UNIVERSITÀ DEGLI STUDI DI MESSINA

DIPARTIMENTO DI SCIENZE MATEMATICHE E INFORMATICHE,
SCIENZE FISICHE E SCIENZE DELLA TERRA

Corso di Laurea Triennale in Informatica

HomeButler: Implementazione di un Telegram Bot in Java

Docente:
Prof. Antonino Galletta

Studenti:
Davide Capra, 515886
Federico Sciuto, 517075

ANNO ACCADEMICO 2022/2023

Indice

1 Introduzione

Questo scritto ha lo scopo di fornire una relazione sull'implementazione del Home_Butler, un bot di Telegram sviluppato completamente in Java, uno dei linguaggi di programmazione orientati agli oggetti più famosi.Il codice sorgente del progetto è disponibile su GitHub, nella repository Home_Butler.

1.1 Telegram

Telegram è un software di messaggistica istantanea, caratterizzato dall'essere open-source per quanto riguarda il lato client e proprietario per il lato server. Grazie a questa piattaforma, gli utenti possono comunicare attraverso chat private o di gruppo con i loro contatti, accedere a canali che permettono la lettura a tutti i partecipanti e la scrittura solo agli amministratori, e interagire con i bot. Questi ultimi rappresentano delle chat particolari, il cui interlocutore è un'entità software automatizzata che solitamente offre servizi specializzati all'utente. Per consentire l'implementazione di applicazioni che si interfacciano alla piattaforma, l'azienda proprietaria di Telegram mette a disposizione sia delle API pubbliche accessibili attraverso chiamate RESTful, sia dei driver ufficiali per vari linguaggi di programmazione.

1.2 Java

Java è un linguaggio di programmazione orientato agli oggetti che ha visto la luce negli anni '90. Il suo obiettivo principale era quello di scrivere codice portabile e indipendente dall'architettura sulla quale veniva eseguito. A tale scopo, non fu creato solo il linguaggio in sé, ma anche un intero ambiente di esecuzione chiamato JRE (Java Runtime Environment). Questo ambiente include una macchina virtuale chiamata JVM (Java Virtual Machine) che interpreta un linguaggio intermedio tra il codice sorgente e l'Assembly, chiamato bytecode. Prima di essere eseguito, il codice sorgente viene compilato in bytecode, in modo che possa essere eseguito su diverse architetture, a condizione che la corretta versione del JDK (Java Development Kit) sia installata sulla macchina. Il JDK è una suite di software che include la JRE e alcune utility per lo sviluppo di software. Il paradigma orientato agli oggetti rende Java altamente versatile per lo sviluppo di codice applicativo. Grazie alla gestione astratta e ad alto livello di costrutti e strutture dati sotto forma di oggetti (istanze di classi), il programmatore può concentrarsi sulla funzionalità e la modularità del software, lasciando ai livelli sottostanti la gestione di aspetti come la memoria. Ad esempio, Java dispone di un tool chiamato garbage collector, che si occupa di liberare la memoria contenente dati non più utilizzati e/o dimenticati.

2 Requisiti

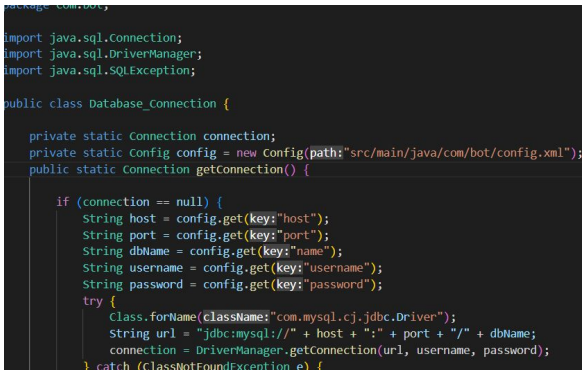
Per lo sviluppo di questo progetto e la conseguente valutazione della conoscenza della programmazione ad oggetti sono stati fissati alcuni requisiti da rispettare. Di seguito i requisiti rispettati:

- Implementazione delle caratteristiche della OOP (incapsulamento e information hiding, polimorfismo, ereditarietà);
- Utilizzo delle interfacce di Java;
- Gestione delle eccezioni;
- Interfacciamento con un database;
- Configurazione attraverso un file XML;
- Building del progetto con Maven.

2.1 Caratteristiche OOP

2.1.1 Incapsulamento e information hiding

Il concetto di *incapsulamento* si riferisce alla particolare caratteristica degli oggetti di contenere al loro interno, o "incapsulare", sia il loro stato che il loro comportamento, ossia i campi e i metodi definiti nella loro classe.



