowed to have your BOT roaming around in servers. This ensures that nobody else can use your BOT account for something else, preventing phishing if your BOT has admin permissions. Storing the token itself is also unsafe because in the event you want to upload it to Github or share it with a friend, someone can simply take your bot and use it for their own need. Simply using the IO class of File to allow my code to read the token from the text file stored in a different part of the system.

JDABuilder

```
JDABuilder jda = JDABuilder.createDefault(read.nextLine());
jda.setActivity(Activity.watching("Everyone use . or \\"));
jda.setStatus(OnlineStatus.ONLINE);
```



JDABuilder will start the compilation of all the code needed to launch the bot. Even though I didn't code *Polymorphism* into the code directly, the JDA class import already does have it prepared for every single part of the method, with *setActivity()* and *setStatus()* as an example here.

Speaking of the two methods, *setActivity()* use is to create a short text below the name of the bot itself, while *setStatus* is to set whether show the bot is online, away, do not disturb, or invisible.

addEventListeners()

```
jda.addEventListeners(new Commands());
jda.addEventListeners(new SlashCommands());
```

Allows the class of *Command()* and *SlashCommands()* to scan for events and later on implement it with code.

ChunkingFilter, MemberCachePolicy, GatewayIntent

```
jda.setChunkingFilter(ChunkingFilter.ALL);
jda.setMemberCachePolicy(MemberCachePolicy.ALL);
jda.enableIntents(GatewayIntent.GUILD_MEMBERS);
```

All of these lines of code have really one purpose, and that is to allow the interaction of the bot with the user that the bot is in.

Command_1.java

```
public interface Command_1 {
    public void onMessageReceived(MessageReceivedEvent event);
```

Command_1.java is an interface file, it is created to allow the method *public void onMessageRecieved(MessageReceivedEvent event)* to be used in more than one class file. The job of this method is to enact an event/ function when the message has been received.

Commands.java & SlashCommands.java

I grouped Commands.java & SlashCommands.java in the same vein because, in essence, they both have a similar file structure.

Accepting Messages

```
@Override
public void onMessageReceived(MessageReceivedEvent event) {
    String [] accept_msg = event.getMessage().getContentRaw().split(" ");
    String prefix = ".";
```

To start off, the class *ListenerAdapter* is extended to this file, while also implementing an interface for the method *public void onMessageRecieved(MessageReceivedEvent event).* An array is created to store the message for commands that require more than one word. The prefix is what the user will type in before the actual command to indicate that they are trying to communicate with the bot.

Replying to Messages

```
if(accept_msg[0].equalsIgnoreCase(prefix + "test")) {
    event.getMessage().reply("this bot is online!").queue();}
```

The if statement will only read the first part of the message that is stored in the array. *equalsIgnoreCase* is there so that no matter if it is a capital letter or lower case the command will still go to. *getMessage()* will find where the message originated from and *reply()* will simply reply to them, and *queue()* will push the actual code to discord.

Sending Messages

```
if(accept_msg[0].equalsIgnoreCase(prefix + "hi")) {
    event.getChannel().sendMessage("Hello!").queue();
```

The only difference between this code and the one above it is that *sendMessage()* will simply post the message on the channel that it was initiated in, it will not reply the user but simply send the specified message.

Embeds

```
if(accept_msg[0].equalsIgnoreCase(prefix + "commands")) {
    EmbedBuilder embed = new EmbedBuilder();
    embed.setTitle("List of Commands for Period", "");
    embed.setDescription("Use the prefix . (Period). Can be upper case or lower case");
    embed.addField("test", "Respond if bot is Online", false);
    embed.addField("reply", "Bot will reply", false);
    embed.addField("addrole", "will give role to users, simply @role @user", false);
    embed.addField("removerole", "will give role to users, simply @role @user", false);
    embed.setColor(Color.BLUE);
    embed.setFooter("This BOT is created by Alfin", event.getGuild().getOwner().getUser().getAvatarUrl());
    event.getChannel().sendMessage(embed.build()).queue();
    embed.clear();
```

EmbedBuilder() will create an embed list that the developer can customize. *setTitle()* will put the title of the embed and *setDescription()* will allow the embed to have a description. *addField()* accepts 2 string input and a boolean, the boolean itself does not actually do anything with the current configuration but if I added buttons in the future it will matter. *setColor()* will allow the edge of the embed to have different colors. For *setFooter()* the code that is present will allow the presence of the server owner's profile pic.

