**Lesson 25. Telegram Bot**

Третя чверть нашого курсу позаду і як завжди маємо перевірити себе та з’ясувати свій прогрес, що зрозумілось добре, а над чим ще необхідно зупинитись додатково. Як ми твердо переконані, перевірити себе можливо тільки працюючи самостійно і саме цьому присвячений даний урок.

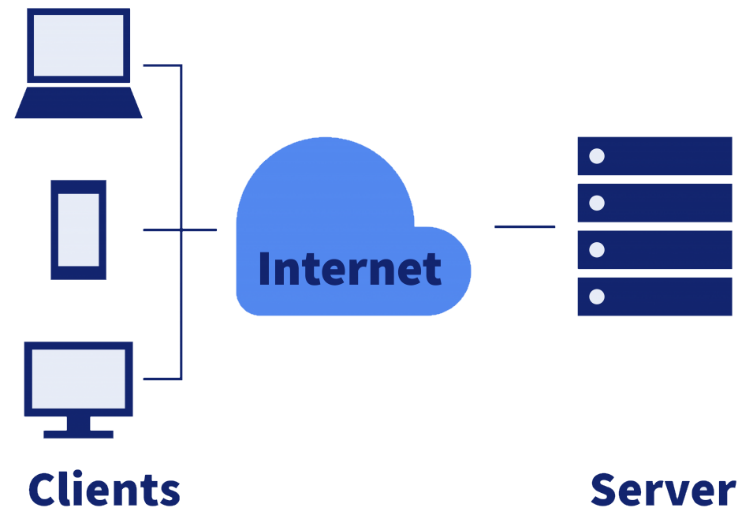
Робота складається із двох частин:

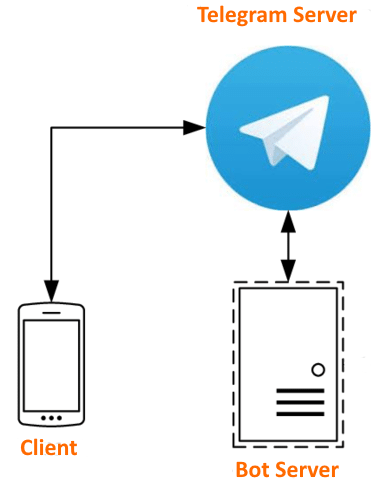
**Clients and Servers**

Перш ніж говорити про те, як працюють телеграм боти, необхідно трошки детальніше розібратись із тим, як працює інтернет загалом.

Усі пристрої підключені до мережі інтернет можна поділити на **клієнти** і **сервери**. Клієнти – всі користувачі інтернету, а сервери – машини, що їх обслуговують.

Коли ми намагаємось потрапити на якусь веб-сторінку (наприклад instagram), наш комп'ютер (планшет, смартфон тощо) надсилає запит на сервер, що зберігає цю веб-сторінку. Сервер обробляє запит та надсилає назад веб-сторінку нашому комп'ютеру, де ми її переглядаємо за допомогою веб-браузера.



Цей процес дозволяє нам переглядати веб-сторінки, відправляти електронні листи та користуватися багатьма іншими інтернет-сервісами.

У випадку із Telegram-ботами сервера матимемо цілих два:

* Telegram-сервер – для зв’язку із клієнтами (вже готовий, створений самим телеграмом);
* Bot-сервер – для забезпечення головної логіки та функціоналу бота (маємо запрограмувати самі, та налаштувати зв’язок із Telegram-сервером).

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>).

**Create New Bot**

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.

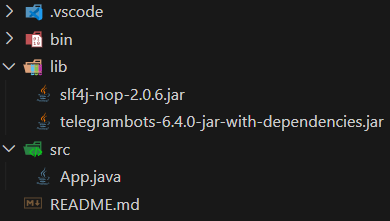
**Creating Bot**

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

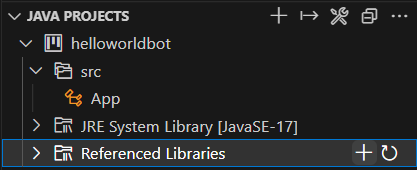
**Крок 1**. Створюємо звичайний проєкт Java і додаємо до нього необхідні бібліотеки.

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:

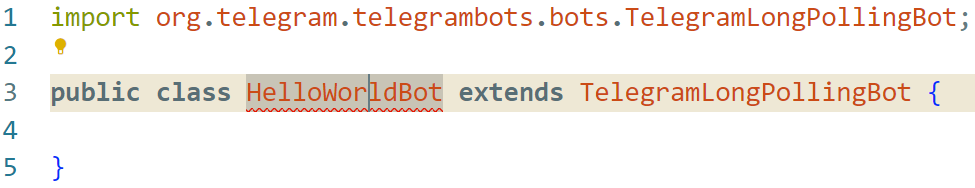


1. Можемо також вказати шлях до файлів використовуючи Referenced Libraries у вкладці Java Projects. Тиснемо + і шукаємо файли:



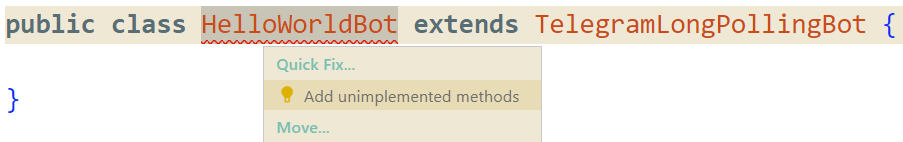
**Крок 2**. Створюємо клас у якому буде реалізована головна логіка нашого бота.