L'*information hiding*, o "nascondimento delle informazioni", è il principio per cui si preferisce che alcune informazioni di un oggetto non siano accessibili direttamente dall'esterno del suo contesto, ad esempio tramite la modalità di accesso *dotted notation*, che utilizza la sintassi `<oggetto>.<campo>` per accedere allo stato dell'oggetto. Invece, si definiscono dei metodi appositi, chiamati "setter" e "getter", che ammettono questa notazione e restituiscono un riferimento al campo al quale si vuole accedere. La stessa politica si applica ai metodi che non si desidera che siano direttamente utilizzabili.

Per garantire una distinzione tra le informazioni accessibili e quelle inaccessibili, Java utilizza due parole chiave: *public* e *private*. La presenza della parola chiave *public* fa sì che i campi e i metodi che la utilizzano nella loro definizione possano essere richiamati da altre classi tramite la dotted notation, mentre questo non è possibile con la parola chiave *private*.

2.1.2 Ereditarietà

L'ereditarietà è un concetto fondamentale della programmazione ad oggetti che permette di definire una classe che eredita da un'altra, chiamata "superclasse", i campi e i metodi comuni ad altre classi figlie. Le superclassi rappresentano oggetti generici e spesso non istanziabili per mancanza di informazioni caratterizzanti, così

che le classi figlie possano ereditare le informazioni di base e aggiungere quelle caratterizzanti.

L'ereditarietà rende il codice riutilizzabile e permette un ragionamento sugli oggetti ad un livello più astratto e comparabile con la realtà. Tuttavia, è importante sottolineare che nonostante l'ereditarietà, le superclassi non permettono l'accesso ai campi "private" alle classi figlie, né la derivazione di metodi con la medesima visibilità. In altre parole, i campi privati della superclasse rimangono accessibili solo dalla superclasse stessa e non sono visibili dalle sue sottoclassi. Lo stesso vale per i metodi privati della superclasse. Questo garantisce l'incapsulamento dei dati e la loro protezione da accessi non autorizzati.

2.1.3 Polimorfismo

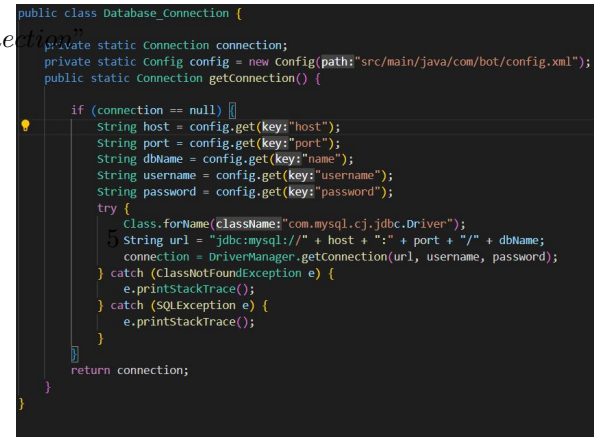
Il termine *polimorfismo* è sinonimo di mutabilità del codice, e per tale ragione ad esso sono associati due termini più specifici: *overload* e *override*.

Il concetto di *overload* si riferisce alla situazione in cui due metodi di una stessa classe hanno lo stesso nome ma firme differenti. La firma di un metodo è l'insieme di informazioni che lo identificano univocamente all'interno della classe, ovvero il nome del metodo e la lista dei tipi dei suoi parametri, nell'ordine in cui sono dichiarati. In questo modo, i due metodi sono considerati entità separate, anche se condividono lo stesso nome. Questo meccanismo consente una migliore leggibilità del codice nei casi in cui ci siano metodi che devono risolvere lo stesso problema, ma con dati iniziali differenti.

L'*override*, invece, si riferisce al meccanismo per cui una classe derivata modifica l'implementazione di un metodo della superclasse, mantenendo la stessa firma ma creando comunque un'entità separata definita nella classe derivata. Questo meccanismo consente alle classi figlie di personalizzare il comportamento ereditato dalla superclasse.

2.2 Database

Per l'implementazione delle funzionalità previste dal bot, è stato necessario appoggiarsi ad un database, e per tale scopo è stato scelto MySQL..



