SPŠE Ječná

Informační a komunikační technologie

Ječná 517, 120 00 Nové Město

Discord bot

Charming bot

Samuel Kuta

Informační technologie

2021/2022

Obsah

1 Goal

Creating a functional Discord bot that can join any server and be useful in some way. Not meant for server managing, I wanted the bot to be able to join any server, and not disturb its structure, just entertain or be useful. That is why, even though the bot does have ADMIN privileges on all servers he is allowed to join, he does not have any commands that can directly interact with the members of a discord server. (Such as kick, mute, ban etc.)

2 Hardware

The hardware the bot was developer on and works well on:

CPU : AMD Ryzen 7 2700X Eight-Core Processor 3.70 GHz

GPU: NVIDIA GEFORCE RTX 2070

RAM : 16GB

3 Software

Software used to develop the bot:

Languages used : Java JDK version 16.0.2

IDE : IntelliJ IDEA version 2022.1

Gradle version 7.4.2

Kotlin: 1.5.31

Groovy: 3.0.9

Ant: Apache Ant(TM) version 1.10.11 compiled on July 10 2021

JVM: 18.0.1.1 (Oracle Corporation 18.0.1.1+2-6)

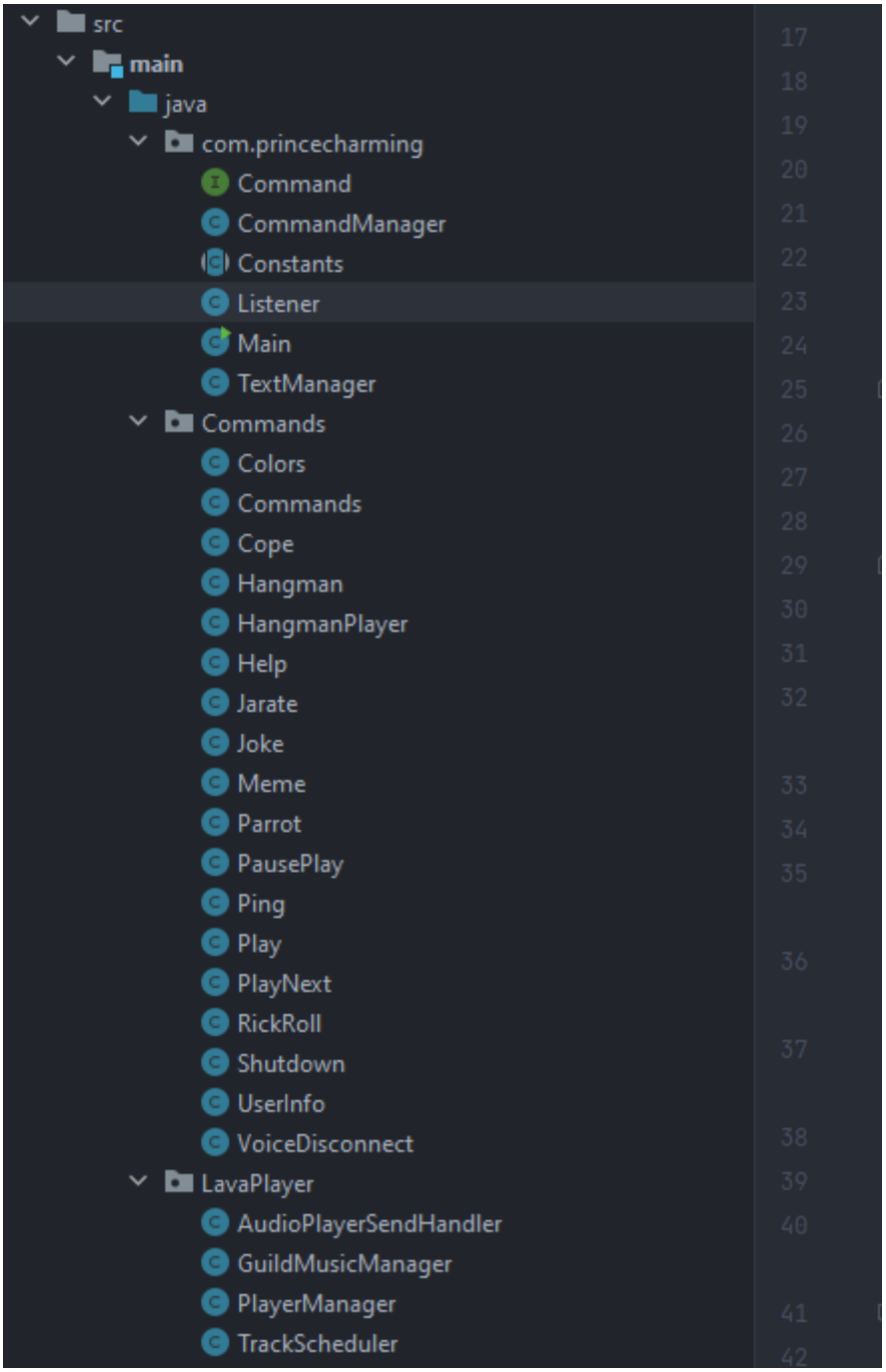
OS: Windows 10 10.0 amd64

4 Code

The bot is programmed using the latest Java Discord API (JDA from now on), which is version 5.0.0-alpha.12 available here.

