**Lesson 27. CommandBot**

Один із найпростіших в той же час найбільш надійних способів взаємодіяти із ботом – команди. Їх просто налаштувати, їх просто обробляти, вони зазвичай мають однозначну відповідь.

Протягом цього заняття познайомимось із командами, а також із тим, як можна додавати у повідомлення бота емоджі, які давно вже стали невд’ємною частиною спілкування онлайн. Емоджі дозволять відповіді бота зробити «живішими» і приємнішими для користувача.

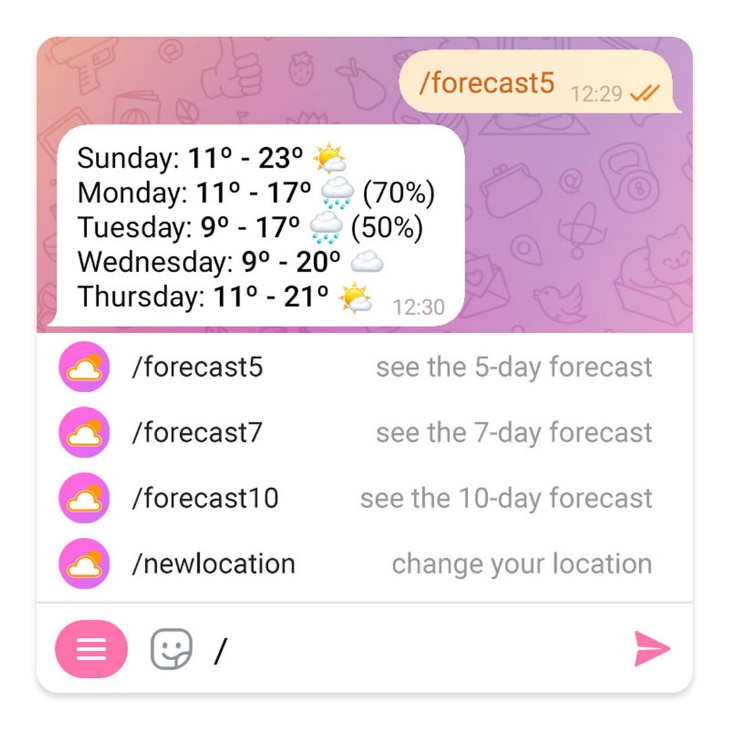
**Commands**

Користувачі можуть надсилати ботам будь-які типи повідомлень, включаючи текст, файли, геолокацію, стікери, голосові повідомлення та навіть гральні кубики для отримання випадкових значень. Однак боти Telegram пропонують також багато інших інструментів для створення гнучких інтерфейсів, адаптованих до ваших конкретних потреб.

Один із них – **команди**, які підсвічуються в повідомленнях і які можна вибрати зі списку після введення символу «**/**».

**Команда** – це всього лиш повідомлення у формі ***/keyword***, яке вказує боту, що робити.

Застосунок Telegram у роботі з командами буде:

* **Виділяти команди в повідомленнях**. Коли користувач натискає на виділену команду, ця команда негайно надсилається знову.
* **Пропонувати список підтримуваних команд** з описами, коли користувач вводить **/** (щоб це працювало, необхідно надати список команд *@BotFather* або за допомогою API). Вибір команди зі списку негайно надсилає її.
* **Показувати кнопку меню**, яка містить усі чи деякі команди бота (які визначені через @BotFather).

Команди завжди мають починатися з символу «**/**» (слеш) і містити до 32 символів. Вони можуть використовувати латинські літери, цифри та підкреслення, хоча для чіткішого вигляду рекомендується використовувати простий текст у нижньому регістрі.

Ось кілька прикладів:

Команди мають бути **якомога конкретнішими** – наприклад, /newlocation або /newrule є кращими, ніж /new команда, яка вимагає від користувача додаткового параметра, наприклад «location» або «rule».

Щоб зробити основні взаємодії більш уніфікованими, розробники телеграм просять всіх розробників ботів підтримувати кілька основних команд. Додатки Telegram матимуть інтерфейсні ярлики для цих команд.

**/start** – починає взаємодію з користувачем, як відправка вступного повідомлення.

**/help** – повертає довідкове повідомлення, наприклад короткий текст про те, що може робити ваш бот, і список команд.

**/settings** – (якщо застосовується) показує налаштування бота для цього користувача та пропонує команди для їх редагування.

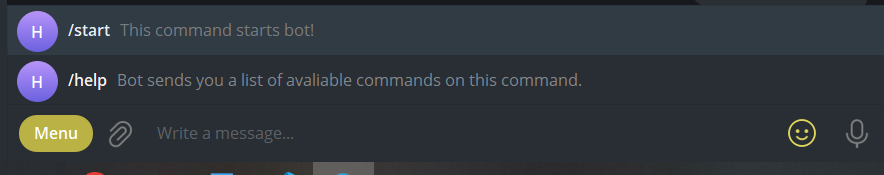
Користувачі побачать кнопку **«Start»**, коли вперше відкриють чат із вашим ботом. Посилання на /help та /settings будуть доступні в меню на сторінці профілю бота, якщо додати їх у @BotFather.