Adding Role

```
if(event.getMessage().getMentionedRoles().toArray().length == 1) {
    if(event.getMessage().getMentionedUsers().toArray().length==1) {
        Member member = event.getGuild().getMember(event.getMessage().getMentionedUsers().get(0));
        Role Give_Role = event.getMessage().getMentionedRoles().get(0);
        event.getGuild().addRoleToMember(member, Give_Role).queue();
        event.getMessage().reply("The Role " + Give_Role.getAsMention() + " has been given to " + member.getAsMention()).queue();
```

This set of codes is for the adding of roles to individuals using the bot. *getMentionedRoles()* would accept the roles that have been given in the message and store them in the list, then it would then be checked with *to.Array().lenght* to see whether one or more roles have been mentioned or not. The code below it is similar the only difference is that instead of accepting roles, it is accepting users with *getMentionedUsers* and it would continue like the code above.

The member object would be the one to actually fetch the users from the array, while the role would the same thing but with the mentioned roles. The rest of the code simply continues and will eventually change the roles of the specified users.

Error-checking

```
    }else {
        event.getMessage().reply("Only 1 user at a time please").queue();
    }
}else {
    event.getMessage().reply("1 role at a time please").queue();
}
```

In the error-checking part of adding a role to a user, currently, if the parameters are outside of boundaries it will simply print this, but in the future, I can increase the size of the allowed roles and user to be changed.

Removing role

```
if(accept_msg[0].equalsIgnoreCase(prefix + "removerole")) {
    if(event.getMessage().getMentionedRoles().toArray().length == 1) {
        if(event.getMessage().getMentionedUsers().toArray().length==1) {
            Member member = event.getGuild().getMember(event.getMessage().getMentionedUsers().get(0));
            Role Remove_role = event.getMessage().getMentionedRoles().get(0);
            event.getGuild().removeRoleFromMember(member, Remove_role).queue();
            event.getMessage().reply("Role has been removed " + Remove_role.getAsMention() + " to " + member.getAsMention()).queue();
        }else {
            event.getMessage().reply("Please mention only 1 user").queue();
        }
    }else {
        event.getMessage().reply("Please mention only 1 role to give").queue();
```

In essence, these 2 functions have almost the same syntax, being the only thing different is the *addRoleToMember()* and *removeRoleFromMember()* other than that they are functionally the same piece of code.

Reflection

I think for reflection I shall start with what I've found out about myself. When I make a schedule (and that is an if at best) I don't tend to stick with it, even when I have alarms set, or focus mode on. I find it very difficult to focus my thought on a specific thing which can result in having issues in both doing work and relaxing. The cycle usually goes, work -> stressed out -> relax -> stressed out -> work and this goes on and on, and the result is that I'm being inefficient at both.

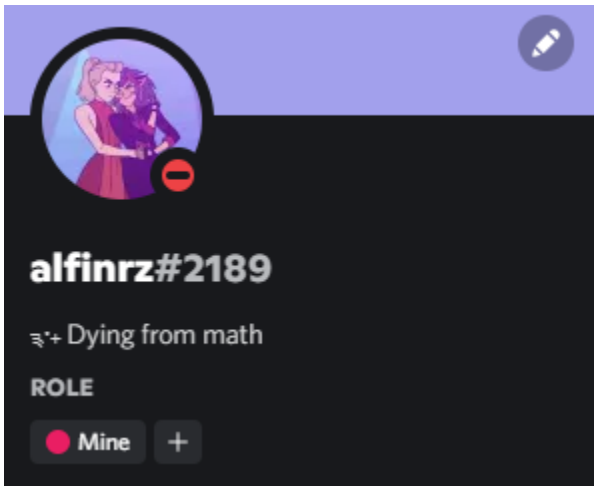
In terms of my skill in coding, my issue from the previous semester is still prevalent, one of them is that it can take much longer than usual to properly understand a basic topic, while also finding it very hard to actually create my own code without errors. And when I do encounter errors, instead of asking for advice, I simply try to push forward by finding sources online which may or may not help.