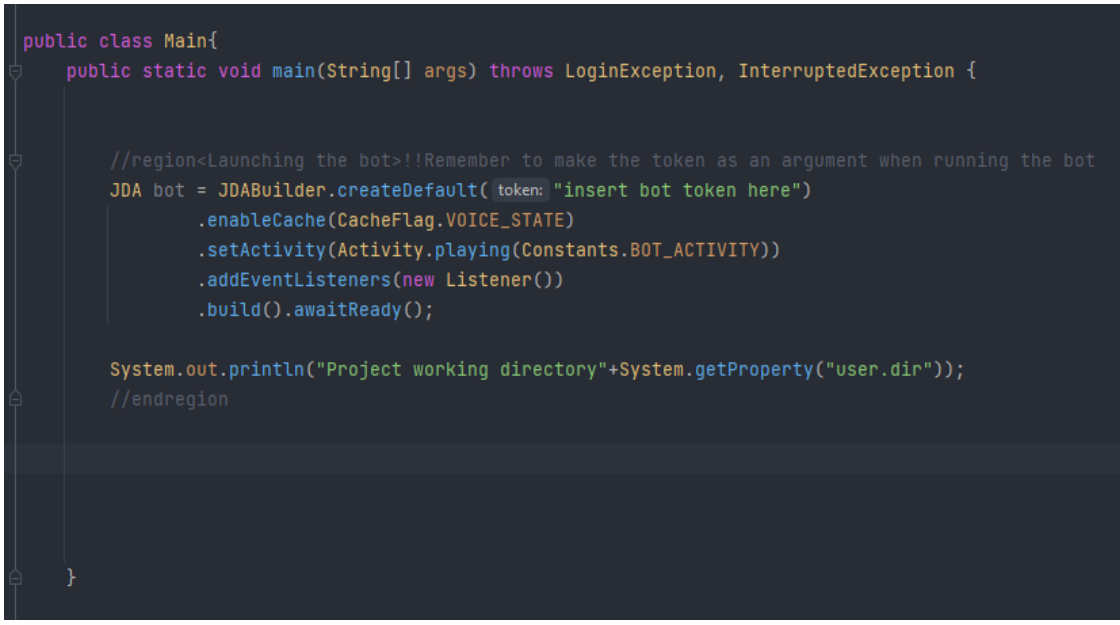
JDA allows you to acces many bulit-in functions to interact with any Discord server, it's settings, and all it's parts such as channels, members ets.

The entire project is made using a total of 28 Java classes split into 3 packages located in src/main/java.



com.princecharming package

- Contains all code related to the bot's setup and functionality
  -Main : The class where the bot is initialized, which turns him online on any server the bot is a member of.



```
public class Main{
    public static void main(String[] args) throws LoginException, InterruptedException {


        //region<Launching the bot>!!Remember to make the token as an argument when running the bot
        JDA bot = JDABuilder.createDefault( token: "insert bot token here")
                .enableCache(CacheFlag.VOICE_STATE)
                .setActivity(Activity.playing(Constants.BOT_ACTIVITY))
                .addEventListeners(new Listener())
                .build().awaitReady();

        System.out.println("Project working directory"+System.getProperty("user.dir"));
        //endregion




    }
```

-Command interface : Interface that all commands supposed to be issued to the bot must implement. It has 4 methods :

-void run(List<String> args, MessageReceivedEvent)

This method is where the main code of every command is, and is later called in CommandManager class.

-String getName()

Returns the name of the command.

