Developer’s Documentation

Developer‘s Documentation of the program Discord Crypto Alert Bot

Programming in Java (NPRG013) at MFF UK.

Author: Tomáš Pop

Date of issue: 18/05/2022

Version: 1.0

Introduction

In the developer's documentation you can read about algorithms, decisions made during the development process and some implementation details included in the Discord Crypto Alert Bot.

Used Tools

For the development, the author used IntelliJ Idea 2021.2 Ultimate on Windows and WSL Ubuntu 20.04 for Linux compilation.

The main reason why the external libraries were utilized, is due to the lack of JSON processing library in the standard Java. Furthermore, to communicate with Discord API, a simple option is to utilize a wrapper such as JDA (Java Discord API) which was used in the project.

To build the project, it is required to use Gradle build tool which enables the project to be multiplatform and easens the process of installation of external dependencies mentioned above.

From the runtime dependencies point of view, the rest of the project is made using the standard library only and the program should be compilable using Java 16 (it should run on other versions too, nevertheless, the author used Java 16 – Amazon Corretto v16.0.2).

Project Structure

ToTheMoon is running using Gradle, necessary build files are contained in the root directory. The entrypoint source file, Entrypoint.java, is located using the typical java directory management and the reversed domain (src/main/java/cz/cuni/mff/semestral/Entrypoint.java). In the semestral directory, there is a variety of directories and within them java files which take care of the logic of the application. In case you want to make changes concerning the code, make sure you refactor properly (IntelliJ Idea helps a lot in this particular manner).

Common Workflow

At the start of the application, nothing happens until an user of a Discord server where the bot is deployed writes something to a text channel (any of them). The user is demanded to signalize a command with a special leading symbol to detach from the regular messages (otherwise the bot would answer to every user's message with a parse error or similar message). The list of commands in the version 1.0 is shown below:

Each user can maintain own watchlist (a list of cryptocurrencies which price the user is interested in) and an alert storage which contains alerts of prices of particular cryptocurrencies. The alert is triggered after a price (or an equivalent in percentage) is surpassed and the user is informed via a message in the channel where the alert was created.

Note that the program should never end uncontrollably or throw any exceptions besides of the information to the user that he has done something wrong. For example, if the user fills in an unknown, not supported action, it is expected that the user is only warned by the console that the behavior is not supported/that the action is not available (for example when even when the number of parameters is not right as shown in the !add [symbol], !alert [symbol] [value] or !rml [symbol] commands).

Invalid i.e. inconvertible parameter values (missing or not integral), non-existing cryptocurrency symbol addition or removal should be handled properly as well.

Considering that the user uses the library otherwise with only valid operations and correct number of parameters, the common dialog should be that the user fills in a command and receives a response as a message. The only output done by the bot wihout reacting to an immediate event (user's message) are the alert triggers.

One and only way to end the program (besides of forcing the end by quitting completely) is to simply stop the execution of the program since our intention is to keep the bot running ideally 24/7.

Interesting Classes and their Data Structures

In this section, there is a brief description of some interesting classes and their data structures utilization.

Communication Class

- The communication class is the only event listener of the JDA (Java Discord API)
- It asks the BinanceConnection hook to connect to the API with a pre-set delay (in order not to spam requets and get suspended)
- It takes care of a proper message receiving and answering accordingly
- It checks regularly for finished (triggered) alerts

Data Structures

- A hook to the Processor class
- A hook to the BinanceConnection class
- EmbedBuilder class which takes care of the visual apperance of the message (Discord MessageEmbed)

Processor Class

- Takes care of the fluent dialog between the user and the rest of the program
- Does all the parsing stuff
- Returns processed user's input to the class which takes care of the dialog between the bot and the user
- Commands definitions

Data Structures

- Options which are currently supported are defined using an enum class
    - Their mapping to the string literal entities are done using an EnumMap which contains the key as an enum item
    - These keys map to their real string representation including their description for a help command (Command data class which has only these two string attributes)
    - These options are mapped as keys in another EnumMap where they are mapped to particular functions (using Supplier for the simplest and Function for the parameterized functions).

Decisions Made

In this section, I would like to shift to the first person view.

Printing

- I really do not enjoy writing language specific printing in pretty much any language where the print function is rather wordy. So across the program I use print as a generic version of print which is stored in the Utilities.java – which otherwise contains mostly language abstraction functions
- The initial version of bot's answers contained a blank text only which was alright, nevertheless, I ran into the MessageEmbed which is fairly used when developing a bot (I wanted a pretty version of !help command) which resulted in minor changes utilizing EmbedBuilder class which takes care of the "pretty" appearance of bot messages
    - EmbedBuilder helped with distinguishing messages between each other
    - In case, multiple users entered commands for the users since the EmbedBuilder contains a footer with the user who initiated the request in a simple manner

All Users Involved

- Originally, I wanted to support only a signel user-bot communication on a detached server

- Nevertheless, from the practical point of view, it was rather convenient to be able to integrate the bot in an arbitrary server and let it live there
- All users support naturally introduces thread safety concerns, nevertheless, the JDA library provides with asynchronous message processing

Conversation Detachment

- Since I wanted to involve the support for all users on the server, the problem arised as soon as I realized that any conversation triggers the bot which was definitiely not intended
- A typical approach is to detach the commands with a specified symbol (the specified leading symbol + the command should not be encountered in a normal conversation)

Encountered Pitfalls

I would like to describe some of the issues which I encountered and hopefully managed to solve

- At the start of the development process, I tried to find a library which would be compliant with Java + Discord which eventually JDA was the choice. Nevertheless, in the first couple of runs I ran into multiple error messages in the command line which I could not get rid of. After looking it up, for some reason, having error messages in the command line during the run is the expected behavior of the JDA library:

```
18:36:49: Executing task ':Entrypoint.main()'...

> Task :compileJava
> Task :processResources NO-SOURCE
> Task :classes

> Task :Entrypoint.main()
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
SLF4J: Failed to load class "org.slf4j.impl.StaticMDCBinder".
SLF4J: Defaulting to no-operation MDCAdapter implementation.
SLF4J: See http://www.slf4j.org/codes.html#no_static_mdc_binder for further details.
[main] INFO JDA - Login Successful!
[JDA MainWS-ReadThread] INFO WebSocketClient - Connected to WebSocket
[JDA MainWS-ReadThread] INFO JDA - Finished Loading!
```

- While testing multiple options of compilation (Windows command line, Linux command line, Windows IntelliJ Idea), I realized that I created multiple instances of the application which resulted in giving duplicated answers for

every command – for the author, it is probably not an issue, nevertheless, this way I realized that the private bot token should not be publicly availabl

    - In case, someone would want to use the logic of the program for an own bot, he should create own bot on https://discord.com/developers/docs/intro and generate his own OAuth2 token which would be used in the Entrypoint.java application instead of mine
- It is necessary to be a bit patient concerning the Discord user interface and bot disconnection (online x offline status) although programmatically the bot was turned off a long time ago, the interface still displayed that the bot is online and available – of course at this situation, the messages entered by the users at this time are not processed anymore and no answer is provided
-

Potential Improvements

The improvement which would be considered the first, is the deployment to the cloud service in order not to care about connection issues on the local machine where the bot is hosted.

In case, there were no cloud service deployment in the future, it may be an idea to store the information in a database to avoid data loss (all users watchlist and alerts storage wipe upon bot disconnection). Nevertheless, the data is not anyhow sensitive and the bot supports multiple commands at once to retain the alerts/watchlist fast back.