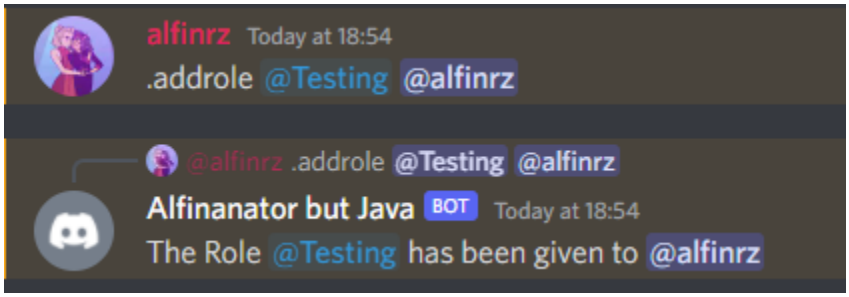
Going back to the start, I stated that I changed the idea relatively recently, and I went with a discord bot again. One of the main reasons for wanting to create a bot again is to see how different it is to code in python and java. After "finishing" this code I do find that the syntax in Java to be a bit more complicated while also needing on average a longer line of code to achieve the same result. But I do like the modularity of Java itself. Overall though I do still prefer Python because of its relatively simplistic syntax while also being the first language I've ever learned, which translates to me going back to python coding syntax if I'm stressed out.

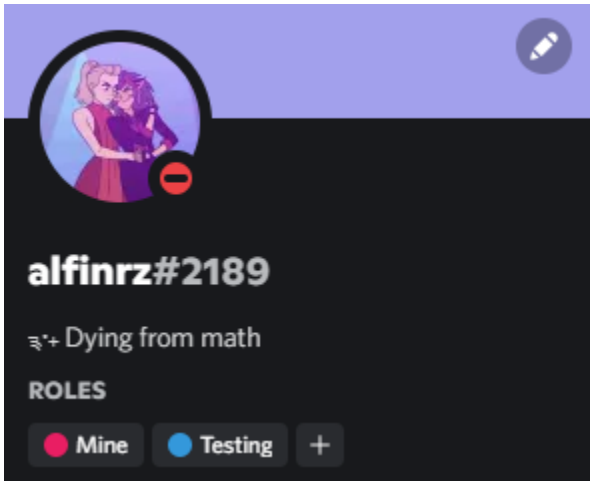
Overall though this semester has increased my knowledge in overall coding, there are still some kinks in the armor that needs to be ironed out, especially my time management. Growing up I didn't have a lot of