-String getHelp()

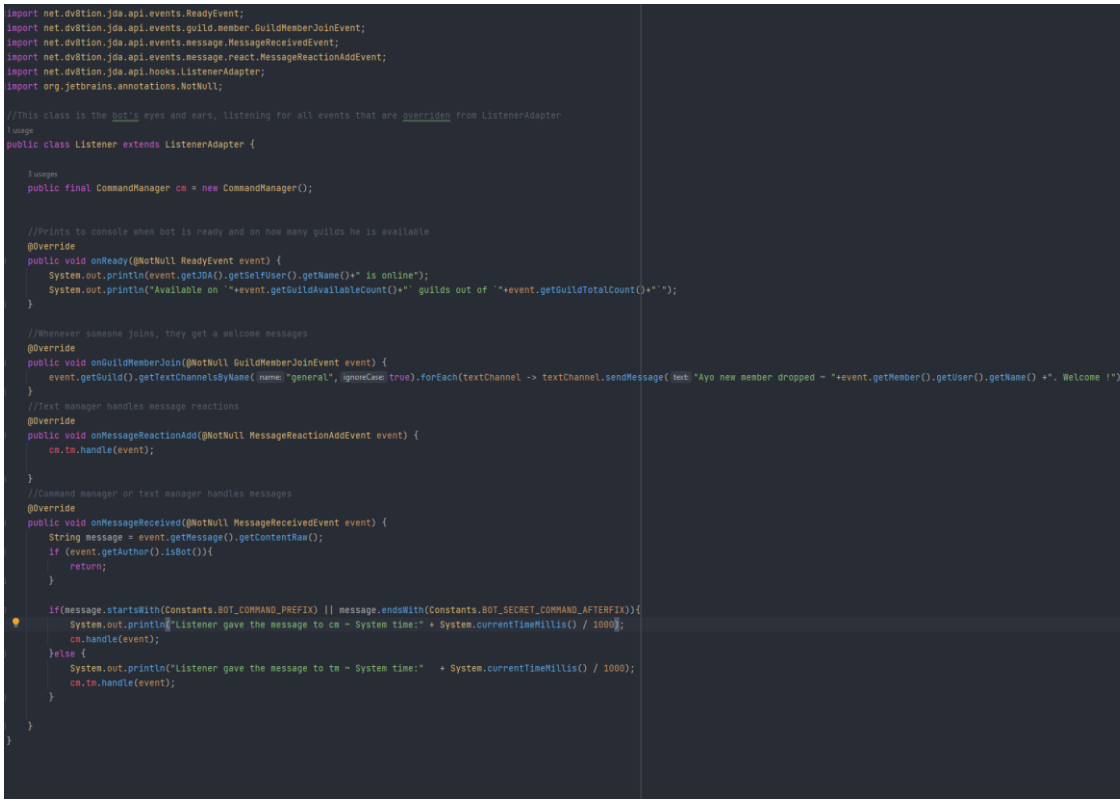
Returns information about the command and how to use it.

-boolean needOwner()

Decides if you need bot owner permission for the command to call it's run method.

```java
package com.princecharming;

import net.dv8tion.jda.api.events.message.MessageReceivedEvent;

import java.util.List;

//Command interface that all bot commands must implement

17 implementations
public interface Command {


    //In this method is the code that is ran everytime a command is issued
    17 implementations
    void run(List<String> args, MessageReceivedEvent event);
    17 implementations
    String getCommandName();
    17 implementations
    String getHelp();

    //Sets if botOwner permission is required for the command
    17 implementations
    boolean needOwner();

}
```

-Listener class : Class that extends ListenerAdapter class provided by JDA, containing all the different event handlers the bot has and can use. In the Listener class, there are 4 listeners implemented in this project, listening for different actions on a Discord server. Listener has it's own CommandManager class as a variable that is used to handle the events the Listener can pick up.



```java
import net.dv8tion.jda.api.events.ReadyEvent;
import net.dv8tion.jda.api.events.guild.member.GuildMemberJoinEvent;
import net.dv8tion.jda.api.events.message.MessageReceivedEvent;
import net.dv8tion.jda.api.events.message.react.MessageReactionAddEvent;
import net.dv8tion.jda.api.hooks.ListenerAdapter;
import org.jetbrains.annotations.NotNull;

//This class is the bot's eyes and ears, listening for all events that are overriden from ListenerAdapter
1 usage
public class Listener extends ListenerAdapter {

    3 usages
    public final CommandManager cm = new CommandManager();


    //Prints to console when bot is ready and on how many guilds he is available
    @Override
    public void onReady(@NotNull ReadyEvent event) {
        System.out.println(event.getJDA().getSelfUser().getName()+" is online");
        System.out.println("Available on `"+event.getGuildAvailableCount()+"` guilds out of `"+event.getGuildTotalCount()+"`");
    }

    //Whenever someone joins, they get a welcome messages
    @Override
    public void onGuildMemberJoin(@NotNull GuildMemberJoinEvent event) {
        event.getGuild().getTextChannelsByName( name: "general", ignoreCase: true).forEach(textChannel -> textChannel.sendMessage( text: "Ayo new member dropped ~ "+event.getMember().getUser().getName() +". Welcome !"
    }
    //Text manager handles message reactions
    @Override
    public void onMessageReactionAdd(@NotNull MessageReactionAddEvent event) {
        cm.tm.handle(event);

    }
    //Command manager or text manager handles messages
    @Override
    public void onMessageReceived(@NotNull MessageReceivedEvent event) {
        String message = event.getMessage().getContentRaw();
        if (event.getAuthor().isBot()){
            return;
        }

        if(message.startsWith(Constants.BOT_COMMAND_PREFIX) || message.endsWith(Constants.BOT_SECRET_COMMAND_AFTERFIX)){
            System.out.println("Listener gave the message to cm ~ System time:" + System.currentTimeMillis() / 1000);
            cm.handle(event);
        }else {
            System.out.println("Listener gave the message to tm ~ System time:"   + System.currentTimeMillis() / 1000);
            cm.tm.handle(event);
        }

    }
}
```