Questo codice definisce una classe chiamata "*Database_Connection*" che gestisce la connessione a un database MySQL. La classe ha un metodo statico chiama-

to "*getConnection*()" che restituisce un oggetto di tipo "Connection", rappresentante la connessione al database. In particolare, il metodo "*getConnection*()" controlla se la variabile "connection" è null (ovvero se non è ancora stata stabilita una connessione al database), e se sì, legge i parametri di connessione dal file di configurazione "config.xml" (che viene caricato attraverso l'oggetto "config" di tipo "Config"), carica il driver JDBC di MySQL e stabilisce la connessione tramite la chiamata a "*DriverManager.getConnection*()". Infine, restituisce l'oggetto "Connection" rappresentante la connessione.

2.3 XML

Il linguaggio XML è nato per la stesura di file di testo capaci di immagazzinare dati per la comunicazione tra componenti software. Nel progetto è stato utilizzato un file `config.xml` contiene le informazioni necessarie per la connessione al database, come l'indirizzo del server, la porta, il nome del database, il nome utente e la password.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <db>
        <host>localhost</host>
        <port>3306</port>
        <name>telegrambot</name>
        <username>root</username>
        <password></password>
    </db>
</config>
```

2.4 Maven

Maven è uno strumento di gestione e building automatico di progetti Java, ospitato dalla nota azienda di software open-source Apache.

Questo tool permette il building automatico tramite un file XML denominato `pom.xml` (*Project Object Model*) nel quale son presenti tutte le informazioni sul progetto e sulle dipendenze.

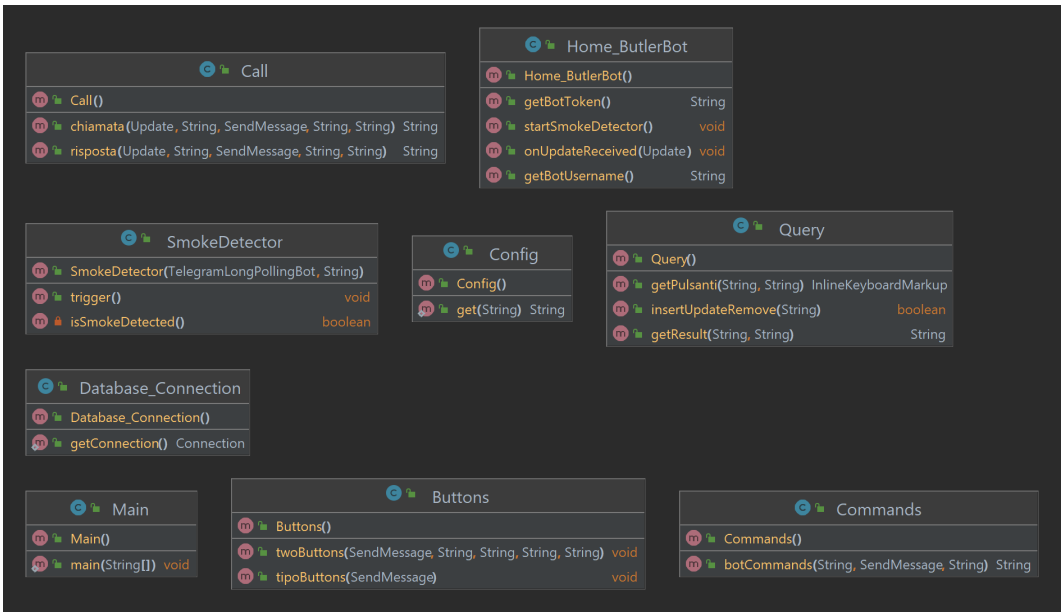
Di seguito una parte del file `pom.xml` del progetto HomeButler, nella quale è possibile notare l'inclusione delle API di Telegram e di MySQL.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.bot</groupId>
  <artifactId>telegrambot</artifactId>
  <version>1.0-SNAPSHOT</version>

  <name>telegrambot</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.telegram</groupId>
      <artifactId>telegrambots</artifactId>
      <version>6.5.0</version>
    </dependency>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.32</version>
    </dependency>
```

3 Diagramma delle classi

Il *diagramma* delle classi è un particolare tipo di diagramma, strutturato secondo lo standard UML (*Unified Modeling Language*), che permette una rappresentazione delle classi, o più in generale delle componenti, di un certo linguaggio di programmazione ad oggetti, includendo i campi e i metodi di ognuno di questi, indicandone la visibilità e la signature, ove possibile; inoltre, è possibile indicare le relazioni di ereditarietà e di implementazione delle interfacce.

Di seguito il diagramma delle classi del HomeButler, generato automaticamente dall'IDE *IntelliJ*, della suite JetBrains.