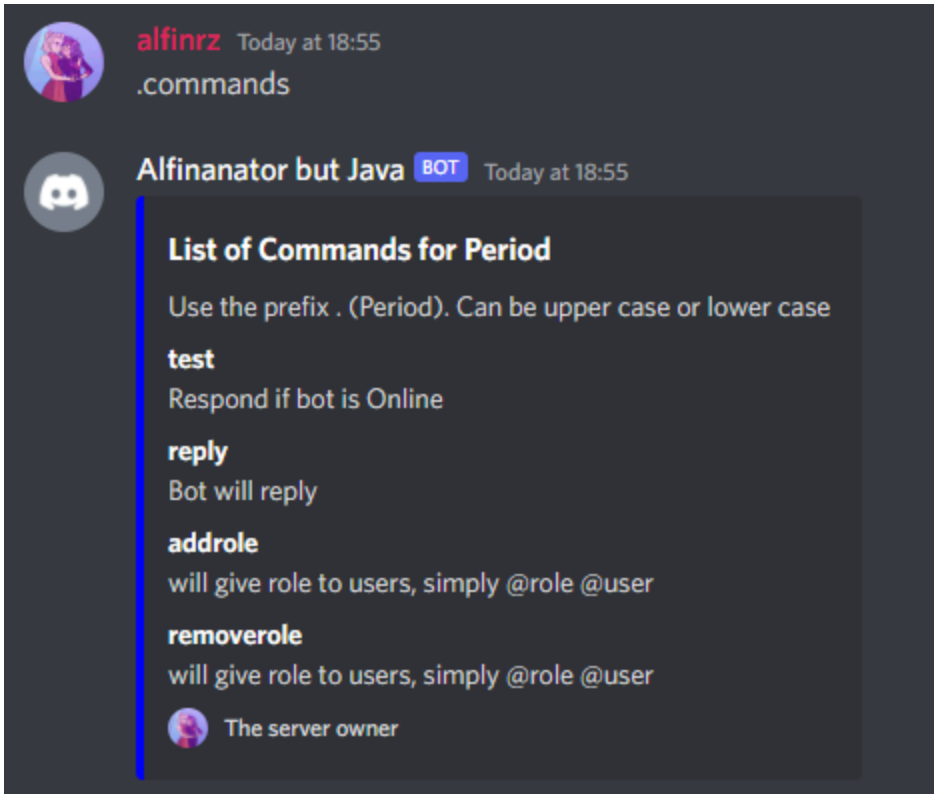
friends because more often than not I was a loner and even when someone was talking to me, I would barely respond. Now I feel like I have way more people to talk to and that does distract me away from my task.

Proof of Code

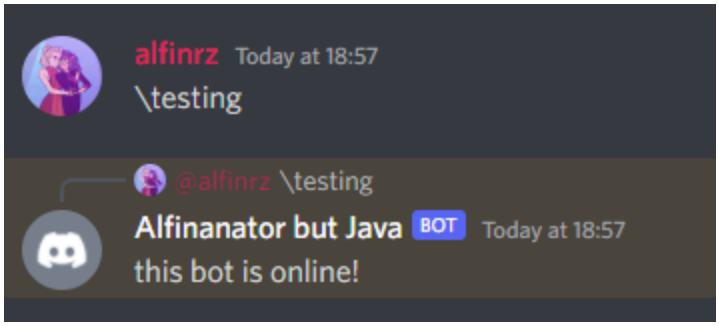


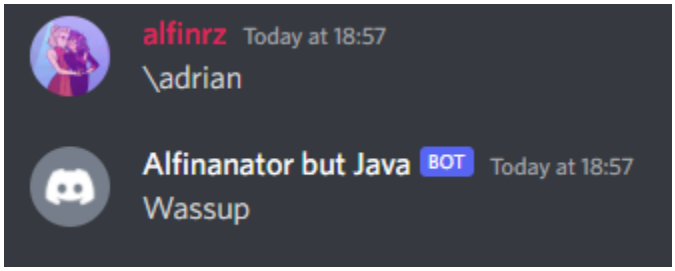





alfinrz Today at 18:55
.commands
Alfinanator but Java BOT Today at 18:55
List of Commands for Period
Use the prefix . (Period). Can be upper case or lower case
test
Respond if bot is Online
reply
Bot will reply
addrole
will give role to users, simply @role @user
removerole
will give role to users, simply @role @user
The server owner


Alfinanator but Java BOT Today at 18:56
List of Commands
Use the prefix \ (backslash)
testing
will reply to you
ahsan
will greet
adrian
will greet
edi
will greet
This BOT is created by <Alfin>





Project Link

https://github.com/alfinrz/OOP_Final-Project_-Semester-2

References

https://ci.dv8tion.net/job/JDA/javadoc/net/dv8tion/jda/api/JDA.html
https://github.com/DV8FromTheWorld/JDA/releases/tag/v4.3.0
https://ci.dv8tion.net/job/JDA/javadoc/net/dv8tion/jda/api/entities/Guild.html
https://ci.dv8tion.net/job/JDA/javadoc/index.html
https://ci.dv8tion.net/job/JDA/javadoc/net/dv8tion/jda/api/entities/package-summary.html