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

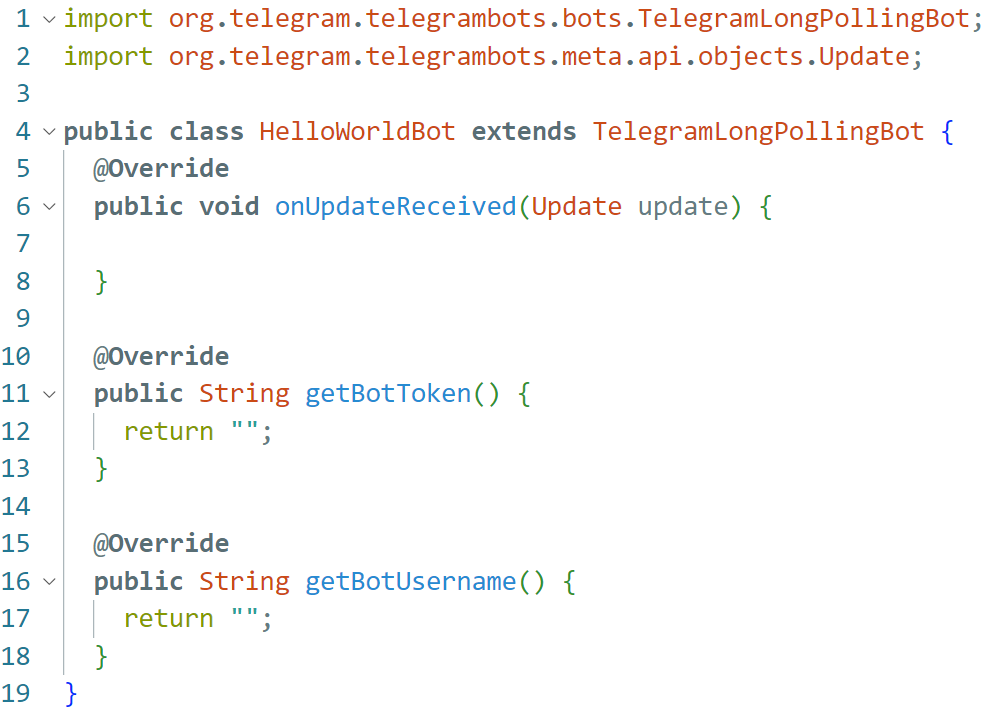


Помилка, яку ми отримали внаслідок наслідування, виникає тому, що клас TelegramLongPollingBot містить абстрактні методи, тобто методи, які не мають реалізації у батьківському класі, але які мають бути обов’язково реалізовані у дочірніх класах.

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:

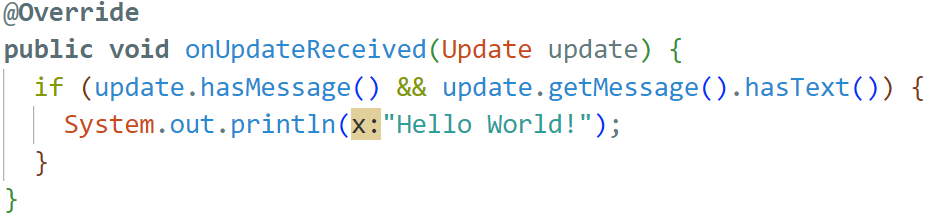
* *getBotToken* – повертає рядок із токеном нашого бота;
* *getBotUsername* – повертає рядок із юзернеймом нашого бота;
* *onUpdateReceived* – містить головну логіку бота. Даний метод викликається автоматично кожного разу, коли сервер Telegram сповіщає йому про зміни, які відбулись у бота (користувач 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, яку було перехоплено в try.

**Приклад**. Створіть

Ключові слова спробувати та зловити поділяються на пари:Створіть **один об’єкт класу** і протестуйте роботу усіх його власних, або переписаних методів.

**Additional Task**

**Завдання 3**. Створіть клас **Customer** (клієнт), що унаслідує клас User. Даний клас повинен додатково мати одне поле:

1. *balance* (кількість коштів на рахунку, грн, дефолтне значення – 0, не може мати значень менше 0).

Клас повинен мати наступні **методи**:

* *getBalance* (повертає поточне значення коштів на рахунку);
* *debit* (додає певну кількість коштів на рахунок);
* *credit* (знімає певну кількість коштів з рахунку, якщо запитаних коштів недостатньо, то вони не знімаються взагалі, а в консоль виводиться відповідне повідомлення);
* *printInfo* (успадкований від User, але виводить інформацію про працівника у консоль в наступному вигляді: *Customer [Jack Doe, email: customer@gmail.com, balance: 300.0 UAH]*).

Клас повинен мати тільки один **конструктор**:

* З одним одним параметром, через який при створенні об’єкта вказується email;

Тепер може порахувати бали і зробити висновки:

**Additional Task**

**Test Part** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / (20)

**Practice Part** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / (10)

**Result** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / (30)

**Висновок.** Підкресліть той варіант, що на вашу думку є найближчий до правди:

* Все круто, мій результат ідеальний;
* Я майже з усім справився/справилась, трохи допрацюю і все буде ок;
* Ну таке… Я багато чого не розумію, треба попрацювати, щоб наздогнати;
* Все погано, я засмучений/засмучена, здається, самостійно працювати не можу.