-Constants class : Class that has some constants and collections that can be accessed by different classes in the entire project, Consider this a sort of „settings“ class, where you can change for example the prefix used to call commands, or the owner ID of the bot and more.

```java
package com.princecharming;


import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

//Class with some constants the rest of the code references, consider this a sort of settings class
public abstract class Constants {

    public static final String BOT_COMMAND_PREFIX = "!";

    4 usages
    public static final String BOT_SECRET_COMMAND_AFTERFIX = "?";

    //Bot owner ID
    1 usage
    public static final Long OWNER_ID = 257377227451269122L;

    1 usage
    public static final String OWNER_NAME = "Horse";

    1 usage
    public static final String BOT_ACTIVITY = " with your dad #]";




    <Colors command stuff>
}
```

-CommandManager class : Class that basically takes care of all MessageReceivedEvents spotted by the Listener class, and manages all bot commands. This class has a list of all the commands, and all the commands have to be added to this list in the constructor of CommandManager.
CommandManager also has it's own „sub-manager" TextManager class, that will be explained later.
CommandManager handles the events caught by the Listener class through it's handle() method, where a MessageReceivedEvent (basically just a message in some text channel on a server) is passed in, and the handle method checks if it's a command, and then splits it into 2 parts : 1)Command name 2) List of arguments – arguments are seperated by spaces.
After it does this, it fetches a command from it's command list and calls it's run() method, passing in the List of arguments and the MessageReceived event..

-TextManager class: Functions the same as CommandManager, it just doesn't keep a list of all the commands. It also has a handle() method, where non command messages are handled.
Essentially exists just to not clutter up the handle method of command manager with too much code.
This class also keeps track of current Hangman (package Commands.Hangman & Commands.HangmanPlayer) players, and Parroted members (package Commands.Parrot).

Commands package

-Contains all bot commands. They are explained fairly well in the source code, so I will not explain every single one here.
Every command must implement the Command interface (com.princecharming.Command) and must be manually added to the constructor of CommandManager (com.princecharming.CommandManager)

LavaPlayer package

-Contains everything required for the bot to play music.
This is achieved by implementing the LavaPlayer audio library for Discord. Here
The version used is a forked repository from the main branch located here.

5 Conclusion

The bot turned out very well I'd say, however due to lack of time I wasn't able to turn the code into a runnable jar or exe file, so the bot must be ran from some Java IDE, preferably IntelliJ IDEA, because it works very well with Gradle and it's dependencies.

Just open the project in an IDE, pass in a bot token and it should work. HOWEVER there are two things that might have to be changed depending on how the working directory is recognized on a PC. In package Commands, two commands require to acces a file in „src/main/resources", and the path to this file is hard coded. It might have to be slightly changed for these commands to work properly, but the but will function completely fine even if these paths are not changed, just the commands won't.

6 Zdroje

[1] Internet věcí. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2020, 27. 11. 2019, 10:48 UTC [cit. 2020-01-04].Available here: https://cs.wikipedia.org/wiki/Internet_věcí

[2] JDA GitHub repository Documentation, 2022, 05.06.2022 21:24 UTC. Available here: https://github.com/DV8FromTheWorld/JDA

[3] LavaPlayer GitHub repository Documentation, 2022, 05.06.2022 21:24 UTC. Available here: https://github.com/sedmelluq/lavaplayer
[4] JDA Javadocs documentation, 2022, 05.06.2022 21:24 UTC. Available here: https://javadoc.io/doc/net.dv8tion/JDA/latest/index.html
[5] Various youtube videos (not sure if I should event put this here as a source but might as well)
[6] Various discussions and posts on Stack Overflow concerning JDA.