4 Funzionalità

Il bot è stato progettato per poter gestire una smarthome, e per raggiungere questo fine si sono supposte delle funzionalità che siano utili per raggiungere questo scopo:

- *Gestione degli Elettrodomestici*, il bot può accendere o spegnere gli elettrodomestici, come la lavatrice, il forno, il frigorifero, ecc.;
- *Controllo delle luci*, il bot può accendere o spegnere le luci in base alle richieste dell'utente.;
- *Notifiche di allarme*, il bot può inviare notifiche all'utente in caso di situazioni di emergenza, come un allarme antincendio o un'intrusione.

4.1 Gestione degli Elettrodomestici e Luci

Questa classe, chiamata Buttons, contiene due metodi per la creazione di tasti inline keyboard, che possono essere utilizzati per inviare messaggi a utenti di Telegram

tramite il bot. Il primo metodo, twoButtons(), accetta cinque parametri: il primo parametro è un oggetto di tipo SendMessage, che rappresenta il messaggio da inviare; i successivi quattro parametri sono coppie di stringhe che rappresentano l'etichetta del pulsante (visibile all'utente) e il valore di callback associato al pulsante. Il metodo crea quindi un oggetto di tipo InlineKeyboardMarkup per rappresentare la tastiera, e aggiunge due pulsanti a una riga della tastiera. I valori di etichetta e callback per i pulsanti vengono presi dai parametri passati al metodo. Infine, il metodo aggiunge la tastiera al messaggio di risposta tramite il metodo setReplyMarkup() dell'oggetto SendMessage. Il secondo metodo, tipoButtons(), accetta un parametro di tipo SendMessage. Questo metodo crea quattro pulsanti con etichette "Elettrodomestici", "Finestre", "Luci", e "Riscaldamenti", e associa a ciascun pulsante un valore di callback corrispondente. Questi pulsanti vengono quindi organizzati in due righe di due pulsanti ciascuna, e la tastiera risultante viene aggiunta al messaggio di risposta tramite setReplyMarkup(). Entrambi i metodi utilizzano classi del pacchetto Telegram Bot API (org.telegram.telegrambots.meta.api), in particolare SendMessage, InlineKeyboardMarkup e InlineKeyboardButton, per creare e aggiungere i pulsanti alla tastiera.

```
public class Buttons {

    public void twoButtons(SendMessage sendMessage, String pulsante1a, String pulsante1b, String pulsante2a, String pulsante2b) {

        InlineKeyboardMarkup keyboard = new InlineKeyboardMarkup();
        List<List<InlineKeyboardButton>> buttons = new ArrayList<>();
        List<InlineKeyboardButton> row = new ArrayList<>();
        InlineKeyboardButton button1 = new InlineKeyboardButton();
        InlineKeyboardButton button2 = new InlineKeyboardButton();
        button1.setText(pulsante1a);
        button2.setText(pulsante2a);
        button1.setCallbackData(pulsante1b);
        button2.setCallbackData(pulsante2b);
        row.add(button1);
        row.add(button2);
        buttons.add(row);
        keyboard.setKeyboard(buttons);
        sendMessage.setReplyMarkup(keyboard);
    }
```

4.2 Notifiche di allarme

Per queste funzionalità abbiamo utilizzato una classe che rappresenta un "rilevatore di fumo" che, ad intervalli di tempo regolari (in questo caso ogni 5 secondi), controlla se è stata rilevata la presenza di fumo. Il "rilevatore di fumo" viene creato attraverso il costruttore, che prende come argomenti un oggetto di tipo "TelegramLongPollingBot" (che rappresenta il bot di Telegram con cui inviare messaggi) e una stringa "chatId" (che identifica la chat in cui inviare i messaggi). La funzione "trigger" avvia un timer che esegue periodicamente un'operazione, in questo caso la verifica della presenza di fumo attraverso la funzione "isSmokeDetected". Se la funzione ritorna "true" (con una probabilità del 40

```java
public class SmokeDetector {

    private final TelegramLongPollingBot bot;
    private final String chatId;

    public SmokeDetector(TelegramLongPollingBot bot, String chatId) {
        this.bot = bot;
        this.chatId = chatId;
    }

    public void trigger() {
        Timer timer = new Timer();
        timer.schedule(new TimerTask() {

            @Override
            public void run() {

                if (isSmokeDetected()) {
                    SendMessage sendMessage = new SendMessage();
                    sendMessage.setChatId(chatId);
                    sendMessage.setText(text:"ALLARME!!! Fumo rilevato!");
                    try {
                        bot.execute(sendMessage);
                    } catch (TelegramApiException e) {
                        e.printStackTrace();
                    }
                }
            }
        }, delay:0, period:5000);
    }
```