**Register Commands**

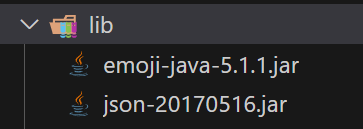
Зареєструвати команди через Батька всіх ботів (<https://t.me/BotFather>) не складно (за допомогою коду це теж можна зробити, але з цим познайомимось трохи згодом):

1. У списку ваших ботів обираємо потрібного;
2. Тиснемо Edit Bot;
3. Тиснемо Edit Commands;
4. Надаємо список у вигляді, який просить бот.

Після цього в меню бота повинні з’явитись додані команди. Наприклад:



**Emoji**

Для використання емоджі у нашому боті знадобляться ще додаткові бібліотеки:

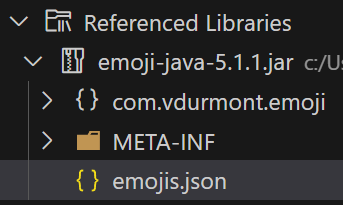
Далі на прикладі простого бота з’ясуємо як додати до повідомлення емоджі. Перш ніж це зробити варто заглянути в клас Emoji з відповідної бібліотеки, аби з’ясувати з яких з чого складається об’єкт емоджі.

Нас найбільше будуть цікавити ось ці три властивості:

**aliases** – «псеводніми» емоджі, щось наче кодові рядки, за якими емоджі можна знайти;

**tags** – за тегами емоджі можуть бути роздідлені на категорії, або тег може містити якусь допоміжну інформацію про емоджі;

**unicode** – код символу емоджі (саме він повинен бути в тексті повідомлення, що телеграм відобразив його як картинку);



Детальну інформацію про емоджі можна знайти у файлі із бібліотеки:

Також з’ясувати еліас емоджі можемо в самому Telegram навівши на якийсь із них курсором:

Еліас зазвичай починається із закінчується символом «**:**». Знаючи еліас можемо відтворити юнікод емоджі у повідомленні. Для цього можна скористатись одним із двох запропонованих способів:

1. Клас **EmojiParser**;

Потрібний нам метод **parseToUnicode**() приймає у якості параметру рядок тексту, а повертає такий самий рядок, але всі знайдені еліаси в ньому заміщує юнікодом емоджі.

1. Клас **EmojManager**;

Даний клас дозволяє за допомогою методу **getForAlias**() знайти об’єкт емоджі за його еліасом. Далі з отриманого об’єкту можна отримати про нього (про емоджі) всі необхідні дані, втому числі юнікод, який потім додати у текстовий рядок повідомлення.

У якості прикладу та поля для досліджень пропонуємо відтворити приклад на скріншоті нижче. В рамках телеграм бота можна поекпериментувати із еліасами, емоджі та запропонованими методами:



Дані класи мають ще багато іншого функціоналу, який ми в рамках курсу не будемо використовувати. Однак це не значить, що самостійно в нього заглибитись ви (учні) не можете за допомогою функціоналу автозаповнення та перегляду файлів бібілотек.

**Create New Bot**

Для того, щоб створити сервер телеграм бота на своєму комп’ютері, необхідно його спочатку зареєструвати в Telegram. Це можна зробити за допомогою батька всіх ботів – @BotFather (<https://t.me/BotFather>).

Перейшовши за посиланням, необхідно в меню команд знайти команду ***/newbot*** і далі дотримуючись інструкцій вказати для бота назву та **username**.

При успішній реєстрації BotFather видасть вам спеціальний і дуже секретний ключ (**token**) для доступу до вашого бота. Саме ці значення (token та username) знадобляться для подальшої розробки бота.

**Creating Bot**

Створивши нашого бота в системі можемо перейти до написання найпростішого із можливих телеграм-бота. У ході роботи дізнаємось купу нового. Тож, до роботи!

**Крок 1**. Створюємо звичайний проєкт Java і додаємо до нього необхідні бібліотеки.

JAR-файли бібліотек, необхідні для створення бота не містяться у стандартному JDK, а отже маємо додати їх у наш проєкт окремо. Розглянемо два простих способи як це зробити:

1. Додаємо JAR-файли прямо у папку lib нашого проєкту:
2. Можемо також вказати шлях до файлів використовуючи Referenced Libraries у вкладці Java Projects. Тиснемо + і шукаємо файли:

**Крок 2**. Створюємо клас у якому буде реалізована головна логіка нашого бота.

Отже створюємо новий клас HelloWorldBot. Наш клас повинен унаслідувати TelegramLongPollingBot, який ми додали у JAR-файлах:

Помилка, яку ми отримали внаслідок наслідування, виникає тому, що клас TelegramLongPollingBot містить абстрактні методи, тобто методи, які не мають реалізації у батьківському класі, але які мають бути обов’язково реалізовані у дочірніх класах.

За допомогою функції QuickFix можемо додати необхідні методи в наш клас:

Редагуємо отриманий код, аби він виглядав для нас більш знайомо:

Отже три методи, які нам необхідно реалізувати мають наступне призначення:

* *getBotToken* – повертає рядок із токеном нашого бота;
* *getBotUsername* – повертає рядок із юзернеймом нашого бота;
* *onUpdateReceived* – містить головну логіку бота. Даний метод викликається автоматично кожного разу, коли сервер Telegram сповіщає йому про зміни, які відбулись у бота (користувач відправив повідомлення, або редагував його, натиснув кнопку, проголосував у опитуванні тощо).

**Крок 3**. Реалізуємо методи HelloWorldBot.

Для *getBotToken* та *getBotUsername* все досить просто. Маємо просто вказати в значення return значення, які ми отримали від Батька Всіх Ботів.

Для *onUpdateReceived* варто звернути увагу на параметр, який має тип Update. Саме у цьому об’єкті міститься необхідна інформація, про те, що саме сталось із ботом, які зміни з ним відбулись. Об’єкт update складається у свою чергу із низки інших об’єктів, а також має набір власних методів. Нам немає необхідності всі їх пам’ятати, оскільки ми можемо їх отримати користуючись підказками IDE.

Для даного уроку нам буде достатньо у методі *onUpdateReceived* запрограмувати наступну логіку: якщо бот отримав повідомлення і у цьому повідомленні міститься текст, то вивести в консоль текст «Hello World!»:

**Крок 4**. Реалізуємо ініціалізацію роботи нашого бота із методу main класу App. На даному етапі пропонуємо зробити запис, такий як пропонують розробники бібліотеки телеграм-бота, а далі зупинимось трошки детальніше:

Після запуску методу *main* якщо ми все зробили правильно, бот повинен запрацювати. Як це перевірити – надіслати боту будь-яке повідомлення. Коли бот отримуватиме повідомлення, повинен у консоль виводити повідомлення «Hello World!».

**Try Catch**

Для наступної задачі нам варто познайомитись з іще одним корисним і важливим блоком, який дозволяє перехоплювати виключення у коді (або помилки).

**Помилки**, або **винятки**, або ж виключення Java можуть виникати у коді, під час його виконання: помилки кодування, зроблені програмістом, помилки через неправильне введення або інші *непередбачені* речі. Ключовим словом являється те, що це *непередбачені* ситуації у коді, тобто такі, під час яких Java не може самостійно вирішити як далі виконуватись.

Коли виникає помилка, Java за замовчуванням зупиняється та генерує повідомлення про помилку. Технічний термін для цього: *Java викликає помилку*, або *викидає виняток/виключення*.

Гарна новина полягає в тому, що в Java існує механізм, який дозволяє перехопити помилку і замість зупинки програми обробити, тобто прийняти рішення що робити, якщо помилка таки виникнула. Конструкція, що дозволяє це робити має назву **try…catch**. Розглянемо як працює.

Інструкція **try** дозволяє вам визначити блок коду, який буде перевірено на помилки під час його виконання. Якщо в коді виникне помилка, то зміни, що спричинив цей код буде відкручено назад, замість цього блоку виконається catch.

Інструкція **catch** визначає блок коду, який буде виконано, якщо в блоці try станеться помилка, а крім того на вхід цього блоку також потрапляє об’єкт помилки (Exception e), яку було перехоплено в try.

**Приклад**. У коді на скріншоті нижче, програма зловивши помилку, замість зупинки програми виведе інформацію у консоль і продовжить своє виконання:

У блок **try…catch** варто «загортати» код, у якому потенційно може виникнути помилка. Деякі методи спеціально позначають ключовим словом **throws**, тоді такі методи обов’язково повинні бути викликані з використанням **try…catch**, або ж у методах, що теж мають позначку **throws**. У другому випадку помилка буде передаватись у метод «поверхом вище» і має оброблена в ньому.

Зверніть увагу, що в коді телеграм-бота у **try…catch** загорнуто основний код, саме тому, що в ньому можуть виникнути помилки і вони відповідно позначені **throws**.

Важливим моментом також є параметр блоку catch. Об’єкт помилки дозволяє отримати інформацію про помилку і відповідно до цього прийняти рішення щодо того, як саме на неї відреагувати. Найпростіше, що можна зробити – просто вивести інформацію про помилку в консоль одним із методів наведених у прикладі вище.

**Mirror Bot**

**Завдання 1**. Створіть бота, що відповідатиме на будь-яке текстове повідомлення тим самим повідомленням.

Усе що нам необхідно змінити у попередньому боті, так це метод *onUpdateReceived*. Тепер він повинен надсилати відповідь не просто в консоль, а в чат звідки було отримано апдейт.

**Homework**

**Завдання 1.** Повіторіть вдома код найпростішого телеграм-бота, або ж Mirror Bot’а. Переконайтесь, що все працює правильно.